Roman Wyrzykowski
Jack Dongarra
Norbert Meyer
Jerzy Waśniewski (Eds.)

LNCS 3911

Parallel Processing and Applied Mathematics

6th International Conference, PPAM 2005
Poznań, Poland, September 2005
Revised Selected Papers

Springer

Lecture Notes in Computer Science 3911

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

Editorial Board

Roman Wyrzykowski Jack Dongarra
Norbert Meyer Jerzy Waśniewski (Eds.)

Parallel Processing and Applied Mathematics

6th International Conference, PPAM 2005
Poznań, Poland, September 11-14, 2005
Revised Selected Papers

Springer

Volume Editors

Roman Wyrzykowski
Częstochowa University of Technology
Department of Computer and Information Sciences
Dąbrowskiego 73, 42-200 Częstochowa, Poland
E-mail: roman@icis.pcz.pl

Jack Dongarra
University of Tennessee
Computer Science Department
1122 Volunteer Blvd., Knoxville, TN 37996-3450, USA
E-mail: dongarra@cs.utk.edu

Norbert Meyer
Poznań Supercomputing and Networking Center
Noskowskiego 10, 61-704 Poznań, Poland
E-mail: meyer@man.poznan.pl

Jerzy Waśniewski
Technical University of Denmark
Department of Informatics and Mathematical Modelling
Richard Petersens Plads, Building 321, 2800 Kongens Lyngby, Denmark
E-mail: jw@imm.dtu.dk

Library of Congress Control Number: 2006925464

CR Subject Classification (1998): D, F.2, G, B.2-3, C.2, J.2

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

ISSN 0302-9743
ISBN-10 3-540-34141-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34141-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11752578 06/3142 5 4 3 2 1 0

Preface

This volume comprises the proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics - PPAM 2005, which was held in Poznań, the industrial, academic and cultural center in the western part of Poland, during September 11–14, 2005. It was organized by the Department of Computer and Information Sciences of the Częstochowa University of Technology, with the help of Poznań Supercomputing and Networking Center. The main organizer was Roman Wyrzykowski.

PPAM is a biennial conference organized since 1994. Five previous events have been held in different places in Poland. The proceedings of the last two conferences were published by Springer in the *Lecture Notes in Computer Science* series (Nałęczów, 2001, vol.2328; Częstochowa, 2003, vol.3019).

The PPAM conferences have become an international forum for exchanging ideas between researchers involved in parallel and distributed computing, including theory and applications, as well as applied and computational mathematics. The focus of PPAM 2005 was on grid computing. The main idea behind this decision was to foster communication and cooperation between the grid application users, application developers and grid middleware developers, to identify the key application requirements and scenarios on the grid, to gather information about tools and toolkits, and to broaden the grid community by encouraging new users to take advantage of grid technologies.

This meeting gathered around 200 participants from 33 countries. A strict refereeing process resulted in acceptance of 130 contributed presentations, while approximately 38% of the submissions were rejected. It is worth mentioning that the conference was visited by both the research community and industry representatives.

Regular tracks of the conference covered important fields of parallel/distributed/grid computing and applied mathematics such as:

- Parallel and distributed architectures
- Parallel and distributed non-numerical algorithms
- Performance analysis, prediction and optimization
- Grid programming
- Tools and environments for clusters and grids
- Numerical and non-numerical applications of parallel/distributed/grid computing
- Evolutionary computing
- Parallel data mining
- Parallel numerics
- Mathematical and computing methods

The plenary and invited talks were presented by:

- Ben Bennett from Intel (USA)
- Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory (USA)
- Geoffrey Fox from Indiana University (USA)
- Jacek Gondzio from the University of Edinburgh, Scotland (UK)
- Rich L. Graham from Los Alamos National Laboratory (USA)
- Kate Keahey from Argonne National Laboratory (USA)
- Eric Kronstadt from IBM T.J. Watson Research Center (USA)
- Bolesław Szymański from Rensselaer Polytechnic Institute (USA)
- Ziga Turk from the University of Ljubljana (Slovenia)
- Jerzy Waśniewski from the Technical University of Denmark (Denmark)

Important and integral parts of the PPAM 2005 conference were the workshops:

- The Second Grid Application and Middleware Workshop - GAMW 2005 organized by Ewa Deelman from the USC Information Sciences Institute (USA) and Norbert Meyer from the Poznań Supercomputing and Networking Center (Poland)
- The Second Grid Resource Management Workshop - GRMW 2005 organized by Jarek Nabrzyski from the Poznań Supercomputing and Networking Center (Poland) and Ramin Yahyapour from the University of Dortmund (Germany)
- Workshop on Large Scale Computations on Grids organized by Przemyslaw Stpiczyński from Marie Curie-Skłodowska University in Lublin (Poland), Dana Petcu from the Western University of Timisoara (Romania), and Marcin Paprzycki from SWPS in Warsaw (Poland)
- Workshop on Scheduling for Parallel Computing organized by Maciej Drozdowski from the Poznań University of Technology (Poland)
- Workshop on Language-Based Parallel Programming Models organized by Ami Marowka from the Hebrew University (Israel)
- Workshop on Dependability of the Distributed Systems organized by Jan Kwiatkowski and Piotr Karwaczyński from the Wroclaw University of Technology (Poland)
- Workshop on HPC Linear Algebra Libraries for Computers with Multilevel Memories organized by Jerzy Waśniewski from the Technical University of Denmark (Denmark)
- Workshop on Parallel Bio-Computing organized by David A. Bader from the Georgia Institute of Technology in Atlanta (USA) and Denis Trystram from ID-IMAG in Grenoble (France)

The PPAM 2005 meeting began with four half-day tutorials:

- Using CLUSTERIX: National Cluster of Linux Systems, by the CLUSTERIX Team from Czestochowa University of Technology, Poznań Supercomputing and Networking Center, Gdańsk University of Technology, and Białystok Technical University (Poland)

- Enterprise GRID Solutions: Eliminating Isolated Technology Islands with InfiniBand, by CISCO
- Scientific Programming for Heterogeneous Systems, by Alexey Lastovetsky from the University College Dublin (Ireland) and Alexey Kalinov from the Institute for System Programming in Moscow (Russia)
- Upgrading Cluster Performance with InfiniBand and the Intel MPI Library, by Tom Lehmann from Intel (USA)

The organizers are indebted to the PPAM 2005 sponsors, whose support was vital to the success of the conference. The main sponsor was Intel Corporation. The other sponsors were: IBM Corporation, Optimus S.A., Cisco Systems, and APC Corporation. We thank all members of the International Program Committee, Workshop Program Comittees and additional reviewers for their diligent work in refereeing the submitted papers. Finally, we thank all of the local organizers from the Częstochowa University of Technology and Poznań Supercomputing and Networking Center, who helped us run the event very smoothly. We are especially indebted to Grażyna Kołakowska, Urszula Kroczewska, Konrad Karczewski, Jarosław Żola, from the Częstochowa University of Technology, and Maciej Stroiński, Sławomir Niwiński from Poznań Supercomputing and Networking Center.

We hope that this volume will be useful to you. We would like to invite everyone who reads it to the next conference, PPAM 2007, which will be held on the Baltic Coast in Gdańsk/Sopot (Poland) on September 9-12, 2007 (http://ppam.pcz.pl).

February 2006

Roman Wyrzykowski
Jack Dongarra
Norbert Meyer
Jerzy Waśniewski

Organization

Program Committee

Jan Węglarz	Poznań University of Technology, Poland Honorary Chair
Roman Wyrzykowski	Częstochowa University of Technology, Poland Chair of Program Committee
Bolesław Szymański	Rensselaer Polytechnic Institute, USA Vice-Chair of Program Committee
Peter Arbenz	Swiss Federal Institute of Technology, Switzerland
Piotr Bała	N. Copernicus University, Poland
Radim Blaheta	Institute of Geonics, Czech Academy of Sciences, Czech Republic
Jacek Blażewicz	Poznań University of Technology, Poland
Tadeusz Burczyński	Silesia University of Technology, Poland
Peter Brezany	University of Vienna, Austria
Jerzy Brzeziński	Poznań University of Technology, Poland
Marian Bubak	Institute of Computer Science, AGH, Poland
Raimondas Čiegis	Vilnius Gediminas Technical University, Lithuania
Bogdan Chlebus	University of Colorado at Denver, USA
Zbigniew Czech	Silesia University of Technology, Poland
Sergei Gorlatch	University of Muenster, Germany
Jack Dongarra	University of Tennessee and ORNL, USA
Maciej Drozdowski	Poznań University of Technology, Poland
Andrzej Gościński	Deakin University, Australia
Frederic Guinand	Université du Havre, France
Thomas Fahringer	University of Innsbruck, Austria
Marta Fairen	Universitat Polit. de Catalunya, Barcelona, Spain
Ladislav Hluchy	Slovak Academy of Sciences, Bratislava
Alexey Kalinov	Institute for System Programming, Russia
Ayse Kiper	Middle East Technical University, Turkey
Jacek Kitowski	Institute of Computer Science, AGH, Poland
Erricos Kontoghiorghes	Université de Neuchâtel, Switzerland
Jozef Korbicz	University of Zielona Góra, Poland
Stanislaw Kozielski	Silesia University of Technology, Poland
Dieter Kranzlmueller	Johannes Kepler University Linz, Austria
Henryk Krawczyk	Gdańsk University of Technology, Poland
Piotr Krzyżanowski	University of Warsaw, Poland
Jan Kwiatkowski	Wrocław University of Technology, Poland
Alexey Lastovetsky	University College Dublin, Ireland

Vyacheslav Maksimov	Ural Branch, Russian Academy of Sciences, Russia
Tomas Margalef	Universitat Autonoma de Barcelona, Spain
Ami Marowka	Hebrew University, Israel
Norbert Meyer	PSNC, Poznań, Poland
Jarek Nabrzyski	PSNC, Poznań, Poland
Marcin Paprzycki	SWPS, Warsaw, Poland
Dana Petcu	Western University of Timisoara, Romania
Edwige Pissaloux	Universite de Rouen, France
Jacek Rokicki	Warsaw University of Technology, Poland
Leszek Rutkowski	Częstochowa University of Technology, Poland
Yousef Saad	University of Minnesota, USA
Franciszek Seredyński	Polish Academy of Sciences, Warsaw, Poland
Robert Schaefer	Institute of Computer Science, AGH, Poland
Norbert Sczygiol	Częstochowa University of Technology, Poland
Jurij Silc	Jozef Stefan Institute, Slovenia
Peter M.A. Sloot	University of Amsterdam, The Netherlands
Przemyslaw Stpiczyński	UMCS, Lublin, Poland
Domenico Talia	University of Calabria, Italy
Andrei Tchernykh	CICESE, Ensenada, Mexico
Sivan Toledo	Tel-Aviv University, Israel
Roman Trobec	Jozef Stefan Institute, Slovenia
Denis Trystram	ID-IMAG, Grenoble, France
Marek Tudruj	Polish Academy of Sciences, Warsaw, Poland
Pavel Tvrdik	Czech Technical University, Prague, Czech Republic
Jens Volkert	Johannes Kepler University Linz, Austria
Jerzy Waśniewski	Technical University of Denmark, Denmark
Bogdan Wiszniewski	Gdańsk University of Technology, Poland
Krzysztof Zieliński	Institute of Computer Science, AGH, Poland
Jianping Zhu	University of Akron, USA

Table of Contents

Parallel and Distributed Architectures

Paralel and Distributed Non-numerical Algorithms

Performance Analysis, Prediction and Optimization

Grid Programming

Tools and Environments for Clusters and Grids

Applications of Parallel/Distributed/Grid Computing

Evolutionary Computing with Applications

Parallel Data Mining

Parallel Numerics

Computer Methods

The Second Grid Application and Middleware Workshop (GAMW'2005)

The Second Grid Resource Management Workshop (GRMW'2005)

Workshop on Large Scale Computations on Grids

Workshop on Scheduling for Parallel Computing

Workshop on Language-Based Parallel Programming Models

Workshop on Dependability of the Distributed Systems

Workshop on HPC Linear Algebra Libraries for Computers with Multilevel Memories

Workshop on Parallel Bio-computing

Multi-version Coherence Protocol for Replicated Shared Objects*

Jerzy Brzeziński, Jacek Kobusiński, and Dariusz Wawrzyniak

Poznan University of Technology, Institute of Computing Science,
Piotrowo 3a, 60-965 Poznań, Poland

Abstract. Sequential consistency is one of the strongest consistency models for replicated shared data. The performance of a system supporting this model is rather low, so weaker models are used to improve the efficiency. However, the construction of application for weaker models is more complicated. The idea of this approach is to provide the data to the application along with the consistency model to which they conform. This allows the application to decide about the quality of accessed data.

1 Introduction

Sequential consistency [1] is one of the strongest consistency models for replicated shared data. It guarantees the correctness of any concurrent algorithm designed for a shared memory multiprocessor. However, the performance of a distributed system supporting sequential consistency is rather low, because additional synchronisation is required when the data is accessed to ensure that the changes are observed in the same order on each replica. An approach of this kind is called pessimistic replication [2] in contrast to optimistic replication, which allows for weaker consistency models [3, 4, 5, 6], resulting in faster update of replicas, thereby improving the performance.

In the case of weaker models sequential consistency is usually considered to be a correctness criterion in the sense that an execution under a weaker model is correct if it is equivalent to the execution under sequential consistency. To ensure the equivalence some restrictions on memory access are imposed, e.g. data-race freedom or concurrent-writes freedom for causal consistency [6]. However, this equivalence is not the necessary condition for the program correctness. Thus, the question is what consistency conditions must be met to ensure that the data are appropriate from the viewpoint of application requirements.

The idea of this approach is to provide the data to the application as soon as possible along with the consistency model to which they conform. In other words, the system keeps as many different versions of a given object as the number of different states resulting from various consistency models. This way an application process can flexibly decide about the quality of accessed data, balancing between efficiency and transparency.

* This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 3 T11C 073 28.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1–8, 2006.

2 Consistency Models

The system consists of a set of n sequential processes $P = \{p_1, p_2, \ldots, p_n\}$ interacting via a set of m shared read/write objects $X = \{x_1, x_2, \ldots, x_m\}$. A process can issue two kinds of operations: reads and writes. A write operation issued by p_i is denoted as $w_i(x)v$, where $x \in X$ is the object and v is the value to be written. Similarly, $r_i(x)v$ denotes a read operations that returns a value v.

Each process observes the operations in the system in some total order called *view*. As the processes are sequential, the view must satisfy a condition called *legality*, which states that a read operation cannot return an overwritten value.

Definition 1 (Legal view). *Let O be a set of operations in a DSM system, and $\rightarrowtail$ is a total order relation. Legal view is the set O ordered by the relation $\rightarrowtail$, if the following condition is satisfied*[1]*:*

$$\underset{w(x)v, r(x)v \in O}{\forall} \left[w(x)v \rightarrowtail r(x)v \wedge \underset{o(x)u \in O}{\nexists} (u \neq v \wedge w(x)v \rightarrowtail o(x)u \rightarrowtail r(x)v) \right]$$

Depending on the consistency model, legal views satisfy additional conditions, based on a *local order* relation, a *process order* relation, or a *causal order* relation. The relations concern the order of issue. Local order for a process p_i, denoted $\overset{\text{PO}}{\rightarrow}_i$, is the order in which operations are issued by p_i. Process order $\overset{\text{PO}}{\rightarrow} = \bigcup_{1 \leq i \leq n} \overset{\text{PO}}{\rightarrow}_i$. Causal order, denoted $\overset{\text{CO}}{\rightarrow}$, is defined as the transitive closure of process order and *read-from order*, which states that $\underset{x \in X}{\forall} w(x)v \overset{\text{CO}}{\rightarrow} r(x)v$.

In the definitions of consistency models, it is assumed that there is a legal view for each process, which comprises the set of all write operations and the operations issued by this process. Let OW denote the set of all write operations in the system, OR denote the set of all read operations in the system, and O_i denote the set of operations issued by p_i. In general, the view for each process may be different, so for p_i it is denoted $\rightarrowtail_i$, and it determines the order of operations from the set $OW \cup O_i$. In the case of sequential consistency [1], the views preserve process order, and all write operations appear in the same order in the view for each process. Causal consistency [6] guarantees only the preservation of causal order, and Pipelined RAM (PRAM) [3] yet less — process order. Cache consistency [4, 5] holds the same order of write operations on a given object in each view. Processor consistency [4, 5] joins the properties offered by cache consistency and PRAM consistency. Local consistency means that the operations of a given process appear in the issue order only in its own view.

3 Supporting Multiple Consistency Models

The coexistence of multiple consistency models in the execution of a parallel program raises the problem of consistency specification. Consistency requirements can be specified along with the object definition or at the issue of operation (i.e. when a given object is accessed). In the case of read/write objects the

[1] For the sake of simplicity it is assumed that each operation writes a unique value.

consistency requirement can concern read or write operations. In hybrid consistency [7] both read and write operations can be labelled as weak or strong. Strong operations force synchronisation that ensures total order of these operations. Mixed consistency [8] requires read operations to be labelled, but allows for two models only: PRAM consistency and causal consistency. Complex consistency [9] supports sequential consistency, causal consistency, PRAM consistency, processor consistency and cache consistency. The model is specified with write operations, which allows some optimisation in message ordering.

This approach is oriented at making the changes available as soon as possible under the conditions defined by consistency models. The weaker consistency model the sooner the changes are available. To this end, different versions of each object are maintained, where each version corresponds to one model from local consistency to sequential consistency, including PRAM, cache, processor, and causal consistency. Consequently, there is a separate view for each version, denoted as $\overset{\text{lc}}{\rightarrowtail}$, $\overset{\text{pm}}{\rightarrowtail}$, $\overset{\text{ce}}{\rightarrowtail}$, $\overset{\text{pc}}{\rightarrowtail}$, $\overset{\text{cs}}{\rightarrowtail}$, $\overset{\text{sq}}{\rightarrowtail}$, respectively. Every modification of a shared object (write operation) appears in each consistency version. Therefore, the consistency model is specified at a read operation. For the sake of formal definition, the set of read operations issued by p_i is separated into subsets corresponding to the consistency models in which the operations are issued, and denoted OR_i^{lc}, OR_i^{pm}, OR_i^{ce}, OR_i^{pc}, OR_i^{cs}, OR_i^{sq}, respectively.

Definition 2 (Multi-version consistency).

(a) $\underset{w1,w2\in OW}{\forall} \left(\underset{1\le i\le n}{\forall} w1 \overset{\text{sq}}{\rightarrowtail}_i w2 \vee \underset{1\le i\le n}{\forall} w2 \overset{\text{sq}}{\rightarrowtail}_i w1 \right)$

(b) $\underset{x\in X}{\forall} \underset{w1,w2\in OW|\{x\}}{\forall} \left(\underset{1\le i\le n}{\forall} w1 \overset{\text{pc}}{\rightarrowtail}_i w2 \vee \underset{1\le i\le n}{\forall} w2 \overset{\text{pc}}{\rightarrowtail}_i w1 \right)$

(c) $\underset{x\in X}{\forall} \underset{w1,w2\in OW|\{x\}}{\forall} \left(\underset{1\le i\le n}{\forall} w1 \overset{\text{ce}}{\rightarrowtail}_i w2 \vee \underset{1\le i\le n}{\forall} w2 \overset{\text{ce}}{\rightarrowtail}_i w1 \right)$

(d) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{sq}}}{\forall} \left(o1 \overset{\text{CO}}{\rightarrow} o2 \Rightarrow o1 \overset{\text{sq}}{\rightarrowtail}_i o2 \right)$

(e) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{cs}}}{\forall} \left(o1 \overset{\text{CO}}{\rightarrow} o2 \Rightarrow o1 \overset{\text{cs}}{\rightarrowtail}_i o2 \right)$

(f) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{pm}}}{\forall} \left(o1 \overset{\text{PO}}{\rightarrow} o2 \Rightarrow o1 \overset{\text{pm}}{\rightarrowtail}_i o2 \right)$

(g) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{pc}}}{\forall} \left(o1 \overset{\text{PO}}{\rightarrow} o2 \Rightarrow o1 \overset{\text{pc}}{\rightarrowtail}_i o2 \right)$

(h) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{ce}}}{\forall} \left(o1 \overset{\text{PO}}{\rightarrow}_i o2 \Rightarrow o1 \overset{\text{ce}}{\rightarrowtail}_i o2 \right)$

(i) $\underset{1\le i\le n}{\forall} \underset{o1,o2\in OW\cup OR_i^{\text{lc}}}{\forall} \left(o1 \overset{\text{PO}}{\rightarrow}_i o2 \Rightarrow o1 \overset{\text{lc}}{\rightarrowtail}_i o2 \right)$

4 Coherence Protocol

To maintain consistency the processes exchange update messages. The update messages are sent to other processes as a result of a write operation. After an

update message is received, it is buffered and referenced in a number of queues before it is applied (i.e. made available). It can also be referenced as the update message that defines the current value of an object in a given consistency version. In other words, the most recent update message suitable to each consistency version of a given object is referenced to provide the current value and other data (e.g. time stamp) when a read request is issued by an application process.

The implementation of the models that require total order of some write operations — i.e. sequential consistency, processor consistency, and cache consistency — is based on global numbers. The assignment of these numbers is simplified by the assumption that there is a function $getNextNum$ to generate a consecutive global number for any write operation, and a function $getNextNumX$ to generate consecutive global numbers for write operations on individual objects. To preserve the dependencies between consistency models the numbers for sequential updates (generated by $getNextNum$) must respect causal order, thereby process order, and also the order of operations on individual objects. Besides, each update message is stamped with the virtual time of issue represented by a vector of write numbers that have potential influence on the current state of the sending process. It allows detecting local order and causal order.

The structure of the update message comprises the following attributes: *oid* — the object identifier, *value* — the new value to be written, $globalWriteNum$ — a global unique number of a write operation, $globalWriteNumX$ — a global unique number of a write operation on the given object, $timeStamp[1..n]$ — vector clock representing the logical time of issue (the value of $issueNum$ of the issuing process), *sid* — the identifier of the sender. For the update messages to be applied in a correct order the protocol maintains a number of data, organised into structures within name spaces corresponding to the consistency models:

Sequential consistency (*seq*)**:** $writeNum$ — the number of write operations applied (made available) as sequential so far, *inQueue* — a buffer of UPDATE messages to be applied in the sequentially consistent version;

Causal consistency (*caus*)**:** $procWriteNum[1..n]$ — the array of numbers of write operations by individual processes applied as causal so far, *inQueue* — a buffer of UPDATE messages to be applied in the causal version;

PRAM consistency (*pram*)**:** $procWriteNum[1..n]$ — the array of numbers of write operations by individual processes applied as PRAM so far, *inQueue* — a buffer of UPDATE messages to be applied in the PRAM version;

Processor consistency (*proc*)**:** $procWriteNum[1..n]$ — the array of numbers of write operations issued by individual processes applied as processor consistent so far, *inQueue* — a buffer of UPDATE messages to be applied in the processor version;

Cache consistency (*cache*)**:** *inQueue* — a buffer of UPDATE messages to be applied in the cache consistent version;

Local consistency (*loc*)**:** no control data are required.

Besides, the following global data structures are used: $issueNum[1..n]$ — the array of numbers of write operations by individual processes causally preceding

Algorithm 1. Update of consistency version

```
   Function Update <model> (u) return Boolean
2: if consistency conditions for <model> are satisfied then
     localCopy[u.oid].<model> ← u
4:   update control data
     return True
6: else return False
   end if
```

Table 1. Update condition

model	update condition (line 2 in Alg. 1)
PRAM	$pram.procWriteNum[u.sid] \leftarrow pram.procWriteNum[u.sid] + 1$
causal	$u.timeStamp[u.sid] = caus.procWriteNum[u.sid] + 1 \wedge$ $\forall_{1 \leq k \leq n, k \neq u.sid} \; u.timeStamp[k] \leq caus.procWriteNum[k]$
cache	$u.globalWriteNumX = localCopy[u.oid].cache.globalWriteNumX + 1$
proc.	$u.globalWriteNumX = localCopy[u.oid].proc.globalWriteNumX + 1 \wedge$ $u.timeStamp[u.sid] = proc.procWriteNum[u.sid] + 1$
seq.	$u.globalWriteNum = seq.writeNum + 1$

Table 2. Control data update

model	update operation (line 4 in Alg. 1)
PRAM	$pram.procWriteNum[u.sid] \leftarrow pram.procWriteNum[u.sid] + 1$
causal	$caus.procWriteNum[u.sid] \leftarrow caus.procWriteNum[u.sid] + 1$
cache	nothing
proc.	$proc.procWriteNum[u.sid] \leftarrow proc.procWriteNum[u.sid] + 1$
seq.	$seq.writeNum \leftarrow seq.writeNum + 1$

all reads issued so far, $lastWriteNum$ — the global number of last write operation issued by the process itself, $lastWriteNumX[1..m]$ — the array of global numbers of write operations on individual objects issued by the process itself, $localCopy[1..m]$ — the array of local copies (replicas) of individual objects.

The decision about the application of an UPDATE message depends on the consistency model, according to the template in Alg. 1. Update conditions are assembled in Tab. 1. Thus, after an UPDATE message is received (Alg. 2) it is applied in the locally consistent version (line 2). Then the message is tried to be applied in PRAM and cache consistent version (lines 3 and 9). If the conditions are met, the control data are appropriately updated (Tab. 2), and the stronger models are checked. If the application is unsuccessful, the message is enqueued. Queues are checked to retry applying the messages after some UPDATE in the corresponding version is applied. This is carried out as shown in Alg. 3 for PRAM consistency. The functions for other consistency models are analogous. A detailed description of the protocol is given in [10].

Algorithm 2. UPDATE message handler

```
    on receipt of UPDATE message u
 2: localCopy[u.oid].loc ← u \* local consistency *\
    pramupd ← UpdatePRAM(u)
 4: if pramupd then TryPRAM
      caus.queue ← caus.queue ∪ {u}
 6:   TryCaus
    else pram.queue ← pram.queue ∪ {u}
 8: end if
    cacheupd ← UpdateCache(u)
10: if cacheupd then TryCache
      proc.queue ← proc.queue ∪ {u}
12: else cache.queue ← cache.queue ∪ {u}
    end if
14: if pramupd ∨ cacheupd then procupd ← TryProc
      if procupd then TrySeq
16:   end if
    end if
```

Algorithm 3. PRAM queue handler

```
    Function TryPRAM return Boolean
 2: retv ← False
    repeat
 4:   applied ← False \* in case the queue is empty *\
      for all u ∈ pram.queue do applied ← UpdatePRAM(u)
 6:     if applied then pram.queue ← pram.queue \ {u}
          caus.queue ← caus.queue ∪ {u}
 8:       retv ← True
        end if
10:   end for
    until ¬applied
12: return retv
```

Table 3. Synchronisation conditions

model	synchronisation condition (line 2 in Alg. 5)
cache	$localCopy[oid].cache.globalWriteNumX \geq lastWriteNumX[oid]$
proc.	$localCopy[oid].proc.globalWriteNumX \geq lastWriteNumX[oid]$
seq.	$seq.writeNum \geq lastWriteNum$
others	$True$

Application Program Interface consists of a write procedure (Alg. 4) and read functions suitable for the consistency models (Alg. 5). Cache consistency, processor consistency and sequential consistency require the same order of some write operations. This means that an update message sent by a given process must be appropriately ordered with other update messages before the subsequent result

Algorithm 4. Write operation

```
    Procedure Write(oid, value)
 2: lastWriteNum ← getNextNum
    lastWriteNumX[oid] ← getNextNumX(oid)
 4: issueNum[i] ← issueNum[i] + 1
    u ← UPDATE(oid, value, lastWriteNum, lastWriteNumX[oid], issueNum, i)
 6: localCopy[u.oid].loc ← u
    UpdatePRAM(u)
 8: UpdateCaus(u)
    cache.queue ← cache.queue ∪ {u}
10: TryCache
    send UPDATE message u to all others
```

Algorithm 5. Read operation template

```
    Function Read <model> (oid)
 2: wait until synchronisation condition
       ∀ issueNum[k] ← max(issueNum[k], localCopy[oid].loc.timeStamp[k])
    1≤k≤n
 4: return localCopy[oid].<model>.value
```

is available to the sending process itself. This, in turn, requires the completion of either the write operation sending the update message or the next read operation to be synchronised with the event of delivery. For the purpose of this implementation, a fast write protocol makes sense, because it requires the synchronisation at a read operation (Tab. 3). The own locally consistent version, PRAM consistent version and causal version are updated immediately (lines 6–8 in Alg. 4). Consequently, no synchronisation is needed when these versions are read — the synchronisation condition is true.

5 Conclusions

This paper presents an approach to supporting multiple consistency models of replicas in one system. It supports the most common general access consistency models: sequential consistency, causal consistency, Pipelined RAM consistency, local consistency, processor consistency, and cache consistency. The decision about the expected consistency properties is postponed to the latest possible moment — when shared objects are to be read. This provides flexibility in the trade-off between the quality of replica and efficiency of access. Thus, the consistency model can be considered as a *Quality of Service* aspect of replication.

The coherence protocol for this model is based on the idea of maintaining multiple versions (corresponding to different consistency models) of each shared object. As compared to an update protocol for sequential consistency, the overhead results mainly from maintaining multiple queues of waiting update messages instead of one queue. In other words, the protocol makes available the

updates that are normally hidden in the buffer. As for the objects themselves there is no overhead, because the update messages must be buffered in any case.

The presented approach is suitable for the systems, in which the consistency requirements are not known in advance, e.g. some open systems like grids, Internet services and so on. If the cooperation between processes can be predicted at the design, it is better to use an approach optimised to the cooperation, e.g. hybrid consistency [7], complex consistency [9], or synchronised consistency models, e.g. weak [11], release [12, 13], entry [14], or scope consistency [15].

References

1. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers **C-28**(9) (1979) 690–691
2. Saito, Y., Shapiro, M.: Optimistic replication. Technical Report MSR-TR-2003-60, Microsoft Research (2003)
3. Lipton, R.J., Sandberg, J.S.: PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Dept. of Computer Science, Princeton University (1988)
4. Goodman, J.R.: Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group (1989)
5. Ahamad, M., Bazzi, R.A., John, R., Kohli, P., Neiger, G.: The power of processor consistency (extended abstract). In: Proc. of the 5th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'93). (1993) 251–260
6. Ahamad, M., Neiger, G., Kohli, P., Burns, J.E., Hutto, P.W.: Casual memory: Definitions, implementation and programming. Distributed Computing **9** (1995) 37–49
7. Friedman, R.: Consistency Conditions for Distributed Shared Memories. PhD thesis, Computer Science Department, Technion–Israel Institute of Technology (1994)
8. Agrawal, D., Choy, M., Leong, H.V., Singh, A.K.: Mixed consistency: A model for parallel programming. In: Proc. of the 13th Annual ACM Symp. on Principles of Distributed Computing (PODC'94). (1994) 101–110
9. Wawrzyniak, D.: Complex Consistency Models of Distributed Shared Memory and their Application to Mutual Exclusion Algorithms. PhD thesis, Institute of Computing Science, Poznan University of Technology (2000)
10. Brzeziński, J., Kobusiński, J., Wawrzyniak, D.: Multi-version approach to the consistency of replicated shared objects. Technical Report RA-007/05, Institute of Computing Science, Poznan University of Technology (2005)
11. Adve, S.V., Hill, M.D.: Weak ordering—a new definition. In: Proc. of the 17th Annual Int. Symp. on Computer Architecture (ISCA'90). (1990) 2–14
12. Bennett, J.K., Carter, J.B., Zwaenepoel, W.: Munin: Shared memory for distributed memory multiprocessors. Technical Report COMP TR89-91, Dept. of Computer Science, Rice University (1989)
13. Keleher, P.: Lazy Release Consistency for Distributed Shared Memory. PhD thesis, Department of Computer Science, Rice University (1994)
14. Bershad, B.N., Zekauskas, M.J.: Shared memory parallel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU-CS-91-170, School of Computer Science, Carnegie-Mellon University (1991)
15. Iftode, L., Singh, J.P., Li, K.: Scope consistency: A bridge between release consistency and entry consistency. Theory of Computing Systems **31**(4) (1998) 451–473

Checkpointing Speculative Distributed Shared Memory*

Arkadiusz Danilecki, Anna Kobusińska, and Michal Szychowiak

Institute of Computing Science,
Poznań University of Technology,
Piotrowo 3a, 60-965 Poznan, Poland
{adanilecki, akobusinska, mszychowiak}@cs.put.poznan.pl

Abstract. This paper describes a checkpointing mechanism destined for Distributed Shared Memory (DSM) systems with speculative prefetching. Speculation is a general technique involving prediction of the future of a computation, namely accesses to shared objects unavailable on the accessing node (read faults). Thanks to such predictions objects can be fetched before the actual access operation is performed, resulting, at least potentially, in considerable performance improvement. The proposed mechanism is based on independent checkpointing integrated with a coherence protocol for a given consistency model introducing little overhead. It ensures the consistency of checkpoints, allowing fast recovery from failures.

1 Introduction

Modern Distributed Shared Memory (DSM) systems reveal increasing demands of efficiency, reliability and robustness. System developers tend to deliver fast systems which would allow to efficiently parallelize distributed processes. Unfortunately, failures of some system nodes can cause process crashes resulting in a loss of results of the processing and requiring to restart the computation from the beginning. One of major techniques used to prevent such restarts is *checkpointing*. Checkpointing consists in periodically saving of the processing state (a *checkpoint*) in order to restore the saved state in case of a further failure. Then, the computation is restarted from the restored checkpoint. Only the checkpoints which represent a consistent global state of the system can be used (the state of a DSM system is usually identified with the content of the memory).

There are two major approaches to checkpointing: coordinated (synchronous) and independent (asynchronous). Coordinated checkpointing requires expensive synchronization between all (or a part) of the distributed processes in order to ensure the consistency of the saved states. The significant overhead of this approach makes it impractical unless the checkpoint synchronization is correlated with synchronization operations of a coherence protocol ([4]). On the other

* This work has been partially supported by the State Committee for Scientific Research grant no. 3T11C 073 28.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 9–16, 2006.

hand, the independent checkpointing does not involve interprocess synchronization but, in general, does not guarantee the consistency. After a failure occurs, a consistent checkpoint must be found among all saved checkpoints, therefore the recovery takes much more time and may require much more recomputation. A variant of the independent checkpoint – *communication induced checkpointing* (or *dependency induced checkpointing*), offers simple creation of consistent checkpoints, storing a new checkpoint each time a recovery dependency is created (e.g. interprocess communication), but its overhead is too prohibitive for general distributed applications. However, this approach has been successfully applied in DMS systems in strict correlation with memory coherence protocols. This correlation allows to reduce the number of actual dependencies and to significantly limit the checkpointing overhead ([2],[10]).

Speculation, on the other hand, is a technique which promises to increase the speed of DSM operations and reduce the gap between DSM systems and message-passing systems. The speculation may involve speculative pushes of shared objects to processing nodes before they would actually demand access [11], prefetching of the shared objects with anticipation that application process would need those objects ([1],[8],[12]) or self invalidation of shared objects to reduce the frequency of "3-hop-misses" ([6],[7]) among other techniques.

The speculation techniques may be argued to be special form of machine learning; it's however a restricted and limited form of learning. The speculation methods are required to be very fast, while they do not necessary have to make correct predictions, as the cost of mistakes is usually considered negligible. Therefore the existing well-known machine learning algorithms are usually not applicable in the DSM.

This paper is organized as follows. Section 2 presents a formal definition of the system model and speculation operations. In Section 3 we propose the conception of a checkpointing mechanism destined for DSM systems with speculation and discuss the proposition. Concluding remarks and future work are proposed in Section 4.

2 DSM System Model

A DSM system is an asynchronous distributed system composed of a finite set of sequential processes $P_1, P_2, ..., P_n$ that can access a finite set O of shared objects. Each P_i is executed on a DSM node n_i composed of a local processor and a volatile local memory used to store shared objects accessed by P_i. Each object consists of several values (*object members*) and *object methods* which read and modify object members (here we adopt the object-oriented approach; however, our work is also applicable to variable-based or page-based shared memory). The concatenation of the values of all members of object $x \in O$ is referred to as *object value* of x. We consider here read-write objects, i.e. each method of x has been classified either as read-only (if it does not change the value of x, and, in case of nested method invocation, all invoked methods are also read-only) or read-and-modify (otherwise). Read access $r_i(x)$ to object x is issued when process P_i

invokes a read-only method of object x. Write access $w_i(x)$ to object x is issued when process P_i invokes any other method of x. Each write access results in a new object value of x. By $r_i(x)v$ we denote that the read operation returns value v of x, and by $w_i(x)v$ that the write operation stores value v to x. For the sake of simplicity of the presentation we assume that each write access to an object writes a unique value.

To increase the efficiency of DSM, objects are replicated on distinct hosts, allowing concurrent access to the same data. A consistent state of DSM objects replicated on distinct nodes is maintained by a *coherence protocol* and depends on the assumed *consistency model*. Usually, one replica of every object is distinguished as the *master replica*. The process holding master replica of object x is called x's owner. A common approach is to enable the owner an exclusive write access to the object. However, when no write access to x is performed, the object can have several replicas simultaneously accessible only for reading (shared replicas). The speculation introduces special part of the system, called the predictor, which is responsible for predicting future actions of the processes (e.g. future read and write accesses) and according reactions.

As a result of a read access issued to an object unavailable locally, the object is fetched from its owner and brought to the requester. Using speculation, however, an object may be fetched from its owner also as a result of a prediction before the actual read access (i.e. *prefetched*). By $p_i(x)$ we will distinguish a prefetch operation of object x resulting from prediction made at process P_i.

Dependency of operations is a relation arising between $w_i(x)v$ and any subsequent $r_j(x)v$, i.e. when process P_j uses (reads) a value written previously by P_i. *Local dependency* reflects the order of operations performed by the same single process.

3 Speculation and Checkpointing

According to our knowledge, the impact of speculation on the checkpointing has not been investigated until now. While it seems impossible (or at least, improbable) that properly implemented speculation may danger the consistency of the system and correctness of the checkpointing algorithms, ignoring the existence of speculation in distributed shared memory system may severely damage speed of both making checkpoints and system recovery because it could create false, non-existing dependencies between nodes, as we will try to show.

We will focus on problems resulting from using prefetching techniques, but our approach should be easily adaptable to other speculation methods. In such techniques a special part of the system, called *predictor*, is responsible for anticipating the possible future read faults and preventing them by fetching respective objects in advance. The prediction may be incorrect in the sense that the process will never actually access the fetched object. Nevertheless, using speculation techniques such as the popular two level predictor MSP ([5]) turns out to increase the efficiency of most DSM applications. Moreover, since the predictor fetches objects using the underlying coherence protocol, it never violates the consistency of the memory.

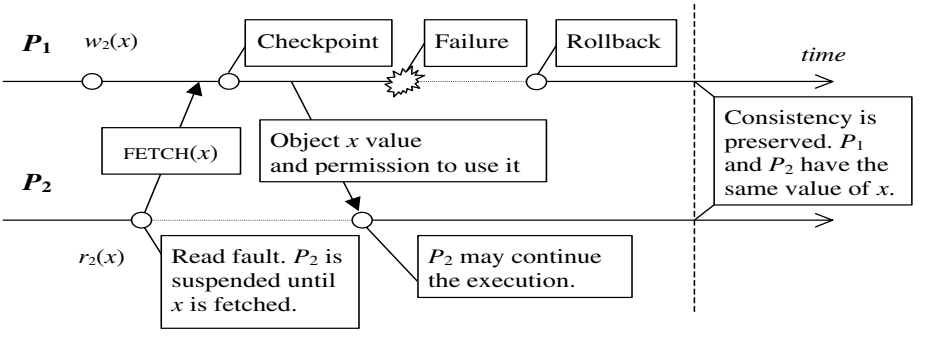


Fig. 1. Scenario without speculation. Real dependency between P_1 and P_2.

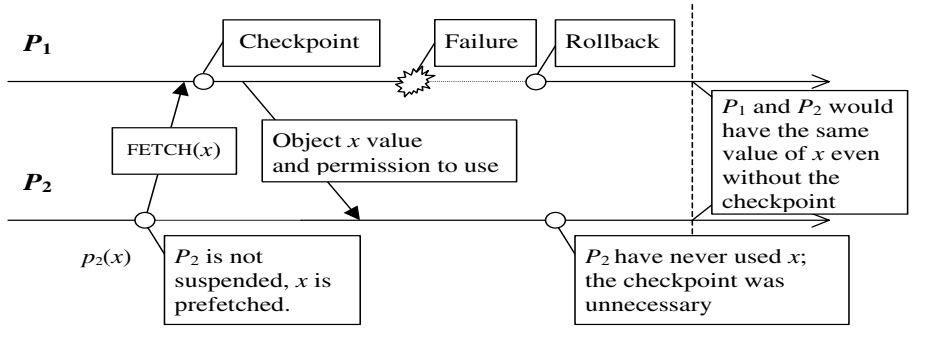


Fig. 2. Scenario with speculation. No dependency between P_1 and P_2.

Let us now consider the hypothetical execution shown in Fig. 1. There is a dependency between processes P_1 and P_2, since P_2 fetches the value modified by P_1. To ensure the consistency in case of a subsequent failure of process P_1, the system forces P_1 to take a checkpoint of the previously modified object x (it may be necessary to save also some other objects in the same checkpoint, in order to preserve local dependency of modifications performed by P_1; this is not shown in the figure).

However, the situation may significantly change with use of speculation. In the scenario presented in Fig. 2 the predictor assumes that the application process P_2 will read the value modified by P_1, so it fetches the object ahead into the local memory of P_2, to avoid a further read-fault. Performing that fetch, the system forces process P_1 to take a checkpoint, as in previous example. However, the prediction eventually turns out to be false and P_2 does not actually access x. Therefore, no real dependency was created and checkpoint was unnecessary. Unfortunately, P_1 was unable to determine that the fetch resulted from a false prediction, even if that fetch operation has been known to be speculative.

The problems presented above are summarized as follows:

Access to objects (fetches) may result from speculation made by predictor and therefore (in case of false prediction) may not result in real dependency;

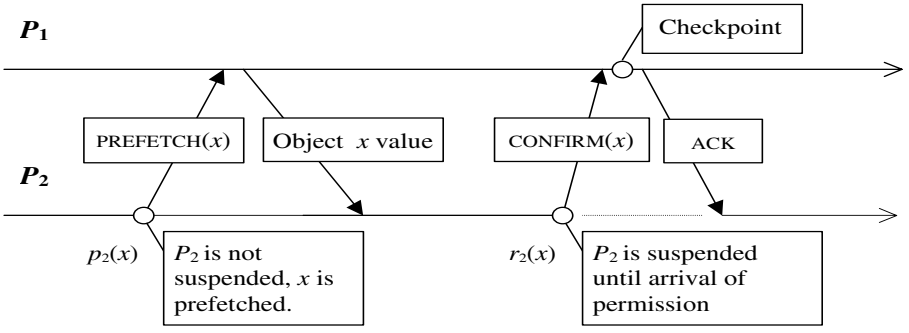


Fig. 3. The coherence decoupling

Even when an access is marked as speculative, process has no way of determining whether true dependency between processes will ever be created, since it cannot determine whether the prediction is correct (otherwise, it wouldn't be called speculation).

A possible solution is introduction of a new replica state and decoupling of access requests for objects into two phases: prefetch and confirmation (Fig. 3). A similar idea of coherence decoupling has been proposed in [3]. A speculative prefetch operation is explicitly distinguished from a coherence operation of a read access. The prefetched object replica is set into state PREFETCHED on the requesting node, and PRESEND on the owner. Further read access performed on the requesting node requires to merely ask for acknowledgement of accessing the object (message CONFIRM). On reception of this message the owner takes a checkpoint of the object, if necessary (e.g. the checkpoint could been taken already before reception of CONFIRM request as a result of some operations issued in the meantime by other processes), and answers with a permission message (ACK).

Please note that ACK message does not contain the value the requested object (since this value has been formerly prefetched and is available for the requesting node). Therefore the overhead of the confirmation operation is in general lower than a read-fault.

If the master replica of the considered object has been modified after a prefetch but before the corresponding confirmation it is up to the coherence protocol to decide about the acknowledgement (reading outdated values may be disallowed depending on the consistency model). Also the coherency protocol may involve invalidation of a prefetched object before the confirmation. This invalidation will be performed for prefetched objects exactly as for all object fetched by nonspeculative operations. Therefore, there is no difference between those two types of operations from the point of view of the coherence (thus, only minor modifications of coherence protocols will be necessary). The only significant difference concerns the checkpointing operations.

Our approach avoids unnecessary taking of checkpoints after a prefetch (when no real dependency is created). The checkpoint is postponed until an actual

dependency is revealed on the confirmation request). To reduce the checkpoint overhead many checkpointing protocols perform a consolidated checkpoint of an entire group of objects (*burst checkpoint* [2]). It is possible to include also the prefetched objects in such a burst checkpoint. This allows to further reduce the checkpointing overhead, since the prefetched object may already be checkpointed at the moment of confirmation and no additional checkpoint will be required. In such a situation, there will be no checkpoint overhead perceived by the application neither on prefetch, nor on actual read access to the prefetched object.

Finally, let us consider a recovery situation presented in Fig. 4. After the value of x has been checkpointed it is modified again, to 2. Process P_2 prefetches the modified value of x from P_1. Then, P_1 fails and recovers, restoring the checkpointed value $x = 1$. Please note that the following confirmation cannot be granted, as it concerns a value of x that became inconsistent after the recovery. The simplest solution could be to invalidate all replicas of x prefetched by other processes. This Invalidation can be performed on recovery of the owner.

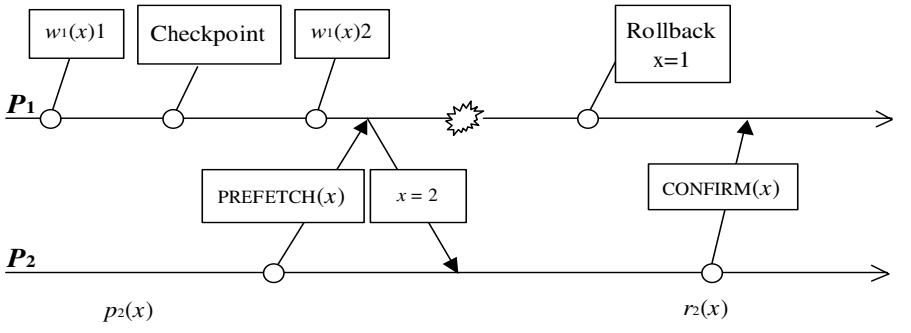


Fig. 4. Possible coherence problems with node failures

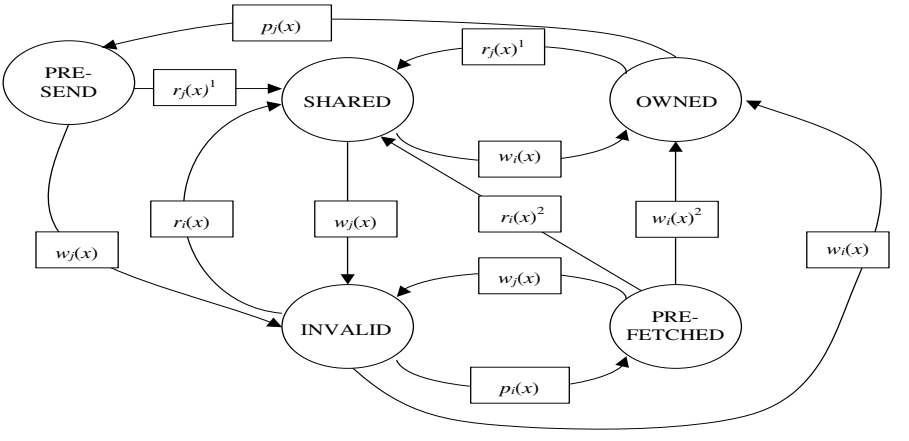


Fig. 5. The example state diagram of the protocol for sequential consistency model

The state diagram for replica of object x at process P_i is presented in Fig. 5. The assumed consistency model is sequential consistency [9]. The superscript indexes at the diagram denote that: [1]operation requires a checkpoint, [2] operation requires a confirmation.

4 Conclusions

This paper describes an approach to checkpointing shared objects with use of speculation. We recognize the false dependencies and unnecessary checkpoints related to speculation operations on the shared objects. We propose the operation decoupling which allows to decrease the frequency of checkpoints. Moreover, we describe additional mechanisms reducing the checkpointing overhead and indispensable modifications of the coherency operations after a failure and recovery.

There are at least three directions in which our approach could be studied and extended. First, to consider the implementation of proposed technique with using concrete coherence model and checkpointing algorithm. Second, to seek the optimizations for increasing positive effects of speculation. Third, to find a way to resolve issues with restarting processes.

Intuitively, there may be many possible optimizations which could be applied to the proposed checkpointing technique. Since our approach is very general, the implementation for a specific coherence model may exploit distinct features of underlying protocols. An obvious optimization might allow to use the prefetched object even before arrival of the permission.

In our approach, if object owner refuses to confirm the prefetch, the prefetched object is invalidated. Another optimization might fetch the current value of the object with ACK message sent back to the requester.

In many typical scientific applications there are program loops which produce strictly defined sequence of requests. Commonly employed in such cases is grouping the objects accessed in the loop into blocks, fetching (or prefetching) them together. Access to the first object from such group may signal that the program loop started again and other objects from this group will also be fetched subsequently. Therefore, it appears useful to confirm the whole group on access to the first object.

Requiring the object owner to deny all confirmation request after the failure may seem to be too harsh. A different solution would allow the object owner to refuse confirmation only for objects prefetched before crash, and acknowledge objects prefetched after the recovery.

References

1. Bianchini, R., Pinto, R., Amorim, C. L.: Data Prefetching for Software DSMs. Proc. International Conference on Supercomputing, Melbourne, Australia (1998)
2. Brzeziński, J., Szychowiak, M.: Replication of Checkpoints in Recoverable DSM Systems. Proc 21st Int'l Conference on Parallel and Distributed Computing and Networks PDCN'2003, Innsbruck (2003)

3. Chang, J., Huh, J., Desikan, R., Burger, D., Sohi, G.: Using Coherent Value Speculation to Improve Multiprocessor Performance. Proc 30^{th} Annual International Symposium on Computer Architecture, San Diego, California (2003)
4. Kongmunvattana, A., Tanchatchawal, S., Tzeng, N.-F.: Coherence-Based Coordinated Checkpointing for Software Distributed Shared Memory Systems. Proc. 20^{th} Conference on Distributed Computing Systems (2000) 556-563
5. Lai, A-C., Babak Falsafi, B.: Memory Sharing Predictor: The Key to a Speculative Coherent DSM. Proceedings of the 26th International Symposium on Computer Architecture (ISCA 26), Atlanta, Georgia (1999) 172-183
6. Lai, A-C., Babak Falsafi, B.: Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction. Proceedings of the 27th International Symposium on Computer Architecture (ISCA 27), Vancouver, BC, Canada (2000) 139-148
7. Lebeck, A., R., Wood, D.: Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors. Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita, Italy (1995) 48-59
8. Lee, S-K., Yun, H-C., Lee, J., Maeng, S.: Adaptive Prefetching Technique for Shared Virtual Memory. First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001). Brisbane, Australia (2001) 521-526
9. Raynal M., Schiper A.: A Suite of Formal Definitions for Consistency Criteria in Distributed Shared Memories, Technical Report PI-968, IRISA Rennes (1995).
10. Park, T., Yeom, H. Y.: A Low Overhead Logging Scheme for Fast Recovery in Distributed Shared Memory Systems. Journal of Supercomputing Vo.15. No.3. (2002) 295-320
11. Rajwar, R., Kagi, A., Goodman, J. R.: Inferential Queueing and Speculative Push. International Journal of Parallel Programming (IJPP) Vo. 32. No. 3 (2004) 273-284
12. Speight, E., Burtscher, M.: Delphi: Prediction-Based Page Prefetching to Improve the Performance of Shared Virtual Memory Systems. 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Monte Carlo Resort, Nevada (June 2002) 49-55

Evaluation of the Acknowledgment Reduction in a Software-DSM System

Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba

Graduate School of Information Systems, The University of Electro-Communications,
1-5-1 Chofugaoka Chofu-shi, Tokyo 182-8585, Japan
{kis, katagiri, honda, yuba}@is.uec.ac.jp

Abstract. We discuss the inter-process communication in software distributed shared memory (S-DSM) systems. Some S-DSM systems, such as TreadMarks and JIAJIA, adopt the user datagram protocol (UDP) which does not provide the reliable communication between the computation nodes. To detect a communication error and recover from it, therefore, an acknowledgment is used for every message transmission in the middleware layer. In this paper, first, we show that an acknowledgment is not necessarily required for each message transmission in the middleware layer. Second, a method to reduce the acknowledgment overhead for a page request is proposed. We implemented the proposed method in our S-DSM system Mocha. The performance of the method was measured with several benchmark programs on both a PC cluster and an SMP cluster.

1 Introduction

As an environment for parallel computation, cluster systems using general-purpose personal computers (PC clusters) are becoming popular. Because a PC cluster has no shared memory, the message passing model is used to develop applications in many cases. On the other hand, the shared memory is an attractive programming model for parallel computers. Software distributed shared memory (S-DSM) has been proposed to realize virtual shared memory on a PC cluster as the middleware layer software. Since the idea of S-DSM was proposed[1], several systems[2, 3, 4] have been implemented.

Some S-DSM systems, such as TreadMarks and JIAJIA, adopt the user datagram protocol (UDP) which does not provide reliable communication between the nodes. To detect a communication error and recover from it, therefore, an acknowledgment (ACK) is used for every message transmission in the middleware layer.

This paper discusses a technique to improve the S-DSM performance. Since the S-DSM system using UDP does not always need an acknowledgment for every message transmission, we propose a method of reducing the acknowledgment overhead. By implementing the proposed method on an S-DSM Mocha[5], we verify its effectiveness.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 17–25, 2006.

2 Omission of Acknowledgment Messages

Figure 1(a) shows how three messages, Msg-1, Msg-2, and Msg-3, are sent from node 1 to node 2. This is an example of conventional S-DSM communication. When the messages are received, node 2 on the receiving side immediately returns acknowledgment (ACK) messages ACK-1, ACK-2, and ACK-3 respectively.

Figure 1(b) shows a communication without acknowledgment. If all transmitted messages are received without error, the acknowledgment costs can be reduced as shown in (b). Omitting all acknowledgment messages means reducing the general send-receive message count in half. This is expected to enhance the communication performance, especially in applications with frequent messages. In the UDP communication, however, an error occurs by a certain frequency. In the example of communication errors shown in Figure 1 (c) and (d), Msg-2 does not reach the destination node in (c), and the arriving order of Msg-2 and Msg-3 cannot be guaranteed in (d). Despite of these communication errors, S-DSM systems should be constructed to operate correctly.

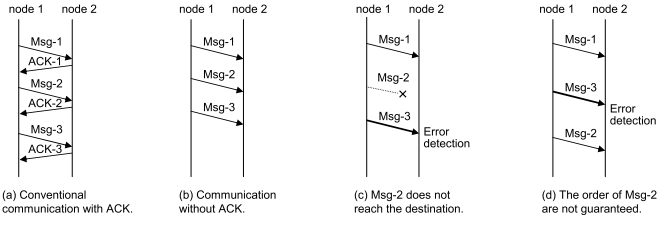


Fig. 1. S-DSM communication with and without acknowledgment

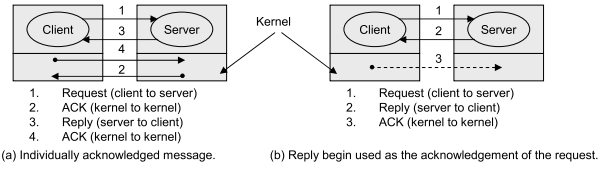


Fig. 2. Communication in a client-server model [6]

As shown in Figure 1(b), this paper is aimed at improving S-DSM performance by omitting acknowledgment messages. Note that the idea of the acknowledgment omission is not novel. For example, reference [6] discusses a technique of omitting acknowledgments in a client-server model. Figure 2 is the communication used in this discussion. An acknowledgment for the request can be omitted by using a reply message as its acknowledgment. Also an acknowledgment for the reply shown as the broken line in Figure 2(b) can be omitted depending on the properties of the reply. In this study, the concept of the acknowledgment omission in a client-server model is applied for the field of S-DSM.

3 Proposal of Acknowledgment Omission

3.1 Mocha: Yet Another S-DSM System

Mocha is an S-DSM system being constructed with two design philosophies: (1) It achieves good performance especially for a PC cluster of many computation nodes. (2) It offers an S-DSM system easy to use as a parallel processing environment.

Mocha Version 0.2 used for evaluation in this paper is a home based S-DSM, where each page is specified to a node by a user. Mocha is implemented to realize a simple and scalable S-DSM system by rewriting the JIAJIA with scope consistency[7]. The followings are points of the code simplification in order to increase readability: (1) JIAJIA's complicated functions for such as home migration and load balancing are removed. (2) Function interfaces are reorganized to make optimization of the source code. (3) The current version of Mocha supports Linux operating system only.

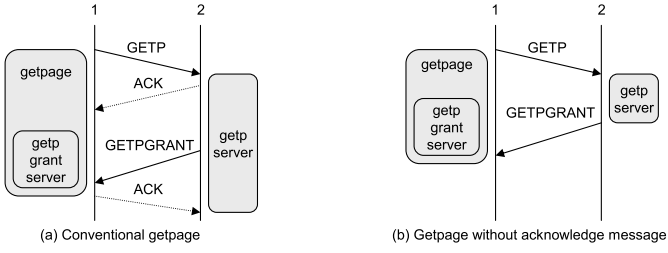


Fig. 3. The behavior of a page request with and without acknowledgment

3.2 Acknowledgment Omission for Page Request

Figure 3(a) shows the behavior of a page request. Suppose that node 1 requires a page and node 2 has the page. Node 1 calls the getpage function with the referenced memory address as a argument. The getpage function creates a GETP message corresponding to the page request and transmits it to node 2. On receiving this message, node 2 returns an acknowledgment (ACK) message as a reply. To meet the page request, node 2 then calls the getpserver function. This function packs the necessary page information in a GETPGRANT message and sends it to node 1. On receiving the GETPGRANT message, node 1 calls the corresponding getpgrantserver function. This function stores the received page information in an appropriate memory area and resets the global variable to terminate the getpage function. The getpage function waits in a busy wait state for the page to arrive. When GETPGRANT arrives, the getpage function exits from the busy wait state and continues the application processing.

The getpage function sends a GETP message and waits in a busy wait state. In this kind of page request processing flow, we propose the method of omitting acknowledgment.

```
1  void getpage(address_t addr){
2    getpwait=1;
3    generate_message(OP_GETP, addr);
4    send_message();
5    while(getpwait); /** busy wait **/
6  }
```

Fig. 4. The pseudo-code of the original getpage(the acknowledgment is not omitted)

```
1  void getpage(address_t addr){
2    getpwait=1;
3    for(i=0; i<GETPAGE_MAX_RETRY; i++){
4      generate_message(OP_GETP, addr);
5      send_message();
6      while(not_timeout() && getpwait); /** busy wait **/
7      if(getpwait==0) break;
8    }
9  }
```

Fig. 5. The pseudo-code of the getpage to be implemented in the proposed method

The exit of the getpage function from the busy wait state guarantees that no communication errors occurred in the two messages of GETP and GETPGRANT. If the function did not receive the GETPGRANT message within the timeout limit in the busy wait state, a communication error might have occurred. In this case, the GETP message is sent again and the system waits for the GETPGRANT message. The system should be designed to have no problems, even if the same message of GETP or GETPGRANT arrives several times. Using the GETPGRANT as an acknowledgment of GETP, acknowledgment of GETP or GETPGRANT can be omitted as shown in Figure 3(b).

Figure 4 shows the pseudo-code of the original getpage for which the acknowledgment is not omitted. Line 2 sets the global variable getpwait, Line 3 generates a page request message, and Line 4 transmits the generated message. When a message corresponding to the request arrives, the received page is stored appropriately and the global variable getpwait is reset. This finishes the while loop in Line 5 and terminates the page request function getpage.

Figure 5 is the pseudo-code of the function getpage to be implemented in the proposed method. The while-loop in Line 6 finishes when the global variable getpwait has been reset or a timeout event has occurred. If the variable getpwait equals to zero in Line 7, the getpage is terminated because the requested page is assumed to have been received. Otherwise, the for-loop from Line 3 transmits the page request message again.

4 Evaluation

For the evaluation, we use a 32-node PC cluster. Every node is connected with a 48-port gigabit Ethernet switch (HP ProCurve Switch 3400cl-48). Each node

is an SMP (symmetric multi-processor) type computer with two processors of Intel Pentium 4 Xeon (2.8 GHz) and 1 GB memory. The system software of the cluster is SCore 5.6.1 by PC Cluster Consortium[8] constructed on RedHat Linux 7.3. The execution time of each benchmark program is calculated by the arithmetic mean of five measurements.

As the benchmark programs, we use LU (parallel dense blocked LU factorization, no pivoting), N-queens[9], SOR (Red-Black Successive Over-Relaxation) and MM (Matrix Multiply). The object code is generated using GCC version 2.96 with O2 optimization option. As a parameter of LU, the matrix size of 1024×1024 (double precision), and the block size of 8 is used. The elapsed time of the sequential version is 267 second. As a parameter of N-queens, the problem size N=17 and the task allocation size of 8 is used. The elapsed time of the sequential version is 55.0 second. As a parameter of SOR, the matrix size of M=4096 and N=4096, iterations=400 is used. The elapsed time of the sequential version is 98.1 second. As a parameter of MM, the matrix size of 1280×1280 is used. The element of the matrix is double precision. The elapsed time of the sequential version is 9.14 second.

4.1 Performance Comparison on a PC Cluster

The data shown in this section was obtained on the configuration of a PC cluster. One process is assigned to one node. Figure 6 summarizes the results with the number of nodes on the x-axis and the speedup on the y-axis. The speedup is normalized by the elapsed time of JIAJIA single node. There are results for JIAJIA Version 2.2, TreadMarks Version 1.0.3.3 (in Figure 6(c) and (c) only),

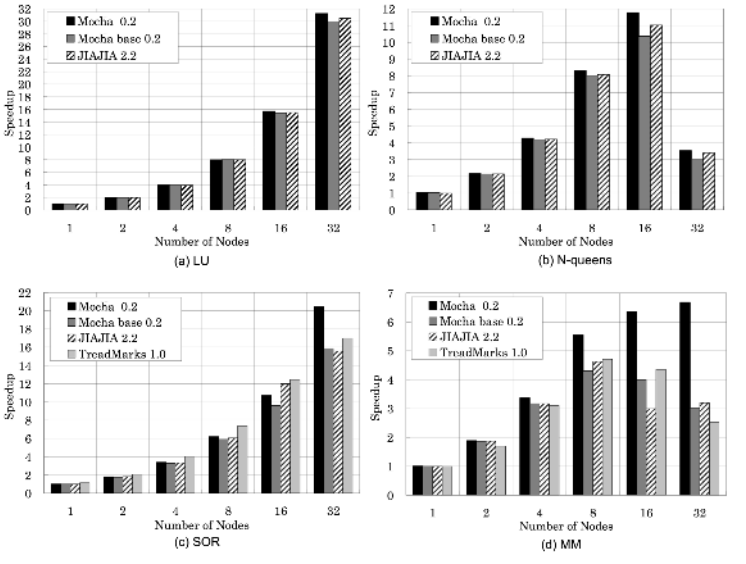


Fig. 6. The performance comparison of the S-DSM systems

and Mocha using the proposed method of omitting the acknowledgment (Mocha) and not omitting the acknowledgment (Mocha base).

The LU benchmark result is shown in Figure 6(a). The almost optimal speedup is achieved by increasing the number of nodes in every S-DSM system since the traffic in LU is small compared with the calculation.

The N-queens benchmark result is shown in Figure 6(b). The speedup on the 16-node configuration is 11.0 for JIAJIA and 11.7 for Mocha. The mocha is 6% faster than the JIAJIA on the 16-node configuration. In the case of 32-node configuration, the performance of every S-DSM system drops extremely because of the lock and unlock contention.

The SOR benchmark result is shown in Figure 6(c). The speedup on the 32-node configuration is 15.5 for JIAJIA and 20.4 for Mocha. The Mocha is 31% faster than the JIAJIA on the 32-node configuration. Some parameters of Mocha are set up so that its performance may become the optimal on the 32-node configuration. Therefore, TreadMarks shows the best speedup on the configurations of less than 32-node.

The MM benchmark result is shown in Figure 6(d). In MM execution, the message GETP and GETPGRANT account for most of the elapsed time of all communications. Therefore, the proposed method of reducing the overhead for the page request produces remarkable effects. The conventional systems, JIAJIA, TreadMarks and Mocha base, do not show performance improvement where the configuration of more than 8 nodes. Even on the 16-node and 32-node configurations, however, Mocha using the proposed method keeps performance improvement. Mocha achieves the speedup of 6.68 on the 32-node configuration. Compared with the 16-node Mocha base, the 16-node Mocha achieves a speedup as large as 58%.

Table 1 summarizes the number of errors (the sum of the communication errors at all nodes) for page request by the S-DSM system Mocha, which uses the proposed method. The data is calculated by the arithmetic mean of five measurements. From Table 1, we see that the frequency of errors is very low.

From the evaluation results in this section, the following conclusion can be obtained. Mocha using the proposed method achieves high performance in all benchmark programs except for SOR on small node configurations. Especially in a benchmark of high page transfer frequency, such as MM, Mocha achieves a drastic speedup as much as 58% on the 16-node configuration, compared with the conventional communication method of not omitting the acknowledgment.

Table 1. The average number of communication errors on Mocha for page request. Our SMP cluster is used as a PC cluster. One process is assigned to each node.

benchmark	2-node	4-node	8-node	16-node	32-node
N-queens	0.00	0.00	0.40	1.40	1.80
LU	0.00	0.00	0.40	1.80	4.00
SOR	0.00	0.00	0.20	0.80	1.20
MM	0.00	0.00	0.00	1.60	6.20

4.2 Performance Comparison of a PC Cluster and an SMP Cluster

This section discusses the performance comparison of a PC cluster (running one process per each node) and an SMP cluster (running two process per each node) assuming the same number of processors are used. Note that the processes running on the same node do not share the memory space.

Figure 7 summarizes the results with the number of processors on the x-axis and the speedup on the y-axis. The speedup is normalized by the elapsed time of JIAJIA single node. There are results for Mocha running one process on a node (Mocha-PC) and Mocha running two processes on a node (Mocha-SMP). The acknowledgments omission is used in both configurations.

The LU benchmark result is shown in Figure 7(a). The performance of Mocha-PC and Mocha-SMP is almost the same since the traffic in LU is small compared with the calculation. The N-queens benchmark result is shown in Figure 7(b). The performance of Mocha-SMP drops compared to Mocha-PC caused by the resource sharing (main memory and network interface) on Mocha-SMP. The speedup of Mocha-SMP is 88% of Mocha-PC on the 16-CPU configuration. The SOR and MM benchmark result is shown in Figure 7(c) and (d) respectively. Like the N-queens result, the speedup of Mocha-SMP is lower than Mocha-PC. In SOR, the speedup of Mocha-SMP is 66% of Mocha-PC on the 32-CPU configuration. In MM, the speedup of Mocha-SMP is 82% of Mocha-PC on the 32-CPU configuration.

The Figure 8 is drawn with the same data of Figure 7(c) but with a different x-axis, the number of nodes. Enabling the SMP improve the performance using the same number of node if a S-DSM system provides the scalable speedup. But the best performance is obtained by the Mocha-PC of 32-node configuration.

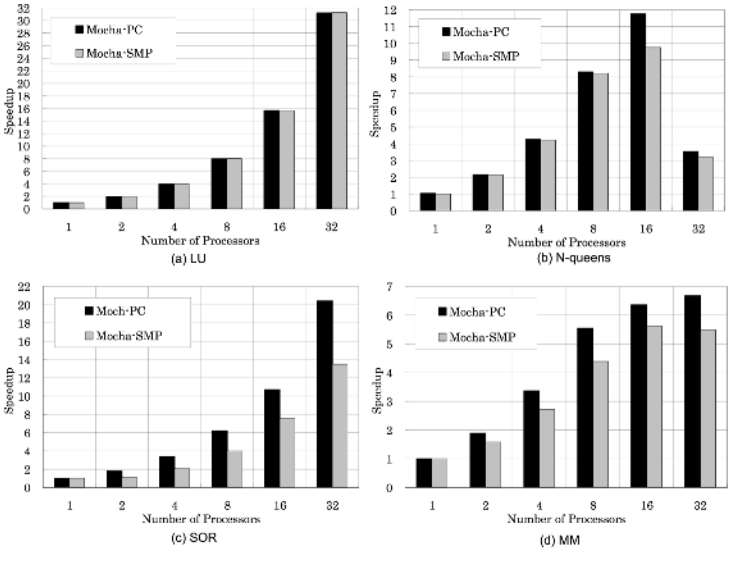


Fig. 7. The performance comparison of a PC cluster and an SMP cluster

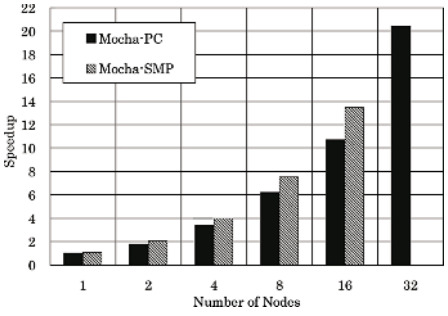


Fig. 8. The effect of SMP. Benchmark is SOR. Mocha-SMP has no data for 32-node configuration because Mocha supports 32 processes at large.

From the evaluation results in this section, the following conclusion can be obtained. Mocha-PC achieves higher performance in all benchmark programs than Mocha-SMP using the same number of processors. For example, Mocha-SMP running SOR could attain only 66% performance of Mocha-PC on the 32-CPU configuration.

4.3 Discussion

In general, the message receiving side checks a sequential number on a message. If the number is different from the expected value, a communication error is detected and error recovery is attempted. This simple error detection method, using a sequential number, may cause a great discrepancy between the error occurrence time and the error detection time and thus makes error recovery difficult. Another method is to transmit one acknowledgment message for n messages received. By increasing the value of n, most of the acknowledgment overhead can be eliminated. Even when this method is used, however, it is difficult to solve the problem of the great discrepancy between the error occurrence time and the error detection time. As a result of studying these candidates, we selected and proposed the method comparatively effective and easy to implement.

5 Conclusions

This paper discussed a technique to improve the S-DSM performance. Since the S-DSM system using the UDP does not always need an acknowledgment for every message transmission, we proposed a method of reducing the acknowledgment overhead for a page request and discussed its implementation.

We implemented the proposed method on our S-DSM system Mocha. The performance was measured with several benchmark programs. From the evaluation results as a PC cluster, the following conclusion can be obtained. Mocha using the proposed method achieves high performance in all benchmark programs except for SOR on small node configurations. Especially in a benchmark of high page transfer frequency, such as MM, Mocha achieves a drastic speedup

as much as 58% on the 16-node configuration, compared with the conventional communication method of not omitting the acknowledgment. From the comparison of a PC cluster and an SMP cluster, we showed that Mocha-PC achieves higher performance in all benchmark programs than Mocha-SMP using the same number of processors.

References

1. Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing. In: Proceedings of the International Conference on Parallel Processing (ICPP'88). Volume 2. (1988) 94–101
2. Keleher, P., Cox, A.L., Dwarkadas, S., Zwaenepoel, W.: TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In: Proceedings of the Winter 94 Usenix Conference. (1994) 115–131
3. Eskicioglu, M.R., Marsland, T.A., Hu, W., Shi, W.: Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures. In: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8, IEEE Computer Society (1999)
4. Cheung, B.W.L., Wang, C.L., Hwang, K.: Migrating-Home Protocol for Implementing Scope Consistency Model on a Cluster of Workstations. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99). (1999)
5. Kise, K., Katagiri, T., Honda, H., Yuba, T.: Mocha Version 0.2: Yet Another Software-DSM System. Technical Report UEC-IS-2005-3, Graduate School of Information Systems, The University of Electro-Communications (2005)
6. Tanenbaum, A.S.: Modern Operating Systems. Prentice-Hall International Editions (1992)
7. Iftode, L., Singh, J.P., Li, K.: Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures. (1996) 277–287
8. PC Cluster Consortium, http://www.pccluster.org/index.html.en/.
9. Kise, K., Katagiri, T., Honda, H., Yuba, T.: Solving the 24-queens Problem using MPI on a PC Cluster. Technical Report UEC-IS-2004-6, Graduate School of Information Systems, The University of Electro-Communications (2004)

Taking Advantage of the SHECS-Based Critical Sections in the Shared Memory Parallel Architectures

Tomasz Madajczak[1,2]

[1] Gdansk University of Technology, Faculty of Electronic, Telecommunication and Informatics, Dep. of Computer Systems Architectures, ul. Narutowicza 11/12, 80-233 Gdansk, Poland
[2] Intel Technology Poland Sp. z o.o. ul. Slowackiego 173, 80-298 Gdansk, Poland
Tomasz.Madajczak@intel.com

Abstract. This document presents a new method for implementing critical sections in the shared memory parallel architectures such as multithreaded multiprocessors integrated on a die. The method bases on Shared Explicit Cache System (SHECS) implemented in the multiprocessor. The document presents the concept of system architecture equipped with SHECS, the algorithm to implement operating system or application level locking service, and the results obtained with the method simulation on the network processor Intel[1] IXP2800.

1 Introduction

The Multicore Shared Memory Parallel Architectures are becoming very popular thanks to mass availability of the multicore SMP (symmetric multi-processors) systems such as multicore IA (Intel Architecture) processors and multicore specialized RISC systems such as the IXA (Intel Exchange Architecture) network processors. The level of parallelism in such systems is enhanced by the hardware threading technologies such as simultaneous multithreading (SMT) and switch-on-event multithreading (SoEMT), respectively. Efficient use of shared resources is always connected with the need of a critical section implementation. Multicore and multiprocessors systems have the ability to rely on specific hardware support and capabilities to synchronize the particular processor cores within the die or the integrated platform. Hardware techniques for critical sections are available in the network processors and they are very efficient, but not always universal [1]. Software techniques are still available for the SMP systems and for cases when the hardware support isn't flexible.

Thus, there is a need for a flexible hardware method that would address the perspectives of multithreaded multiprocessors systems. This document presents such a new method based on the use of SHECS being an additional, manageable, explicit cache system. It is derived from the Folding method [2].

[1] Intel is a registered trademark of Intel Corporation in the United States and other countries.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 26–33, 2006.

1.1 Folding Method in Network Processors

The Folding method was introduced in the network processors Intel IXP2000 [3] as a universal method for programming software critical section for the threads of a single microengine (that is a RISC processor). The method uses the microengine's internal local memory and CAM (content addressable memory) lookup engine that comprises an internal explicit cache system managed with software.

Folding caches the read data to be modified. It manages with the following critical section scenario that firstly reads a resource, then modifies it, and finally writes back the modified data. The read resource is stored within this system and considered locked if its use counter is greater than 0. Folding may occasionally lost the order of threads entering the critical section, because in case of finding locked entry the algorithm repeats the entering. This order lost means that the critical section may be starving for some threads, as they have no luck and they always repeat entering. Extending the Folding method onto a number of processors is not an easy task and also starving critical sections aren't useful in the general purpose parallel systems.

All these issues are solved in the SHECS-based method. It bases on the new explicit cache memory system architecture that eliminates the Folding's limitations and disadvantages. Thus, the SHECS-based method is more universal, flexible, and may have more applications.

1.2 Cache Coherency

The Parallel Shared Memory Architectures built with using general purpose processors with internal caches may have implemented a mechanism for enforcing the coherence of internal caches. Hardware solutions for this problem are presented in [4][5], while software algorithms are discussed in [6][7]. Generally there are two approaches: implicit methods that hide the problem for the system user or software, and explicit methods assuming that the system user or software is aware of the problem, treats cache as a normal shared resource and solves it with cache-locking [8][9] or other locking method. The introduced method addresses the problem in the similar way as explicit methods. It assumes explicit locking with integrated data transfers of the most recent cached data value and additionally it copes with hardware threading.

2 The Concept of SHECS-Based Critical Sections

2.1 SHECS Architecture

SHECS consists of CAM banks controller, a number of CAM banks and some shared fast SRAM memory for caching[2]. Fig. 1 shows that SHECS should be connected in a similar way as shared memory to the parallel processors P1-PN. The key assumption is providing a number of CAM banks that can

[2] The concept of Shared Explicit Cache System is patent pending in the U.S. patent office.

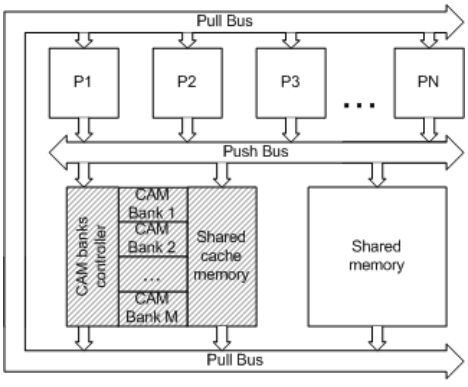


Fig. 1. SHECS in a shared memory parallel architecture

work together or separately. Every CAM bank is a functionally complete unit comprising entries tag and lock latches, lookup-locks FIFO queue and configuration for associated region of the cache memory. Every operation on the explicit cache system should have associated the mask that determines which banks are associated to the operation (single bit in the mask controls (enables/disables) one CAM bank). From the high level point of view the masking capability allows coupling a number of independent CAM banks into a single CAM. Such a single CAM should also have a consistent old tag removing policy - least recently used tag should be removed upon adding a new tag within all the banks enabled with a particular mask. This requirement is realized with the CAM banks controller logic. Because the SHECS method is hardware-based the way of enforcing coherency of critical sections is hardware signaling. The signal arrival wakes up a virtual or hardware thread that normally waits on that signal for the completion of CAM-lookup-lock operation.

2.2 SHECS Functionality

A functionally complete SHECS provides the following operations within the specified banks:

- CAM lookup with data locking and integrated reading
- CAM entry unlocking
- CAM entry's cache reading and writing with entry unlocking
- CAM banks managing (clearing the bank, adding, deleting tag, etc.)

Every CAM bank may have associated a line of cache memory to store data to be locked and modified. Because, the memory is a part of the explicit cache memory system, the reading of such a memory line may be integrated with successful CAM lookup operation. If processors P1-PN have a support for reading data bursts directly into the registers (such as network processors) this data may be used at once. Another way to use CAM lookup with integrated data transportation is to store data bursts within the processors' local caches.

SHECS may handle some number of pending simultaneous CAM-lookup-locks. This limitation is connected with the depth L of the FIFO queue storing pending lookup-locks. Such a FIFO is implemented in the each CAM bank. This depth may be exceeded if a parallel application performs more than L simultaneous CAM-lookup-locks for the same tag value. In that case the CAM-lookup-lock requests are rejected with the result indicating locking fullness. In the other words the locking is not starving as long as the number of requesting task is lower and equal to the depth L. It should be calculated with the following formula:

$$L = N * K * C \tag{1}$$

where: N - the number of processors; K - the number of hardware threads in each processor; C - depth redundancy constant $>= 1$.

The value of depth L implemented according to formula (1) manages with all synchronization problems that use maximally all available parallel resources. However if a program tries to be executed with higher level of parallelism than available parallel resources (in the other words the program have more virtual threads or processes than there is hardware threads in the parallel systemt multiplied with constant C), then the locking implemented with using SHECS may be starving for some virtual threads.

2.3 Implementing SHECS-Based Full-Locking Critical Sections

Full-locking technique (described well in [10]) assumes that every element that must be used or modified coherently may be locked independently for a certain time. Because the explicit cache system has hardware limitations such as limited number of CAM entries and limited number of pending locks for a particular CAM bank (due to the limited depth L), the implementation of full-locking critical sections should combine the explicit cache system functionality and some Operating System mechanismism to queue rejected SHECS requests. Otherwise, if only using the hardware technique, the parallel program decomposition should allow starving, that may happen if the program has more threads or processes than the depth L that want to coherently use the same resource. A non-starving implementation is depicted in Fig. 2. It assumes some Operating System support for starving avoidance and bases on the following facts:

- The critical data, that is stored in SHECS, is considered locked if the lock bit is set for the relevant CAM entry tag
- Otherwise if the lock bit is clear, it is considered unlocked and may be reused (it stores the most recently critical data)
- If the critical data is not stored in SHECS it is unlocked

From the funtional point of view the results of section entering are:

- bypass - critical data has been stored unlocked in SHECS and after locking it is transfered to a new critical section owner
- locked-bypass - critical data has been stored locked in SHECS and after lock release it is transfered to a new critical section owner

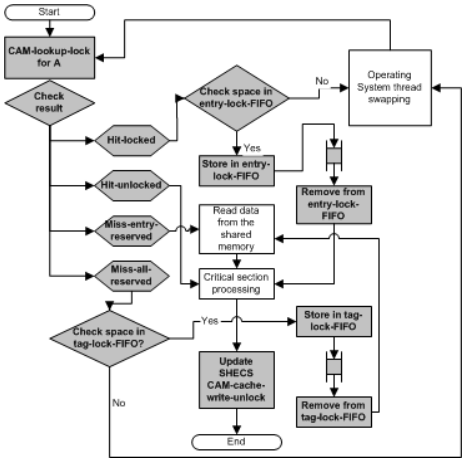


Fig. 2. The SHECS's algorithm for full-locking critical sections (gray elements mark hardware processing)

- reload - critical data has not been stored in SHECS and the entry's ownership is granterd to a new critical section owner
- locked-reload - all SHECS's entries has been used and after freeing one of them its ownership is granted to a new critical section owner

3 Simulation on Network Processor

3.1 Simulation Method

The full-locking approach was implemented and simulated. The simulation environment was the Developer Workbench 4.2 for Intel's IXA Network Processors. Two microengines were serving two SHECS banks with capacity of 32 cacheable entries in total. This was possible thanks to the fact that the microengines are equipped with local CAM and local memory. The parallel application, that was searching and modifying a binary tree, was executed on four microengines.

Every thread in the application performs in parallel a random search for a key in a binary tree and then modifies the node data. The binary tree had 129 nodes with keys from 0 to 128. To compare the results, two methods of coherent modification was implemented:

- Program 1 SHECS-based critical sections being between `shecs_lookup_lock()` and `shecs_write_unlock()` primitives.
- Program 2 Spin-locks-based critical sections (starving) being between `lock_node()` and `unlock_node()` primitives.

Program 1 Modifying nodes of binary tree with using SHECS-based critical sections

```
while (no_loops--) {/* do in parallel*/
  key = PSEUDO_RANDOM_NUM mod 128;
  result = shecs_lookup_lock(/*in*/key,/*out*/index,/*out*/node);
  if (result == LOOKUP_BYPASS)
    get node_addr from node;
  else //result == ENTRY_RESERVED
    search_tag_in_bintree(/*in*/key,/*out*/node,/*out*/node_addr);
  modify node;
  shecs_write_unlock(/*in*/key,/*in*/node_addr,/*in*/index,/*in*/node)
}
```

Program 2 Modifying nodes of binary tree with using spin-lock-based critical sections

```
while (no_loops--) {/* do in parallel */
  key = PSEUDO_RANDOM_NUM mod 128;
  search_tag_in_bintree(/*in*/key,/*out*/node,/*out*/node_addr);
  lock_node(node_addr); // spin lock test_and_incr
  modify node; write node to memory;
  unlock_node(/*in*/node_addr);//spin lock decr
}
```

3.2 Simulation Results

The results prove the SHECS's ability to maximize parallel processing performance thanks to the explicit searching for keys combined with transportation of the most recent value of critical data. Particularly, Fig. 3 shows that with SHECS-based critical section parallel program still remains scalable with the number of parallel resources while the scalability for spin-lock based critical sections has finished.

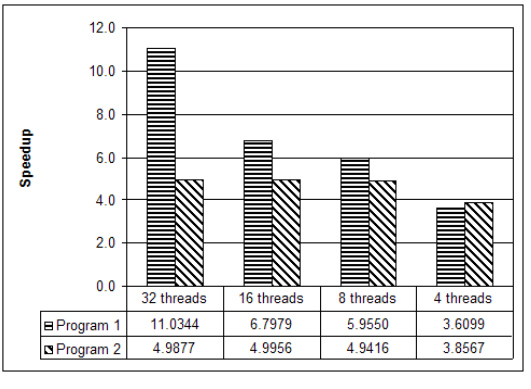


Fig. 3. Speedups vs. the number of parallel threads for Program 1 and Program 2

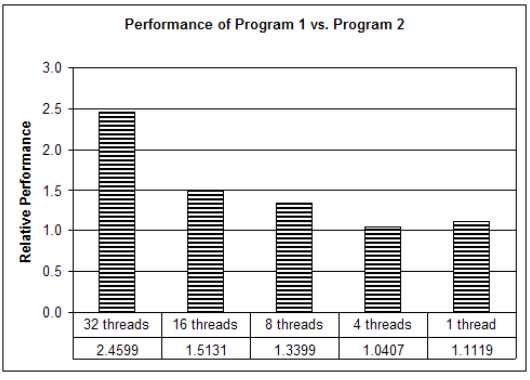


Fig. 4. Performance comparison between Program 1 and Program 2

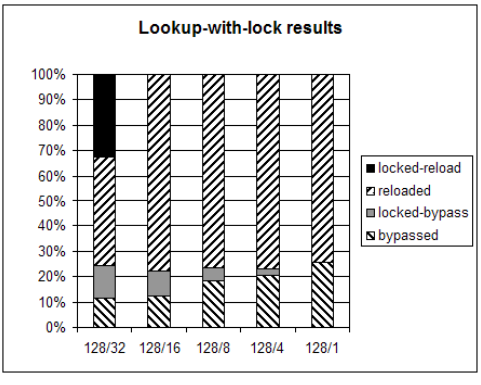


Fig. 5. The distribution of the SHECS lookup-with-lock results for Program 1

4 Final Remarks

SHECS enables achieving the best possible performance gain for data-driven parallelism in the Shared Memory Parallel Architectures thanks to managing with critical sections at hardware speed. SHECS may be also used to speed-up data searching algorithms, thanks to providing explicit associative searching for keys in the designated CAM banks. Thus, the parallel algorithms, that search and modify shared dynamic data structures, can benefit from both SHECS features critical section synchronization with caching data and associative searching. Such a feature combination is a very powerful proposition for the shared memory architectures. SHECS increases the real parallelism in such systems and also proposes critical sections support for managing the cache coherence problems.

The only disadvantage of SHECS is the cost - it is an additional, manageable cache system. The content addressable memories (CAMs) (the key ingredient of SHECS) are still pricy and they aren't used in the general purpose systems.

However, the Moore's Law constantly decreases the cost of silicon circuits and it may cause that in the feasible future SHECS will be implemented and will offer performance gain in the parallel multicore systems integrated on a die.

References

1. H. Krawczyk, T. Madajczak: Optimal Programming of Critical Sections in Modern Network Processors under Performance Requirements. In the Proc. of IEEE Parelec Conf. Dresden 2004. 2004
2. M. Adiletta, D. Hooper, M. Wilde, Packet over SONET: Achieving 10 Gbps Packet Processing with an IXP2800. Intel Technology Journal Vol. 6 issue 3. 2002
3. M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich: The Next Generation of Intel IXP Network Processors. Intel Technology Journal Vol. 6 issue 3. 2002
4. A. Nanda, A. Nguyen, M. Michael, and D. Joseph: High-Throughput Coherence Controllers. In the Proc. of the 6th Int'l HPC Architecture. 2000
5. M. Azimi, F. Briggs, M. Cekleov, M. Khare, A. Kumar, and L. P. Looi: Scalability Port: A Coherent Interface for Shared Memory Multiprocessors. In the Proc. of the 10th Hot Interconnects Symposium. 2002
6. A. Grbic: Assessment of Cache Coherence Protocols in Shared-memory Multiprocessors. A PhD thesis from Grad. Dep. of Electrical and Computer Engineering University of Toronto. 2003
7. G. Byrd: Communication mechanisms in shared memory multiprocessors. A PhD thesis from Dep. Of Electrical Engineering of Stanford University. 1998
8. N. Aboulenein, J. Goodman, S. Gjessing, P. Woest: Hardware support for synchronization in the Scalable Coherent Interface (SCI). In Eighth International Parallel Processing Symposium. 1994
9. U. Ramachandran, J. Lee: Cache-based synchronization in shared memory multiprocessors. In Supercomputing '95. 1995
10. R. Jin, G. Yang, G. Agrawal: Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming, Interface, and Performance. IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No.10. 2004

Dynamic SMP Clusters in SoC Technology – Towards Massively Parallel Fine Grain Numerics

Marek Tudruj[1,2] and Lukasz Masko[1]

[1] Institute of Computer Science of the Polish Academy of Sciences,
01–237 Warsaw, ul. Ordona 21, Poland
[2] Polish–Japanese Institute of Information Technology,
02–008 Warsaw, ul. Koszykowa 86, Poland
{tudruj, masko}@ipipan.waw.pl

Abstract. This paper evaluates new architectural solutions for data communication in shared memory parallel systems. These solutions enable creation of run-time reconfigurable processor clusters with very efficient inter-processor data exchange. It makes that data brought in the data cache of a processor, which enters a cluster, can be transparently intercepted by many processors in the cluster. Direct communication between processor caches is possible, which eliminates standard data transactions. The system provides simultaneous connections of processors with many memory modules that further increases the potential for parallel inter-cluster data exchange. System on chip technology is applied. Special program macro-data flow graphs enable proper structuring of program execution control, including specification of parallel execution, data cache operations, switching processors between clusters and multiple parallel reads of data on the fly. Simulation results from symbolic execution of graphs of fine grain numerical algorithms illustrate high efficiency and suitability of the proposed architecture for massively parallel fine-grain numerical computations.

1 Introduction

This paper discusses fine-grain parallel numerical computations in a new cluster-based shared memory system architecture. The basic feature of the proposed architecture consists in dynamically reconfigurable shared memory processor (SMP) clusters which can adjust to computational and communication requirements of application programs. It provides features of dynamically reconfigurable embedded systems whose structure is adjusted to program needs accordingly to a pre-compiled strategy based on program analysis.

A new method for data exchange has been proposed, which consists in transferring data in data caches of processors that are dynamically switched between SMP clusters [6, 7, 8]. This method converts data transmissions through memory and some global network, by dynamic cluster reconfiguration with data transfers performed directly between data caches. With this method, multiple parallel

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 34–42, 2006.

reads of data by many processors to their data caches take place (reads on the fly, similar to cache injection [5]) while a processor writes data from its cache to the cluster memory. Reads on the fly combined with processor switching, called "communication on the fly", provide a very fast way for data exchange between processor clusters.

Processor data cache functionality is another specific feature of the proposed architecture. A task can be executed in a processor only if all data required for its execution are loaded to the processor's data cache. It defines a data cache-controlled macro-data flow program execution paradigm. Due to such data caching strategy, data reloading and thrashing in processor caches are eliminated during computations. In this respect, the proposed strategy provides incomparably better behavior than other caching solutions like cache pre-fetching, data-forwarding and cache injection [3, 4, 5].

Comparing earlier versions of this architecture [7], the assumption of Systems on Chip (SoC) technology [1, 2] brings modularity at the program structure and system levels. The dynamic clusters are now implemented inside SoC modules connected by a global network. It puts limits on cluster size but at the same time eliminates excessive signal propagation time. Systems on chip technology will soon provide parallel systems including thousands of processors [2]. Such systems, supported by adequate communication solutions, can increase efficiency for fine-grain parallel programs. Another novelty of the new version is multiple access to the data caches from the memory side. It increases efficiency, especially for fine grain computations, since simultaneous data transactions "cache-memory" are possible in one processor, including parallel operand pre-fetching/result storing and communication on the fly.

The proposed architectural solutions have been verified by simulation experiments performed using a special graph representation. Execution of very fine grain matrix multiplication with recursive data decomposition was simulated using a graph simulator written in C. It enabled simple modification of computation and communication grain down to a very fine level. Unpublished simulation results from execution of program graphs in the proposed architecture and NUMA systems are presented. They show efficiency of the proposed architecture for massive fine grain parallel computations.

The paper is composed of 3 parts. In the first part, the features of the assumed executive system architecture are presented. In the second part, the program graph representation used for parallel program design and estimations are outlined. In the third part, results of simulation experiments are presented.

2 Dynamic SMP Clusters Based on SoC Technology

Fig. 1a presents the general structure of the proposed system. Based on processors P and shared memory modules M, dynamic clusters can be created at program run-time using local data exchange networks. The processors, memory modules and networks are embedded in system on chip modules interconnected by a global data network. Fig. 1b presents architectural structure of a SoC

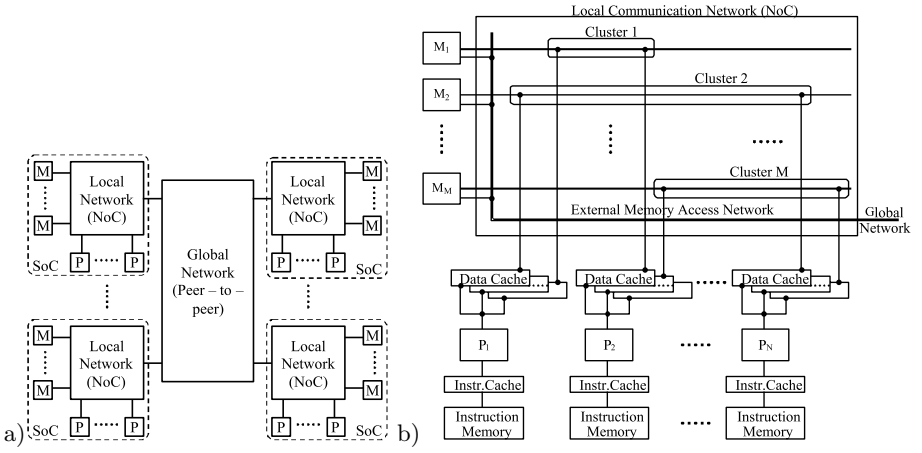


Fig. 1. General system structure a) and internal structure of the SoC module b)

module. It includes a number of processors (P_i), a set of instruction memory modules, a set of data memory modules (M_j), a set of separate data and instruction caches and a set of local cluster networks to which processors (i.e. their data caches) can be connected. Each processor is equipped with many data cache modules, which provide multi-ported access to/from memory modules. Therefore, a processor can belong to many clusters at a time. This feature strongly improves data communication efficiency. All memory modules are also connected to the External Memory Access Network, which can be a crossbar switch.

To control communication in clusters: data pre-fetch, write, reads on the fly (similar to cache injection), processor switching between clusters, communication on the fly operations can be used. A read on the fly consists in parallel capturing of data, which are being written on a local cluster network by a processor. Synchronization of reading processors with the writing one is provided. Processor switching between clusters consists in connecting a processor to a new cluster (i.e. its local network). A processor switched to a cluster can bring in its cache data, which are useful for the cluster. When the processor writes data to cluster memory, processors in the target cluster can read data on the fly. The synergy of processor switching with reads on the fly is called communication on the fly. Tasks in programs are so built that they do not require data cache reloading during their execution. All data have to be pre-fetched to processor's data cache before a task begins. Current task results are sent to the cluster memory module only after task completes. This program execution paradigm is called cache-controlled macro data-flow principle. It completely prevents data cache thrashing. When data produced by a task are to be modified by other parallel tasks, new target addresses are used. Such single assignment rule avoids cache consistency problem.

A program in a processor can contain memory write or read requests and synchronized read on the fly requests. The requests are serviced in processors by Communication Request Controllers (CRCs) using separate request queues. Each memory module has an arbiter, which co-ordinates memory requests issued by CRCs. Writes have higher priority than reads and short transactions have higher priorities than longer ones. Reads on the fly are similar to cache injection. They consist in reading data on the fly from a memory module bus whose address lines are snooped by a special address snooping unit. Read on the fly requests are stored in the address snooping tables. Exact synchronization of states of the process that writes with reading processes is necessary in the proposed architecture. Special inter-processor synchronization hardware has to be included in the system to enable parallel execution of many synchronization operations. More details on the architecture of the system can be found in [6, 7, 8].

3 Extended Macro-data-flow Graphs of Programs

An extended macro data flow graph representation (EMDFG) is used to specify programs for this architecture. New kinds of nodes are: memory module bus arbiter nodes (CA), the global data network arbiter node (GA), read nodes (R) from memory modules to processor's data caches, write nodes (W) from data caches to memory modules, processor switch nodes (Mi) and barriers (Bi). A program has to be structured into sections, which are executed by dynamic SMP clusters i.e. fixed subsets of processors connected to the some shared memory modules. It can be done automatically at compile time by program graph analysis or manually by directives of a programmer.

Fig. 2a shows an exemplary graph representation of a read on the fly. A barrier synchronizes all the involved processors. It conditions the write from the data cache to the cluster memory module. Reads on the fly can be done if all involved

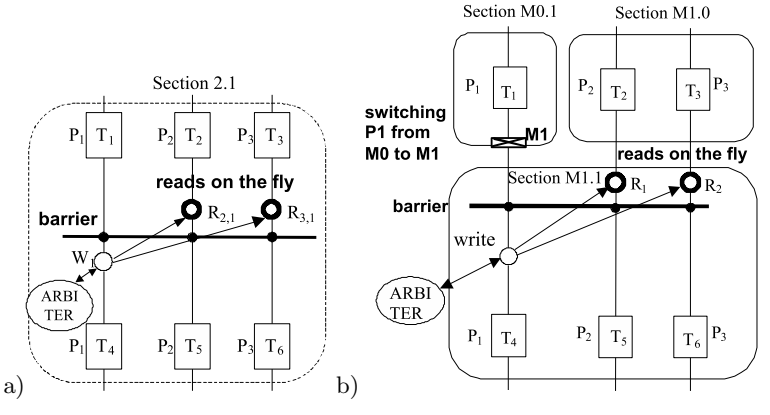


Fig. 2. Reads on the fly a) and communication on the fly b)

processors have introduced the memory read requests to the address snooping table. Then, the processors can snoop the data on the memory bus and read (all or their portions) to their data caches. In tasks assigned to the same processor data can be transferred through data caches.

Fig. 2b represents communication on the fly. A crossed rectangle represents switching of processor P1 from the cluster of the memory module M0 to the new processor cluster. It is labeled with the target memory module identifier M1. Communication on the fly between clusters is introduced into program graph sub-graphs, which show features of data communication "one-to-many". Unification of sections can be introduced in the program sub-graphs, which show "many-to-one" data communication features. An automatic program graph scheduler for the proposed architectural features is under design [9].

4 Experimental Results

We will examine speedup and program execution efficiency in the proposed architecture, especially for fine-grained programs. Square matrix multiplication $A \times B = C$, based on parallel recursive decomposition of matrices into quarters is an example where we can easily modify the parallel computation grain. At the 1^{st} recursion level we have 8 multiplications (M_i) of quarters of A, B and 4 additions (Ad_j) of resulting quarters to produce quarters of C. At the 2^{nd} recursion level, each input matrix quarter is further divided into 4 quarters and multiply nodes are replaced by entire graphs from the 1^{st} recursion level. At each recursion level, sub-matrices resulting from multiplication are divided into half-matrices — left and right, which are then added in parallel. Due to these assumptions, addition and communication times are reduced by a factor of two.

The complete algorithm graph of matrix multiplication at the 1st recursion level contains 4 isolated elementary subgraphs as a subgraph of a graph shown in Fig. 3a denoted by an oval rectangle. At the 2^{nd} recursion level the graph contains 16 isolated elementary subgraphs, that are shown in Fig. 3a. To these subgraphs we apply the technique of processor switching between clusters and reads on the fly. The transformed elementary sub-graph at the 2^{nd} recursion level from Fig. 3a, for the system where one processor can be connected to two memory modules at a time (dual-ported data cache is used), is shown in Fig. 3b. The sub-graph has been structured to use of 4 processors and 8 memory modules that are a basis for dynamically reconfigurable processor clusters (see program sections in ovals).

Each elementary sub-graph has been structured to the use of 4 processors (P_1, P_2, P_3, P_4) and 8 memory modules (M_1, M_{1x}, M_2, M_{2x}, M_3, M_{3x}, M_4, M_{4x}) that are a basis for dynamically reconfigurable processor clusters. In the beginning, each processor is connected to two memory modules (clusters) to read data for multiplication. After multiplication, processors P_1, P_2 and P_3, P_4 are switched to opposite clusters (organized around M_2, M_1, and M_4, M_3) to bring in their caches results for further computations of other processors in these clusters. Processor switching takes place also after first additions. The dynamic SMP

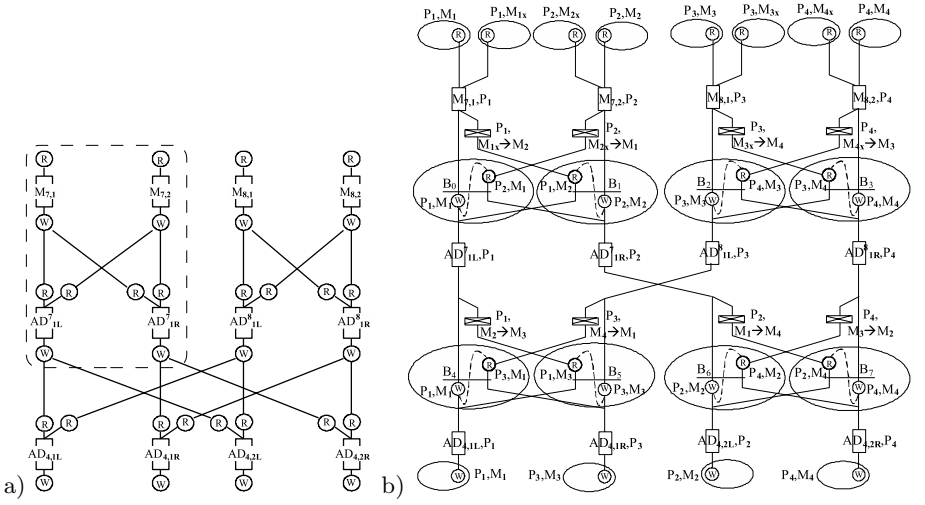


Fig. 3. Initial a) and structured b) elementary EMDFG at 2^{nd} recursion level

clusters organized at the 2^{nd} recursion level during execution of the middle part of elementary sub-graphs are composed of 2 processors. Each processor belongs to two clusters at a time. There are no standard data reads in the graph, except for initial data reads and final writes. All data exchange is done by reads on the fly, communication on the fly or data transfers through caches. It eliminates global exchange network transactions — only writes remain.

We present below comparative study of results of simulated symbolic execution of the discussed matrix multiplication algorithm graphs in the proposed architecture with different included features. The basic parameters were the matrix size (1024 - 64) and the recursion level (1 - 4). SoC modules are built for 16 processors which co-operate with 32 memory modules. We have further compared execution in our architecture with execution in the equivalent idealistic NUMA architecture, where programs were built according to the cache controlled macro-data flow paradigm with a single assignment strategy applied for writing shared data (no cache coherence protocols).

Two values of relations between computation speed of processors / communication speed i.e. the memory bus were examined: 6:1, 3:1 (see Table 1). The relation 6:1 is equivalent to 2.4 GFLOPS processors connected with data memory modules via 400 MHz busses. For this relation, the assumed operation execution times were: 1 unit for floating point 8-byte scalar addition and multiplication, 6 units for transmission of a 8-byte scalar between data cache and memory. The costs of barrier execution, processor switching and memory bus arbitration were assumed 1 unit. For the relation 3:1, the equivalent transmission cost was assumed to be 3 units. For matrix size 1024 at the 4th recursion level processors were executing multiplication of matrices 64x64. For speed relation 6:1 the speedup was about 3770 from 4096 processors with the efficiency of 0.92. Our architecture was 1.08 times faster than idealistic NUMA without cache coherence

Table 1. Experimental results for various matrix sizes and computation to communication speed relations

				Computation to memory speed					
				6:1			3:1		
recursion level	matrix size per processor	no. of SoCs	no. of processors	speedup vs. rec. level 0	speedup per processor	speedup vs. NUMA	speedup vs. rec. level 0	speedup per processor	speedup vs. NUMA
matrix size: 1024									
0	1024	1	1	1.0	1.00	1.00	1.0	1.00	1.00
1	512	1	8	7.9	0.99	1.01	8.0	1.00	1.00
2	256	4	64	62.9	0.98	1.02	63.4	0.99	1.01
3	128	16	512	491.9	0.96	1.04	501.7	0.98	1.02
4	64	256	4096	3766.5	0.92	1.08	3923.8	0.96	1.04
matrix size: 64									
0	64	1	1	1.0	1.00	1.04	1.0	1.00	1.02
1	32	1	8	7.4	0.92	1.12	7.7	0.96	1.06
2	16	4	64	50.8	0.79	1.24	56.3	0.88	1.14
3	8	16	512	315.6	0.62	1.39	383.6	0.75	1.24
4	4	256	4096	1650.2	0.40	1.48	2187.8	0.53	1.29

protocol. This result is a lower bound of improvement. The comparison to the real NUMA with data cache swapping and time consuming coherence protocols would be incomparably worse since in a standard NUMA system programs would be executed with cache swapping and coherence protocols. If 64x64 matrix multiplication would be further recursively decomposed according to our method, we obtain 7.4 speedup from 8 processors (with excellent efficiency 0.92), 50.8 speedup from 64 processors (with very good efficiency 0.79), 315.6 speedup from 512 processors (with acceptable efficiency 0.62) and 1650.2 speedup from 4096 processors (with efficiency of 0.40) at the consecutive recursion levels. We can accept the granularity of 8x8 multiplication in each processor with the efficiency of 0.62 as the lowest limit. It gives 1.39 times better speedup than the idealistic NUMA. Smaller computation grain (4x4) gives the efficiency below 0.5, although the speedup in this case is over 1.48 times better than for the idealistic NUMA.

We can approximate efficiency of multiplication of 1024x1024 matrices at further recursion levels, assuming that larger SoC modules will be used that will enable avoiding global communications also at higher recursion levels. With such assumption, all the elementary isolated subgraphs at higher recursion levels will be implemented entirely inside SoC modules. The elementary subgraphs (the requested SoC size) at the n-th recursion level is executed by 2n processors. So, at the 5-th recursion level the SoC module has to contain 32 processors and 64 memory modules. For multiplication of 1024x1024 matrices based on elementary 32x32 matrix multiplication in each processor, we should use a structure of 1024 (32-processor) SoCs with 32768 processors. It gives the approximate speedup of

27721 at the efficiency level of 0.84 with the improvement of 1.21 over idealistic NUMA. The use of 16x16 elementary matrix multiplication in each processor for the 1024x1024 problem gives the speedup of 191227 with the efficiency of 0.73 in a hierarchical structure of 4096 (64-processor) SoCs with 262144 processors - 1.34 times better than idealistic NUMA.

In the experiments described above, high communication improvement due to application of communication on the fly, was neutralized by a relatively large memory access latency comparing computation speed. This relation defines the acceptable lowest computation grain level for a given parallel program (problem size) in respect to the obtained efficiency (per processor). We checked how the situation would improve, if we doubled the memory speed.

Simulation results for the computation speed/memory speed relation equal to 3:1 are shown in Table 1. In the 1024x 1024 multiplication at the 4^{th} recursion level (64x64 elementary matrix multiplication) we obtain the speedup of 3924 out of 4096 processors with the efficiency of 0.96. It is 1.04 times better than for the idealistic NUMA. For the 64x64 matrix multiplication, the implementation based on 32x32 elementary multiplication gives speedup of 7.7 with the efficiency of 0.96. 16x16 elementary multiplication gives in this case much improved speedup of 56.3 out of 64 processors with efficiency 0.88 (1.14 times better than for NUMA). 8x8 and 4x4 elementary multiplications are still good with the efficiency of 0.75 and 0.53.

The use of 32x32 elementary multiplication for the execution of the 1024x1024 multiplication gives now the approximate speedup at the level of 30017. This gives excellent efficiency of 0.91 out of 1024 (32-processor) SoCs with 32768 processors. With 16x16 elementary multiplication we obtain the upper bound of speedup of 220870 with the efficiency of 0.84 from 16384 (64-processor SoCs).

5 Conclusions

New architecture of a system based on dynamic shared memory processor clusters with a new approach to inter-cluster communication has been discussed in the paper.

Simulation experiments were performed for matrix multiplication with recursive data decomposition with the use of a program graphs execution simulator. The experiments have shown that execution based on communication on the fly gives better results than execution in idealized standard NUMA architecture. It implies superiority of our architecture over real NUMA with data swapping and coherence protocols. In the studied example, all communication through global data network could be transformed into processor switching with data and local communication inside clusters.

Program execution speedup was sensitive to relations between processor speed and memory latency, which has defined the acceptable level of applied fine grain parallelism. Computation grain based 32x32 multiplications performed sequentially in parallel processors gives excellent parallel execution efficiency of 0.96 and 0.92 for speed relations 3:1 and 6:1. The grain of 16x16 matrix multipli-

cations was giving very good speedup with the efficiency of 0.88 and 0.79 for speed relations 3:1 and 6:1, respectively. Also the grain determined by 8x8 was acceptable giving the efficiency above 0.5 (0,75 and 0.53 for speedup relation of 3:1 and 6:1, respectively). 4x4 multiplication grain was giving efficiency above 0.5 only for the 3:1 speed relation.

The experiments confirmed suitability of the proposed architecture for fine-grained typical parallel numerical computations also for large problem sizes. That confirms high potential of contribution of the proposed solutions to the design of massively parallel systems for large scale intensive numerical computations.

References

1. L. Benini , G. de Michelli, Networks on Chips: A New SoC Paradigm, Computer, Jan. 2002, pp. 70-78.
2. Ch. Rowen, Engineering the Complex SOC, Fast, Flexible Design with Configurable Processors, Prentice Hall PTR, 2004.
3. D. M. Tullsen, S.J.Eggers, Effective Cache Pre-fetching on Bus Based Multiprocessors, ACM Trans. on Computer Systems, Vol.13, N.1 Feb. 1995, pp. 57-88
4. D. A. Koufaty et al. Data Forwarding in Scaleable Shared Memory Multi-Processors, IEEE Trans. on Parallel and Distr. Technology, Vol. 7, N. 12, 1996, pp. 1250-1264.
5. A. Milenkovic, V. Milutinovic, Cache Injection: A Novel Technique for Tolerating Memory Latency in Bus-Based SMPs, Proc. of the Euro-Par 2000, LNCS 1900, pp. 558-566.
6. M. Tudruj, L. Masko, Communication on the Fly and Program Execution Control in a System of Dynamically Configurable SMP Clusters, 11-th Euromicro Conf. on Parallel Distributed and Network based Processing, February, 2003, Genova - Italy, IEEE CS Press, pp. 67-74.
7. M. Tudruj, L. Masko, Communication on-the-fly in dynamic SMP clusters - towards efficient fine grain computations, Fifth Intl. Conference on Parallel Processing and Applied Mathematics, PPAM 2003, Czestochowa, Poland, Sept.2003, LNCS, Springer Verlag.
8. M. Tudruj, . Mako, Dynamic SMP Clusters with Communication on the Fly in NoC Technology for Very Fine Grain Computations, The 3rd Intl. Symp. on Parallel and Distributed Computing, Cork, Irlandia, 5-7 July 2004, IEEE Computer Society Press.
9. L. Masko, Program graph scheduling for dynamic SMP clusters with communication on the fly, International Symposium on Parallel and Distributed Computing ISPDC 2004, Cork, Irlandia, 5-7 July 2004, IEEE Computer Society Press.

Frequency of Co-operation of Parallel Simulated Annealing Processes

Zbigniew J. Czech[1] and Bożena Wieczorek[2]

[1] Silesia University of Technology, Gliwice,
and University of Silesia, Sosnowiec, Poland
zjc@polsl.gliwice.pl
[2] Katowice School of Economics, Katowice, Poland
bb@star.iinf.polsl.gliwice.pl

Abstract. A parallel simulated annealing algorithm to solve the vehicle routing problem with time windows is considered. The objective is to investigate how the frequency of co-operation of parallel simulated annealing processes influences the quality of solutions to the problem. The quality of a solution is measured by its proximity to the optimum solution.

Keywords: Parallel simulated annealing, frequency of co-operation of parallel processes, vehicle routing problem with time windows, bicriterion optimization.

1 Introduction

The vehicle routing problem with time windows (VRPTW) consists in establishing a set of routes beginning and ending at a depot which serves a set of customers. For the purpose of delivery (or pick up) there is a fleet of vehicles. The set of routes which solves the problem visits each customer exactly once, ensures that the service at any customer begins within the time window and preserves the vehicle capacity constraints. In addition, the set of routes should minimize, firstly, the number of vehicles used, and secondly, the total distance traveled by the vehicles.

In this work a parallel simulated annealing algorithm to solve the VRPTW is considered. The objective is to investigate how the frequency of co-operation of parallel simulated annealing processes influences the quality of solutions to the problem. The quality of a solution is measured by its proximity to the optimum solution.

The results of this work extend our previous results reported in [3, 4]. Parallel simulated annealing to solve the VRPTW is applied by Arbelaitz et al. [1]. Onbaşoğlu and Özdamar present the applications of parallel simulated annealing algorithms in various global optimization problems [5]. The methods of parallelization of simulated annealing are discussed by Azencott [2].

In section 2 the VRPTW is formulated. Section 3 describes the parallel simulated annealing algorithm. Section 4 discusses the experimental results. Section 5 concludes the work.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 43–50, 2006.

2 Problem Formulation

The VRPTW is formulated as follows. There is a central depot of cargo and n customers (nodes) located at the specified distances from the depot. The locations of the depot ($i = 0$) and the customers ($i = 1, 2, \ldots, n$), and the shortest distances $d_{i,j}$ and the corresponding travel times $t_{i,j}$ between any two locations i and j are given. The cargo have to be delivered to (or picked up from) each customer i according to the delivery demand q_i by a fleet of K vehicles. Each vehicle serves a subset of customers on the route which begins and ends at the depot. The vehicles have the same capacity Q. The total sum of demands of customers served by a vehicle on a route cannot exceed Q. For each customer a service time window $[e_i, f_i]$ and a service time h_i are defined. e_i and f_i determine, respectively, the earliest and the latest time for start servicing. The customer i is served by a single vehicle exactly once, within the time window $[e_i, f_i]$. The vehicle can arrive at the customer before the time window, but in such a case it has to wait until time e_i when the service can begin. The latest time for arrival of the vehicle at customer i is f_i. The time window is also defined for the depot, $[e_0, f_0]$, and it determines the time slot in which all deliveries should be effected. The aim is to find the set of routes which guarantees the delivery of cargo to all customers and satisfies the time window and vehicle capacity constraints. Furthermore, the size of the set equal to the number of vehicles needed (primary goal) and the total travel distance (secondary goal) should be minimized. The VRPTW is a NP-hard bicriterion optimization problem in which the optimization criteria are hierarchical. More formally, there are three types of decision variables in the problem. The first decision variable, $x_{i,j,k}$, $i, j \in \{0, 1, \ldots, n\}$, $k \in \{1, 2, \ldots, K\}$, $i \neq j$, is 1 if vehicle k travels from customer i to j, and 0 otherwise. The second decision variable, t_i, denotes the time when a vehicle arrives at customer i, and the third decision variable, b_i, denotes the waiting time at that customer. The aim is to:

$$\text{minimize} \qquad K, \quad \text{and then} \tag{1}$$

$$\text{minimize} \qquad \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} \sum_{k=1}^{K} d_{i,j} x_{i,j,k}, \tag{2}$$

subject to the following constraints:

$$\sum_{k=1}^{K} \sum_{j=1}^{n} x_{i,j,k} = K, \quad \text{for } i = 0, \tag{3}$$

$$\sum_{j=1}^{n} x_{i,j,k} = \sum_{j=1}^{n} x_{j,i,k} = 1, \quad \text{for } i = 0 \text{ and } k \in \{1, 2, \ldots, K\}, \tag{4}$$

$$\sum_{k=1}^{K} \sum_{j=0, j\neq i}^{n} x_{i,j,k} = \sum_{k=1}^{K} \sum_{i=0, i\neq j}^{n} x_{i,j,k} = 1, \quad \text{for } i, j \in \{1, 2, \ldots, n\}, \tag{5}$$

$$\sum_{i=1}^{n} q_i \sum_{j=0, j\neq i}^{n} x_{i,j,k} \leq Q, \text{ for } k \in \{1, 2 \ldots, K\} \tag{6}$$

$$\sum_{k=1}^{K} \sum_{i=0, i\neq j}^{n} x_{i,j,k}(t_i + b_i + h_i + t_{i,j}) \leq t_j, \text{ for } j \in \{1, 2, \ldots, n\} \tag{7}$$

$$\text{and } t_0 = b_0 = h_0 = 0,$$

$$e_i \leq (t_i + b_i) \leq f_i, \text{ for } i \in \{1, 2, \ldots, n\}. \tag{8}$$

Formulas (1) and (2) define the minimized functions. Eq. (3) specifies that there are K routes beginning at the depot. Eq. (4) expresses that every route starts and ends at the depot. Eq. (5) assures that every customer is visited only once by a single vehicle. Eq. (6) defines the capacity constraints. Eqs. (7)–(8) concern the time windows. Altogether, eqs. (3)–(8) define the feasible solutions to the VRPTW.

3 Parallel Simulated Annealing

The parallel simulated annealing algorithm, which we call the algorithm of co-operating searches (CS), comprises p processes, P_1, P_2, ..., P_p. Each of them generates its own annealing chain divided into two phases (Fig. 1). Each phase consists of some number of cooling stages, and each cooling stage consists of some number of annealing steps. The main goal of phase 1 is minimizing the number of routes of the solution, whereas phase 2 minimizes the total length of these routes. However in phases 1 and 2 both the number of routes and the total length of routes can be reduced. On every annealing step a neighbor solution is determined by moving one or more customers among the routes (line 13 in Fig. 1). Generally, in simulated annealing the neighbor solutions of lower costs obtained in this way are always accepted. The solutions of higher costs are accepted with the probability $e^{-\delta/T_i}$ where T_i, $i = 0, 1, \ldots, i_{\max}$, is a parameter called a temperature of annealing, which falls from some initial value T_0 according to the formula $T_{i+1} = \beta T_i$, where β ($\beta < 1$) is a constant and δ denotes an increase of the solution cost. A sequence of steps for which the temperature of annealing remains constant is called a cooling stage. The cost of solution s in phase 1 of our algorithm is computed as $cost_1(s) = c_1 N + c_2 D + c_3(r_1 - \bar{r})$ and in phase 2 as $cost_2(s) = c_1 N + c_2 D$, where N is the number of routes in solution s (equal to the number of vehicles needed), D is the total travel distance of the routes, r_1 is the number of customers in a route which is tried to be removed, $\bar{r}$ is an average number of customers in all routes, and c_1, c_2, c_3 are some constants. Since the basic criterion of optimization is the number of routes, it is assumed that $c_1 \gg c_2$.

In the parallel algorithm the processes co-operate with each other every ω annealing step of phase 2 passing their best solutions found so far (Fig. 2). The

```
1   Create the initial_solution by making use of some heuristics;
2   old_solution := initial_solution; best_solution := initial_solution;
3   for f := 1 to 2 do {execute phase 1 and 2}
      {beginning of phase f}
4     T := T_{0,f}; {initial temperature of annealing}
5     repeat {a cooling stage}
6       for i := 1 to L do
7         annealing_step_f(old_solution, best_solution);
8       end for;
9       T := β_f · T; {temperature reduction}
10    until a_f cooling stages are executed;
      {end of phase f}
11  end for;

12  procedure annealing_step_f(old_solution, best_solution);
13    Create new_solution as a neighbor to old_solution
        (the way this step is executed depends on f);
14    δ := cost_f(new_solution) − cost_f(old_solution);
15    Generate random x uniformly in the range (0, 1);
16    if (δ < 0) or (x < e^{−δ/T}) then
17      old_solution := new_solution;
18      if cost_f(new_solution) < cost_f(best_solution) then
19        best_solution := new_solution;
20      end if;
21    end if;
22  end annealing_step_f;
```

Fig. 1. Phase 1 and 2 of annealing chain

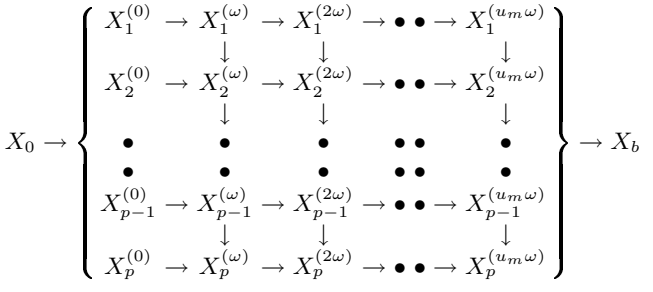


Fig. 2. Scheme of processes co-operation (X_0 – initial solution; X_b – best solution among the processes)

chain for the first process is completely independent. The chain for the second process is updated at steps $u\omega$, $u = 1, 2, \ldots, u_{\mathrm{m}}$, to the better solution between the best solution found by the first process so far, and the realization of the last step of simulated annealing of the second process. Similarly, the third process chooses as the next point in its chain the better solution between its own and

the one obtained from the second process. This means that the best solution found by process P_l is piped for further enhancement to processes $P_{l+1} \ldots P_p$.

The temperature of annealing decreases in phase 2 according to the formula $T_{i+1} = \beta_2 T_i$ for $i = 0, 1, 2, \ldots, a_2$, where a_2 is the number of cooling stages. In this work we investigate two cases in establishing the points of process co-operation with respect to temperature drops. In the first case processes co-operate frequently during each of temperature plateaus, $\omega = L/v$, where $v = \{50, 40, 20, 10, 5, 2\}$. In the second case, of rare co-operation, the processes interact at the end of each cooling stage, $\omega = L$, or there are several temperature drops before co-operation takes place, $\omega = vL$, $v = \{2, 3\}$.

4 Experimental Results

The parallel CS algorithm was serialized and implemented using C language and the following values of parameters: $c_1 = 40000$, $c_2 = 1$, $c_3 = 50$, $L = 100000$, 50000, 33320 and 25000, $a_1 = 40$, $a_2 = 100$, $\beta_1 = 0.95$, $\beta_2 = 0.98$. The initial temperature of annealing, $T_{0,f}$, was computed at the beginning of each annealing phase in such a way that the probability of worsening the solution cost by Δ in the first annealing step, $e^{-\Delta/T_{0,f}}$, was equal to some predefined constant (in our case 0.01).

The computational experiments were carried out on the R101, R108 and R112 test instances of the test set by Solomon (`http://w.cba.neu.edu /~msolomon /problems.htm`) which consists of 56 problem instances. In Table 1 the experimental results for the CS algorithm are shown. A single experiment consisted in creation an initial solution, X_0, to the VRPTW by making use of some heuristics. Then this solution was improved in a chain of simulated annealing steps executed by a given number of parallel processes. In order to ensure that the processes explore the parts of the space search of roughly the same size, we decreased the length of annealing chains while the number of processes grew up. For the numbers of processes $p = 5$, 10, 15 and 20, the numbers of annealing steps were set to $L = 100000, 50000, 33320$ and 25000, respectively. The columns of Table 1 contain: p – number of processes, ω – interval of co-operation of parallel processes, w_m – number of solutions with the minimum number of routes (e.g. 19 for the R101 test instance), H – number of hits into the best solution found (e.g. $\langle 19, 1650.79864 \rangle$ for the R101 test instance, where 19 is the number of routes and 1650.79864 the total distance of these routes; the best result for this test reported in the literature equals $\langle 19, 1645.79 \rangle$), $\bar{y}$ – mean value of total distances for solutions with the minimum number of routes (since total distances of routes are minimized, the lower value of $\bar{y}$ the better), s – standard deviation of total distances, z – value of test statistics. It can be seen from Table 1 (column R101) that all values of w_m are equal 500 in 500 experiments, what means that the R101 test instance is very easy to solve with respect to finding solutions with the minimum number of routes. The numbers of hits into the best solution, H, are larger for higher frequencies of processes co-operation. To support statistically the hypothesis that higher frequency of co-operation give better results, we

Table 1. Experimental results for R101, R108 and R112 test instances (values in each row were computed in 500 experiments)

R101							R108							R112						
p	ω	w_m	H	$\bar{y}$	s	z	p	ω	w_m	H	$\bar{y}$	s	z	p	ω	w_m	H	$\bar{y}$	s	z_1
5	$L/50$	500	7	1652.8	1.3	13.6	5	$L/50$	466	0	971.7	6.7	2.0	5	$L/50$	27	1	1009.7	14.4	–
10	$L/50$	500	63	1651.8	1.1	21.5	10	$L/50$	467	0	972.0	7.1	2.4	10	$L/50$	27	0	1008.6	14.3	–
15	$L/50$	500	118	1651.6	1.0	21.6	15	$L/50$	459	1	972.6	7.4	1.1	15	$L/50$	28	0	1007.6	9.9	–
20	$L/50$	500	163	1651.5	1.1	20.6	20	$L/50$	442	0	972.9	8.1	0.6	20	$L/50$	30	0	1012.5	9.0	–
5	$L/40$	500	14	1652.8	1.3	12.5	5	$L/40$	472	0	971.7	6.5	2.0	5	$L/40$	28	0	1006.9	12.2	0.8
10	$L/40$	500	54	1651.9	1.1	20.3	10	$L/40$	456	1	972.2	7.6	1.9	10	$L/40$	26	0	1008.1	12.4	0.1
15	$L/40$	500	111	1651.5	0.9	22.7	15	$L/40$	455	0	972.6	7.7	1.0	15	$L/40$	38	0	1011.7	12.7	-1.5
20	$L/40$	500	131	1651.5	1.0	20.3	20	$L/40$	440	0	972.8	7.7	0.7	20	$L/40$	34	0	1013.8	12.0	-0.5
5	$L/20$	500	13	1652.8	1.3	13.6	5	$L/20$	470	1	971.9	6.7	1.5	5	$L/20$	26	0	1006.2	11.7	1.0
10	$L/20$	500	39	1652.0	1.1	19.3	10	$L/20$	479	3	971.8	6.3	3.1	10	$L/20$	35	0	1011.9	13.2	-0.9
15	$L/20$	500	80	1651.7	1.2	17.9	15	$L/20$	449	3	972.2	7.2	2.0	15	$L/20$	28	0	1007.6	13.3	0.0
20	$L/20$	500	116	1651.5	1.0	21.1	20	$L/20$	441	0	972.5	7.8	1.3	20	$L/20$	19	0	1010.3	13.9	0.6
5	$L/10$	500	10	1652.8	1.3	13.3	5	$L/10$	475	3	971.7	6.0	2.2	5	$L/10$	25	0	1008.3	10.8	0.4
10	$L/10$	500	42	1652.0	1.1	18.3	10	$L/10$	463	1	971.7	6.9	3.1	10	$L/10$	31	0	1010.2	12.9	-0.4
15	$L/10$	500	59	1651.7	1.0	19.2	15	$L/10$	462	0	972.0	7.2	2.3	15	$L/10$	37	0	1009.5	9.9	-0.8
20	$L/10$	500	94	1651.6	1.0	20.2	20	$L/10$	445	2	973.3	8.3	-0.3	20	$L/10$	34	0	1009.2	10.2	1.4
5	$L/5$	500	10	1652.9	1.3	11.9	5	$L/5$	473	0	972.4	6.7	0.4	5	$L/5$	23	0	1008.9	11.7	0.2
10	$L/5$	500	32	1652.1	1.2	17.0	10	$L/5$	475	0	972.0	6.8	2.4	10	$L/5$	25	0	1007.2	11.3	0.4
15	$L/5$	500	64	1651.8	1.1	17.7	15	$L/5$	459	1	973.2	8.3	-0.1	15	$L/5$	24	0	1010.0	13.1	-0.8
20	$L/5$	500	93	1651.8	1.2	15.8	20	$L/5$	457	0	973.1	8.6	0.1	20	$L/5$	37	0	1004.0	11.2	3.4
5	$L/2$	500	6	1652.9	1.2	12.1	5	$L/2$	470	0	971.9	6.6	1.7	5	$L/2$	21	0	1005.4	9.9	1.2
10	$L/2$	500	15	1652.2	1.1	16.3	10	$L/2$	467	1	971.6	6.9	3.3	10	$L/2$	23	0	1006.9	10.6	0.5
15	$L/2$	500	31	1652.0	1.1	15.3	15	$L/2$	468	0	972.1	7.9	2.0	15	$L/2$	18	0	1005.8	11.5	0.6
20	$L/2$	500	61	1651.9	1.3	13.7	20	$L/2$	477	0	972.1	7.8	2.2	20	$L/2$	28	0	1007.8	11.6	1.7
5	L	500	3	1653.5	1.3	4.6	5	L	472	1	972.7	7.0	-0.2	5	L	35	1	1006.6	12.2	0.9
10	L	500	5	1653.1	1.3	5.1	10	L	466	3	972.3	7.4	1.7	10	L	22	0	1002.3	10.8	1.8
15	L	500	5	1652.8	1.2	5.3	15	L	455	3	972.4	7.3	1.6	15	L	28	2	1004.4	14.2	1.0
20	L	500	6	1652.6	1.3	5.7	20	L	466	0	972.9	7.6	0.5	20	L	30	0	1005.8	11.3	2.6
5	$2L$	500	0	1653.6	1.3	4.3	5	$2L$	463	3	972.4	6.7	0.3	5	$2L$	29	0	1001.6	12.7	2.2
10	$2L$	500	4	1653.1	1.3	4.8	10	$2L$	463	1	973.0	7.4	0.2	10	$2L$	21	0	1005.8	13.9	0.7
15	$2L$	500	2	1652.8	1.3	4.3	15	$2L$	471	0	972.5	7.4	1.3	15	$2L$	30	0	1005.2	12.6	0.8
20	$2L$	500	8	1652.7	1.3	5.5	20	$2L$	461	2	973.2	7.8	0.0	20	$2L$	25	0	1006.0	10.3	2.5
5	$3L$	500	0	1653.9	1.4	–	5	$3L$	457	3	972.6	6.5	–	5	$3L$	38	0	1006.4	11.8	1.0
10	$3L$	500	0	1653.5	1.3	–	10	$3L$	465	1	973.1	6.9	–	10	$3L$	25	0	1007.5	13.1	0.3
15	$3L$	500	5	1653.2	1.4	–	15	$3L$	452	0	973.1	7.8	–	15	$3L$	25	0	1003.3	12.1	1.4
20	$3L$	500	5	1653.1	1.4	–	20	$3L$	446	2	973.2	7.1	–	20	$3L$	25	1	1004.3	13.2	2.7

use the mean values, $\bar{y}$, and standard deviations, s. Let us test the hypothesis $H_0 : \mu_A \leq \mu_B$ versus the alternative hypothesis $H_a : \mu_A > \mu_B$, where μ_A denotes (for a given number of processes p) the mean value of the population of the total routes distances for the experiment with $\omega = 3L$, and μ_B the mean values of populations for the experiments with $\omega = L/v$, $v \in \{\frac{1}{2}, 1, 2, 5, 10, 20, 40, 50\}$. In all cases for which hypothesis H_0 is rejected we can claim that the intervals of co-operation $\omega = L/v$, $v \in \{\frac{1}{2}, 1, 2, 5, 10, 20, 40, 50\}$, give superior solutions with respect to proximity to the optima as compared to the interval $\omega = 3L$. Using the test statistics: $Z = (\bar{y}_A - \bar{y}_B)/\sqrt{s_A^2/n_1 + s_B^2/n_2}$ we reject hypothesis H_0 at

the $\alpha = 0.01$ significance level, if $Z > Z_{0.01} = 2.33$. The calculated values of the test statistics are given in column z of Table 1 (R101). The values of z for this test instance indicate that the higher frequency of co-operation of processes the better quality of solutions.

The results for the R108 test instance show that this test is more difficult to solve than the previous one. The values of w_m lie between 440 and 477, and the numbers of hits into the best solution (equal to the best solution reported in the literature, $\langle 9, 960.87528 \rangle$) are significantly smaller than those for the R101 test instance. In terms of frequency of co-operation of processes, the values of z imply — although not so clearly as for the R101 test instance — that more desirable are higher frequencies of co-operation. For example, for $p = 10$ the values of z belong to the range $2.4 \ldots 3.3$ for frequencies between $L/20$ and $L/2$, what suggests that these frequencies give solutions of the best quality.

The results in Table 1 prove that the R112 test instance is the hardest to solve among the tests we investigated. Merely less than 40 solutions among 500 had the minimum number of routes (cf. w_m column), and in 18000 experiments the best solution known in the literature, $\langle 9, 982.13919 \rangle$, was hit only 5 times (cf. H column). For this test instance the smallest values of $\bar{y}$ were obtained for a low frequency of processes co-operation. Therefore here we assume that μ_A and μ_B (from hypotheses H_0 and H_a) denote the mean value of the population of the total routes distances for the experiment with $\omega = L/50$, and the mean values of populations for the experiments with $\omega = L/v$, $v \in \{\frac{1}{3}, \frac{1}{2}, 1, 2, 5, 10, 20, 40\}$, respectively. The values of the modified test statistics, given in column z_1, show that for achieving solutions of good quality, low frequencies of processes co-operation should be used.

5 Conclusions

Simulated annealing processes can co-operate with various frequency. One may hypothesize that in order to get good results the frequencies of processes co-operation are to be high. It turned out however that high frequencies of co-operation have their advantages and disadvantages. On the one hand, if the processes exchange their best solutions very frequently, the probability for improving the quality of the final solution increases. But on the other hand, the higher frequency of co-operation the shorter Markov chains within which the processes explore the search space freely. It follows from the theory and practice of simulated annealing that in general longer Markov chains are more desirable. Our experiments shown that for more difficult[1] tests, like R112, better results were obtained when the frequency of processes co-operation was low, i.e.

[1] We measure the difficulty of a test instance by two probabilities: Pr_1 — probability that after execution of the algorithm, a solution with the minimum number of routes is found, and Pr_2 — probability that a total length of routes of that solution is not worse than by 1% with respect to the shortest length achieved. In our experiments these probabilities for the R112 test instance were 0.08 and 0.12, respectively, and for the R101 test — 1.0 and 1.0.

when the processes executed longer annealing chains. For easier tests, like R101 and R108, higher frequencies of co-operation proved advantageous.

Acknowledgements

We thank Piotr Czarnas and Przemyslaw Gocyla for their contributions to this work. We also thank the following computing centers where the computations of our project were carried out: Academic Computer Centre in Gdansk TASK, Academic Computer Centre CYFRONET AGH, Krakow (computing grants 027/2004 and 069/2004), Poznan Supercomputing and Networking Center, Interdisciplinary Centre for Mathematical and Computational Modelling, Warsaw University (computing grant G27-9), Wroclaw Centre for Networking and Supercomputing (computing grant 04/97). The research of this project was supported by the State Committee for Scientific Research grants 3 T 11F 00429 and BK-239/RAu2/2005.

References

1. Arbelaitz, O., Rodriguez, C., Zamakola, I., Low cost parallel solutions for the VRPTW optimization problem, Proceedings of the International Conference on Parallel Processing Workshops, (2001).
2. Azencott, R., (Ed.), Simulated annealing. Parallelization techniques, J. Wiley, NY, (1992).
3. Czarnas, P., Czech, Z.J., Gocyla, P., Parallel simulated annealing for bicriterion optimization problems, Proc. of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM'03), (September 7–10, 2003), Częstochowa, Poland, 233–240.
4. Czech, Z.J., Czarnas, P., A parallel simulated annealing for the vehicle routing problem with time windows, Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain, (January, 2002), 376–383.
5. Onbaşoğlu, E., Özdamar, L., Parallel simulated annealing algorithms in global optimization, Journal of Global Optimization 19: 27–50, 2001.

Maximal Group Membership in Ad Hoc Networks

Mamoun Filali[1], Valérie Issarny[2], Philippe Mauran[3], Gérard Padiou[3], and Philippe Quéinnec[3]

[1] IRIT-CNRS-Université Paul Sabatier
filali@irit.fr
[2] INRIA-Roquencourt
issarny@inria.fr
[3] IRIT-ENSEEIHT
{mauran, padiou, queinnec}@enseeiht.fr

Abstract. The notion of Group communication has long been introduced as a core service of distributed systems. More recently, this notion appeared with a somewhat different meaning in the field of mobile ad hoc systems. In this context, we study the group membership problem. After specifying the basic safety properties of such groups and a maximality criterion based on cliques, we propose a group membership algorithm. Lastly, with respect to this criterion, we compare our algorithm with two group membership algorithms for ad hoc environments. Moreover, a formal description in TLA+ has been programmed and verified by model-checking for small networks.

1 Introduction

The notion of Group communication has long been introduced as a core service of distributed systems [1]. More recently, this notion appeared with a somewhat different meaning in the field of mobile ad hoc systems. We introduce group communication protocols in the classical setting. Then, we present the features of mobile ad hoc systems that motivate the design of new definitions and protocols for group communication.

Group Communication Protocols. Group communication services have emerged from two domains : asynchronous distributed systems for fault-tolerant purposes [1, 3, 6] and (distributed) multi-agent systems (MAS) for agent coordination purposes [8]. They have been extensively studied from a formal as well as from a practical standpoint, in the field of distributed computing systems [3] in which:

- the number of connected nodes can be sizeable, but (typically) is not huge.
- the connection of a node to the system remains relatively stable (which does not rule out unexpected failures or disconnections).

In this setting, the group communication service appears as a fundamental service, which allows to multicast messages to a set of nodes (or *group members*), in order to build (more easily) fault-tolerant distributed systems, or to manage the consistency of the group members' interactions, at the application level.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 51–58, 2006.

Group communication is based on the definition of *groups*, i.e. sets of nodes. Messages sent to a group are dispatched to each member of the group. Such a group communication layer is particularly relevant and useful for fault tolerance based on the replication of services or data, or for some classes of applications, such as groupware. Its implementation rests on two basic services:

- a *group membership service*, which maintains a list of current group members, or, more precisely, a representation (a *view*) of this list for each group member. The main problem is to ensure the coherence of each member's view. Changes in the group composition, due to members leaving, or failing, or to new members joining, can be taken into account by the notion of primary view, which defines consistency rules for group evolution [4].
- a *reliable multicast service* which delivers messages to each group member. This level allows to schedule message delivery, depending on the application level requirements in terms of cost and consistency.

Group Communication Protocols in Mobile Ad Hoc Systems. Pervasive computing, which has received an increasing attention during the last decade, brings quite a different setting for distributed systems and applications:

- The ability for any node to join (or quit) the system anywhere, at any time, a priori rules out asymetrical (centralized) protocols in the vein of client/server patterns.
- Scalability becomes a key factor, due to the potentially huge and highly transient set of nodes that make up the system.
- Whereas the design of classical group communication protocols relies on point-to-point communication, *local broadcast* is a natural and often appropriate communication pattern in pervasive computing environments.
- In the same way as networks, groups are not a priori defined. They are rather built up "on the fly", in an ad hoc manner, depending on the requirements and availability of interacting sites at a given time, in a given area.
- Lastly, requirements on autonomy and resource consumption lead to favor robustness and locality, to the detriment of determinism and synchronism, which meets up with the stress layed on scalability.

Thus differences appear on the one hand, in the purpose of groups : in classical group communication systems, the main goal is to determine whether sites belong to a given (unique) group, whilst in the setting of pervasive computing systems, the main issue is to build groups (which may be partitioned) out of neighboring nodes, and, on the other hand, in the way groups evolve : in the classical setting, view updates are incremental and no periodic installation of new views occurs "from scratch".

The notion of group has thus been revisited, in the light of these new constraints, by several recent systems. Our proposal comes within this scope. It is based on the analysis of two existing proposals [2, 8], coming from different, but complementary, fields and concerns, as regards the group communication service, namely mobile ad hoc networks, and (embedded) multi-agent systems. Furthermore, we specify the properties of our protocol in the TLA+ formal framework [5], in order to assess and compare this protocol to existing ones.

2 Group Membership Properties in Ad Hoc Networks

In ad hoc networks, nodes and links continuously appear and disappear. In such a context, group members must exhibit a high connectivity to meet robustness and fault tolerance criteria. The most robust groups of processes of a given size are those that correspond to cliques in the underlying interconnection topology of the network.

Fig. 1. Partitions in cliques

A clique of a graph is any complete subgraph [7]. Figure 1 illustrates partitions in cliques of a graph. Several maximality criteria have been defined on cliques. With respect to the group membership problem, the left partition is better than the right one: groups have more members (i.e. the partition has less cliques). However, two cliques cannot be merged to form a bigger clique. We choose this non-extendible property as a maximality criterion.

Cliques can be used to specify the basic properties of groups. Each grouped process must eventually obtain an *installed view* that contains the members of its current clique. Group membership algorithms should aim at computing partitions of maximal cliques to assign a view to each node. A partition insures that each process belongs exactly to one group and (maximal) cliques provide the most robust groups.

A formal statement of these properties is given by the following definitions: we consider a set of nodes *Node*, the vertices of the network graph. This graph is specified as a set of pairs of nodes and installed views are subsets of *Node*.

$$Graph \subseteq Node \times Node \qquad Views \in [Node \rightarrow \text{SUBSET} \quad Node]$$

Communication properties in ad hoc networks lead to consider specific graphs. More precisely, we assume that a node can send messages to itself and that if a node p can communicate with q, then q can communicate with p. Therefore the graphs we consider for ad hoc networks are reflexive and symmetric:

$$AdHocGraph \triangleq Graph \cup \{\langle p, p\rangle : p \in Node\} \cup \{\langle q, p\rangle : \langle p, q\rangle \in Graph\}$$

In the remainder of this section, we assume the installed views are **non empty** sets. Views must verify consistency properties: an installed view of node p always contains p, views are cliques and views are disjoint sets.

View safety

$$\begin{array}{ll} \forall p \in Node: & p \in View[p] \\ \forall p \in Node: & View[p] \times View[p] \subseteq AdHocGraph \\ \forall p, q \in Node: & (View[p] = View[q]) \ \vee \ (View[p] \cap View[q] = \emptyset) \end{array}$$

First, we specify a local minimal requirement for a group membership service. Views with only one member (*singleton* set) should be avoided: for any couple (p, q) of nodes, if their installed views are eventually reduced to a singleton set, these nodes must not be neighbors. In other words, if a node eventually belongs to a singleton set, its neighbors belong to non singleton cliques.

$$\forall p, q \in Node : View[p] = \{p\} \wedge View[q] = \{q\} \Rightarrow \langle p, q \rangle \notin AdHocGraph$$

A stronger property specifies a maximal criterion : a view cannot be extended with an other distinct view.

$$\forall p, q \in Node : (View[p] \times View[q] \subseteq AdHocGraph) \Rightarrow View[p] = View[q]$$

A restricted form of this property implies that a singleton view cannot extend an existing view.

When all nodes have obtained an installed view, a final requirement states that all views are a covering of the graph : $\bigcup_{p \in Node} View[p] = Node$

From their mutually exclusive property, it follows that the set of views $\{View[p] : p \in Node\}$ is a partition of $Node$.

3 Group Membership Algorithm

The group membership algorithm aims at building non extendible cliques. Each node has the same behaviour. Starting from a *singleton* state, a node performs successive steps bounded by a timeout to reach a final *grouped* state. Figure 2 illustrates this sequence of phases. Such a sequence starts according to a classical approach in distributed algorithms, namely, a diffusing computation. At least one node performs a first local broadcast of a message (possibly after a timeout). Other nodes enter the phase when they receive this message and propagate the current phase by broascasting the same message type.

This sequence of phases is performed by a node until it enters the grouped state. A node remains in a group for a while and repeats the group membership protocol. The lifetime of a group is assumed to be much longer than the time required to build a group. This periodic behavior allows to adapt to the dynamic nature of the network.

During the *Discovering* phase, a node acquires the list of its one-hop neighbors. Then, each node broadcasts this list during the *Publishing* phase. When a

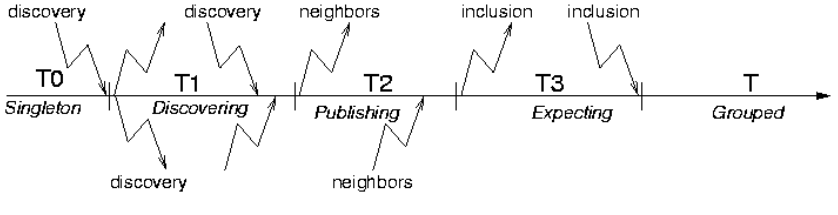


Fig. 2. Phases of the algorithm

node has received all the lists of its neighbors, it has a complete knowledge of its neighborhood at a 2-hops distance. With this knowledge, each node either decides to wait an inclusion request from an other node or to compute a clique and broadcast this view to target members. This decision relies upon a total priority order defined on nodes. A node evaluates a new view if and only if its priority is greater than all its 2-hops neighbors.

Figure 3 illustrates a node with its neighbors and 2-hops neighborhood. Node priority is assumed to be equal to their identity. This node of maximal priority will decide to build a view including either the nodes {8,10,15} or the nodes {7,11,15}. The clique {6,15} could also be considered, but larger cliques should be preferred.

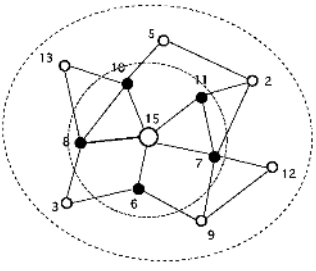


Fig. 3. Neighborhoods

The main idea for choosing the maximum over the 2-hops neighborhood is that the same node cannot be twice chosen to be a member of two distinct views. If the same node could be selected by two other nodes, then the distance between these two nodes should be at most 2. It follows that the node will be selected by at most one of them since the priority of a node that selects a view is greater than the priority of any other node at distance 2.

Properties of such algorithms can only be specified under stability assumptions on the underlying network during a bounded period. Henceforth, we assume that the underlying network connections are stable from the beginning of the protocol until all nodes are grouped. However, if this stability condition does not hold, the algorithm still guarantees the three view safety properties (See section 2), but cannot guarantee any longer the maximality property (non extendible property).

3.1 State Transition Graph

Figure 4 describes the state transitions of a node. A transition is performed when a message is received or a timeout occurs. During a phase, specific message type(s) can be broadcast (`!m`) and/or received (`?m`).

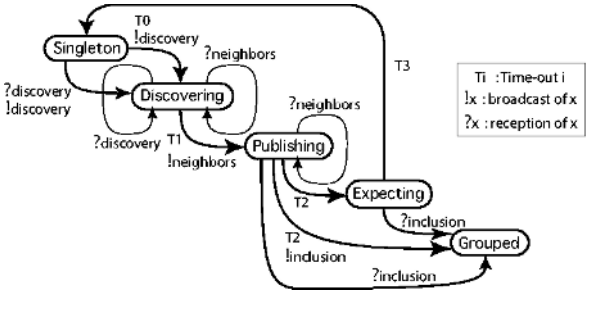


Fig. 4. State transition graph

There are three message types containing the sender's identity: *discovery*, *neighbors* and *inclusion*. A *neighbors* message also contains its sender neighbors and an *inclusion* message contains the resulting view.

Each node maintains the following local variables: its current state, a set of neighbors, the neighborhood of each of its neighbors and a view defined when the node is grouped.

3.2 Message Handling

For a singleton node, the reception of a first *discovery* message or a timeout `T0` occurrence triggers a transition toward the *Discovering* state and a broadcast of a `discovery` message toward reachable nodes. The sending node is recorded as a new neighbor and the current state remains or becomes *Discovering*. The discovery process is propagated by broadcasting a `discovery` message. If the node is already in the *Discovering* state, the same actions are performed but the state remains unchanged.

In the *Discovering* or *Publishing* state, if a node receives a `neighbors` message, its neighborhood is updated with the content of the message, namely the neighbors of the sending node.

In the *Expecting* or *Publishing* state, if a node receives an `inclusion` message, it accepts the content of the message as its current view and becomes grouped.

3.3 Timeout Handling

When a timeout occurs, according to the current state of the node, a transition is performed. We assume the propagation time of a message to be the dominant duration. Timeout choice rests upon the following constraints:

- Timeout `T0`: must be long enough to mask the asynchronous timing of phases among nodes ; nodes acquire a weak synchronous behavior after the *Discovering* phase and remain at most shifted from one phase.
- Timeout `T1`: in the *Discovering* state, a node has to broadcast its identity and receive its neighbors. As neighbors broadcast their identity as soon as they receive a `discovery` message, `T1` must be at least longer than 2 broadcasts (plus enough time for local processing) ;
- Timeout `T2`: same constraints as `T1` in so far as the nodes have the same behavior with respect to the `neighbors` messages.
- Timeout `T3`: in the *Expecting* state, a node has at most to wait for the choice of a view and its broadcast. Therefore, `T3` must be longer than this duration.

From the *Singleton* state, a timeout occurrence triggers a transition toward the *Discovering* state and a local broadcast of a `discovery` message that contains the sending node name.

From the *Discovering* state, a timeout occurrence triggers a transition toward the *Publishing* state and a local broadcast of a `neighbors` message which contains the current neighbors list of the sending node.

From the *Publishing* state, a timeout occurrence either leads to evaluate a new view if the current node has the maximal priority over its 2-hops neighborhood and to enter the *Grouped* state or to wait for an `inclusion` message in the *Expecting* state.

From the *Expecting* state, when a timeout occurs, the node returns into the *Singleton* state.

Number of Messages. In the best case, the network itself is a clique, and one iteration of the protocol is enough to build a group with all the nodes; if N is the number of nodes, it needs N broadcasts (`discovery` message) + N broadcasts (`neighbors` message) + 1 broadcast (`inclusion` message).

In the worst case, the network is linear and nodes are placed according to their priority (that is, nodes 1 and 2 are connected, nodes 2 and 3 are connected,...). Then, each iteration builds only one group with the two highest priority nodes. Then $N/2$ iterations and $O(N^2)$ broadcasts are required.

4 Related Algorithms

With respect to the general group membership problem, our study is mainly concerned by group construction algorithms in partitionable networks [9]. In this section, we first make some general remarks about group construction algorithms used in partitionable networks, then, we present the main features of two algorithms: the first one is used in the context of ad hoc networks [2], while the second is used in the context of multi-agent systems [8].

Views are close to the underlying network topology. A view is a connected component: it contains processes that can be reached (through one or several hops) from each other. However, this definition is usually strengthened: a view contains processes that can reach each other in one hop, i.e., a view is a clique of the underlying network. Moreover, initially, either a node knows its immediate neighbors or has to discover them through a broadcasting primitive.

The algorithm of [2], the starting point of our study, concerns group management in mobile ad hoc networks. Although, the algorithm is also concerned by security aspects as well as by application level aspects, we discuss here group membership aspects only. First, in order to take into account the dynamic topology of ad hoc networks and the resource consumption constraints of mobile applications group maintenance is periodic. Periodically, during a discovery phase, a node builds dynamically its neighbors set. Once a node knows its neighbours[1], it sends to *one* of them (the maximal one) its set of neighbours. Then, nodes which have received some sets of neighbors may choose to build a connected view. This algorithm cannot avoid extendible views. However, it has good properties with respect to the number of messages and the minimization of energy.

The algorithm of [8] concerns also group membership. This algorithm is based upon the construction of a common knowledge amongst neighboring nodes. Each node broadcasts the list of its neighbors which is assumed to be initially known.

[1] Timeouts are used for bounding the discovery period.

Then, each node gathers the neighborhood of its neighbors. Once this common knowledge is built, the following stages differ from our algorithm, as this knowledge is used to define views through a consensus: in the last stage of a loop iteration, the participants have to agree on a common view. In order to avoid cycles, clique choices have to be restricted over iterations and the algorithm tolerates extendible cliques to ensure the convergence of the loop. This algorithm is more concerned by the complexity issues for computing cliques: basically, it uses pre-specified thresholds over the size of cliques.

5 Conclusion

In this paper, we have been mainly concerned by group membership protocols in ad hoc networks. After specifying its basic safety properties and a maximality criterion about the installed views, we have proposed a group membership algorithm. Lastly, with respect to this criterion, we have compared our algorithm with two group membership algorithms. We have also specified the properties of the algorithm as well as its description in the TLA+ formalism. Currently, we have performed model-checking experiments with the TLC tool [5]. On small size graphs (6 nodes), we have been able to automatically check the correctness of our algorithm. We are now working on its formal correctness (any number of nodes) through theorem proving techniques.

References

1. K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.
2. M. Boulkenafed, V. Issarny, and J. Liu. Group management for in-home ad hoc networks. In *ECRTS International Workshop on Real-Time for Multimedia - Special Focus on Real-time Middleware for Consumer Electronics (RTMM)*, June 2004.
3. G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications : A comprehensive study. *ACM Computing Surveys*, 33(4):427–469, December 2001.
4. R. De Prisco, A. Fekete, N. Lynch, and A. Shvartsman. A dynamic view-oriented group communication service. In *Proceedings of the 17th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 227–236, June 1998.
5. L. Lamport. *Specifying Systems : The TLA+ language and tools for Hardware and Software Engineers*. Addison Wesley Professional, 2002.
6. S. Mishra, C. Fetzer, and F. Cristian. The Timewheel group communication system. *IEEE Transactions on Computers*, 51(8):883–899, August 2002.
7. S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, pages 215,217–218. Addison-Wesley, Reading MA, 1990.
8. P. Tosic and G. Agha. Maximal clique based distributed group formation for task allocation in large-scale multi-agent systems. In *Proceedings of the International Workshop on Massively Multi-Agent Systems, Kyoto, Japan*, December 10-11 2004.
9. Özapl Babaoğlu, R. Davoli, and A. Montresor. Group communication in partitionable systems : Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336, April 2001.

Multi-thread Processing of Long Aggregates Lists

Marcin Gorawski and Rafal Malczok

Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice, Poland
{Marcin.Gorawski, Rafal.Malczok}@polsl.pl

Abstract. In this paper we present a solution called Materialized Aggregate List designed for the storing and processing of long aggregate lists. An aggregate list contains aggregates, calculated from the data stored in the database. In our approach, once created, the aggregates are materialized for further use. The list structure contains a table divided into pages. We present three different multi-thread page-filling algorithms used when the list is browsed. The Materialized Aggregate List can be applied as a component of a node on every aggregation level in indexing structures, such as, an aR-tree. We present test results estimating an efficiency of the proposed solution.

1 Introduction

Query evaluation time in relational data warehouse implementations can be improved by applying proper indexing and materialization techniques. View materialization consists of first processing and then storing partial aggregates, which later allows the query evaluation cost to be minimized, performed with respect to a given load and disk space limitation [8]. In [1, 4] materialization is characterized by workload and disk space limitation. Indices can be created on every materialized view. In order to reduce problem complexity, materialization and indexing are often applied separately. For a given space limitation the optimal indexing schema is chosen after defining the set of views to be materialized [2]. In [5] the authors proposed a set of heuristic criteria for choosing the views and indices for data warehouses. They also addressed the problem of space balancing but did not formulate any useful conclusions. [7] presents a comparative evaluation of benefits resulting from applying views materialization and data indexing in data warehouses focusing on query properties. Next, a heuristic evaluation method was proposed for a given workload and global disk space limitation. We are working in the field of spatial data warehousing. Our system (Distributed Spatial Data Warehouse – DSDW) presented in [3] is a data warehouse gathering and processing huge amounts of telemetric information generated by the telemetric system of integrated meter readings. The readings of water, gas and energy meters are sent via radio through the collection nodes to the telemetric server. A single reading sent from a meter to the server contains a timestamp, a meter identifier, and the reading values. Periodically the extraction system loads the data to the database of our warehouse.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 59–66, 2006.

In our current research we are trying to find the weakest points of our solution. After different test series (with variations of aggregation periods, numbers of telemetric objects etc.) we found that the most crucial problem is to create and manage long aggregate lists. The aggregate list is a list of meter reading values aggregated according to appropriate time windows. A time window is the amount of time in which we want to investigate the utility consumption. The aggregator is comprised of the timestamp and aggregated values.

When we want to analyze utility consumption we have to investigate consumption history. That is when the aggregate lists are useful.

2 Motivation

In the system presented in [3] aggregate lists are used in the indexing structure, aggregation tree, that is a modification of an aR-Tree [6]. Every index node encompasses some part of the region where the meters are located and has as many aggregate lists as types of meters featured in its region. If there are several meters of the same type, the aggregate lists of the meters are merged (aggregated) into one list of the parent node.

The aggregate lists are stored in the main computer memory. Memory overflow problems may occur when one wants to analyze long aggregation periods for many utilities meters. If we take into consideration the fact that the meter readings should be analyzed every thirty minutes, simple calculations reveal that the aggregate list grows very quickly with the extension of an aggregation period. For instance, for single energy meter an aggregate list for one year has $365 \cdot 48 = 17520$ elements. In order to prevent memory overflows we designed a memory managing algorithm applied in the system presented in [3]. The mechanism defines a memory limit when the system starts. The limit is always checked before some new aggregate list is created. If upon being loaded a new list threatens to exceed a limit, the mechanism searches for a less frequently read node in the indexing structure and removes its aggregate lists from the memory, providing space for the new lists. The mechanism performs well when system uptime is not long. The creation and removal of aggregate list produces memory fragmentation that results in memory overflow errors, even though the memory limit had not been exceeded. Hence we decided to apply a new approach to storing and processing aggregate lists with no length limitations. We named our solution a Materialized Aggregate List (MAL).

The main idea of the proposed solution is to provide a user with a simple interface based on the standard Java list mechanism – a set of two functions: $hasNext()$ and $next()$ which permits the convenient browsing of the list contents. Our main goal was to create a list that could be used as a tool for mining data from the database as well as a component of indexing structure nodes.

3 MAL Details

Our main intention when designing the MAL was to build a solution free of memory overflows which would allow aggregate list handling with no length

limitations. We want to apply this solution in an indexing structure, such as aR-tree. In indexing structure the MAL can be used as a node component in both the lowest level nodes (tree leaves) for retrieving aggregates from the database and in nodes on higher aggregation levels for retrieving aggregates from child nodes. We applied the following approach: every list iterator uses a table divided into pages (the division is purely conventional). When an iterator is created some of the pages are filled with aggregators (which pages and how many is defined by the applied page-filling algorithm, see description below). Applying a multi-thread approach allows filling pages while the list is being browsed. The solution also uses an aggregates materialization mechanism that strongly speeds up the aggregates retrieval. The configuration aspects concern the following: the number of pages, the size of a single page, the number of available database connections and the applied page-filling algorithm.

The actual list operation begins when a new iterator is created (*iterator*() function call). The iterator gets a table from the table pool (see description below) and two values are calculated:

- border date. The border date is used for identifying page and managing the materialized data. The border date is calculated by repeatedly adding to the install date (defined in category block of the configuration file) a width of aggregation window multiplied by the size of the table page. The border date is equal to the timestamp of the first aggregator in the page.
- starting index. In the case that starting date given as a parameter in the *iterator*() function call is different from the calculated border date, the iterator index is adjusted so that a the first *next*() function call returns the aggregator with the timestamp nearest to the given starting date.

Consider the following: we have the install date 2004-01-01 00:00:00, an aggregation window width of 30 minutes and page size of 240. So, as can be easily calculated, on one page we have aggregates from five days (48 aggregates from one day, $\frac{240}{48} = 5$). Next we create an iterator with the starting date 2004-01-15 13:03:45. The calculated border date will be 2004-01-11 00:00:00, starting index 218 and the first returned aggregator will have the timestamp 2004-01-15 13:30:00 and will contain the medium consumption between 13:00:00 and 13:30:00.

3.1 Page-Filling Algorithms

As a new iterator is constructed some of its table pages are filled with aggregators. Which pages and how many of them depends on the used page-filling algorithm. Each page is filled by a separate thread. The threads operate according to the following steps:

1. Check whether some other thread filling a page with an identical border date is currently running. If yes, register in the set of waiting threads and wait.
2. Get a database connection from the connection pool (see description below).
3. Check if the required aggregates were previously calculated and materialized. If yes, restore the data and go to 5.

4. Fill the page with aggregates. Materialize the page.
5. Release the database connection and browse the set of waiting threads for threads with the specified border date. Transfer the data and notify them.

In the subsections below we present three different page-filling algorithms used for retrieving aggregates from the database.

Algorithm SPARE. Two first pages of the table are filled when a new iterator is being created and the SPARE algorithm is used as a page-filling algorithm. Then, during the list browsing, the algorithm checks in the $next()$ function if the current page (let's mark it n) is exhausted. If the last aggregator from the n page was retrieved, the algorithm calls the page-filling function to fill the $n+2$ page while the main thread retrieves the aggregates from the $n+1$ page. One page is always kept as a "reserve", being a spare page.

This algorithm brings almost no overhead – only one page is filled in advance. If the page size is set appropriately so that the page-filling and page-consuming times are similar, the usage of this algorithm should result in fluent and efficient list browsing.

Algorithm RENEW. When the RENEW algorithm is used, all the pages are filled during creation of the new iterator. Then, as the aggregates are retrieved from the page, the algorithm checks if the retrieved aggregator is the last from the current page (let's mark it n). If the condition is true, the algorithm calls the page-filling function to refill the n page while the main thread explores the $n+1$ page. Each time a page is exhausted it is refilled (renewed) immediately.

One may want to use this algorithm when the page consuming time is very short (for instance the aggregators are used only for drawing a chart) and the list browsing should be fast. On the other hand, all the pages are kept valid all the time, so there is a significant overhead; if the user wants to browse the aggregates from a short time period but the MAL is configured so that the iterators have many big pages – all the pages are filled but the user does not use all of the created aggregates.

Algorithm TRIGG. During new iterator creation by means of the TRIGG algorithm, only the first page is filled. When during n page browsing the one before last aggregator is retrieved from the page the TRIGG algorithm calls the page-filling function to fill the $n+1$ page. No pages are filled in advance. Retrieving the next to last aggregator from the n page triggers filling the $n+1$ page. The usage of this algorithm brings no overhead. Only the necessary pages are filled. But if the page consumption time is short the list-browsing thread may be frequently stopped because the required page is not completely filled.

3.2 MAL in Indexing Structure

The idea of the page-filling algorithm in the MAL running as a component of a higher-level node is the same as in the TRIGG algorithm for the database iterator. The difference is in aggregates creating because the aggregates are created

using aggregates of lower-level nodes. The creation process can be divided into two phases. In the first phase the aggregates of the lower-level nodes are created. This operation consists of creating the aggregates and materialization. The aggregates in the list of the higher-level node are created through merging the aggregates of the lower-level nodes in the second phase. The second phase uses the materialized data created in the first phase and its execution takes less than 10% of the whole time required for creating the aggregates of the higher-level node.

3.3 Resource Pools

In our solution we use a concept of a resource pool. We use two pools: database connection pool and MAL iterator table pool. The first pool stores available database connections and its idea is well-known from J2EE applications. The second pool is configured to store MAL iterator tables. Thanks to such approach we are able to easily control the amount of memory consumed by the system (configuring the pool we decide how many tables it will contain at maximum) and the number of concurrently running threads (if no table is available in the pool a new iterator will not start its page-filling threads until some other iterator returns a table to the pool). In figure 1 we present a simple schema of resource pool operation.

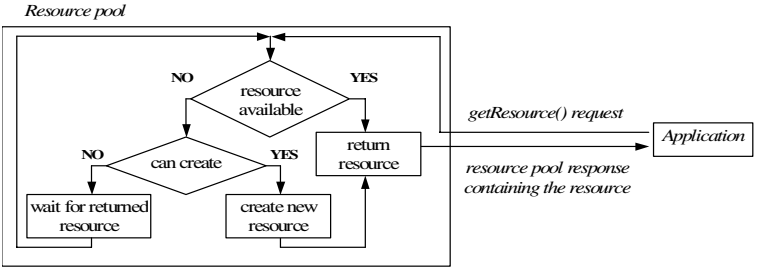


Fig. 1. Schema of a resource pool operation

3.4 Materialization

In the presented operation of the page-filling function, points (3) and (4) mention a concept of materialization. We introduced the materialization mechanism in the DSDW system presented in [3] and the tests revealed the mechanism extreme efficiency. The idea is to store once calculated aggregators as binary data in the database, using the BLOB table column. In the current approach we use a table with three columns storing the following values: the object identifier (telemetric object or indexing structure node), page border date and aggregators in binary form. The page materialization mechanism operates identically for each page-filling algorithm. The MAL can automatically process new data added by the extraction process. If some page was materialized but it is not complete, then the page-filling thread starts retrieving aggregates from the point where the data was not available.

4 Test Results

This section contains a description of the tests performed with the current implementation of the presented solution. All the tests were executed on a machine equipped with Pentium IV 2.8 GHz and 512 MB RAM. The software environment was Windows XP Professional, Java Sun 1.5 and Oracle 9i. The tests were performed for the three page-filling algorithms for the database iterator and for the index iterator. The aggregates were created for 3, 6, 9, and 12 months. The aggregates were created with a time window of 30 minutes. The created aggregates were not used in the test program; the program only sequentially browsed the list. Aggregates browsing was performed twice: during the first run the list has no access to the materialized data, and during the second run a full set of materialized data was available. We tested the MAL for the following parameters:

- page size: 48 (1 day), 240 (5 days), 336 (7 days), 672 (14 days), 1008 (21 days), 1488 (31 days – 1 month), 2976 (62 days – 2 months),
- page number: $2 \div 10$,

We set no limit to the number of database connections. The number of tables available in the table pool was limited to 1 because increasing this number brought no benefit. Our goal was to find the best combination of the MAL parameters: the page-filling algorithm, number of pages and size of a single page. The choice criterion consisted of two aspects: the efficiency measured as a time of completing the list-browsing task and memory complexity (amount of the memory consumed by the iterator table).

4.1 Page Size and Page Number

We first analyze the results of completing the list-browsing task during the first run (no materialized data available) focusing on the influence of the page number and the size of a single page. We investigated the relations between these parameters for all three algorithms, and we can state that in all the cases their influence is very similar; graphs of the relations are very convergent.

The list browsing times for small pages are very diverse. For the presented results the times for a page of size 48 vary from 30 to 160 seconds depending on the amount of pages. MAL operation for a page of size 240 is much more stable; the differences resulting from the different number of pages do not exceed 25 seconds. For all tested cases we can notice that the page number does not strongly influence MAL time efficiency. The bigger page size the smaller the page number influence. Hence we can conclude that the best MAL configuration concerning the page number and size is a combination of a small number of pages $4 \div 6$ and page size, with which the list operates stably. For the presented case the best choice seems be the page size of 672.

4.2 Page-Filling Algorithm

After choosing the optimal page parameters, we compared the time efficiency of the page-filling algorithms when applied in a theoretical indexing structure.

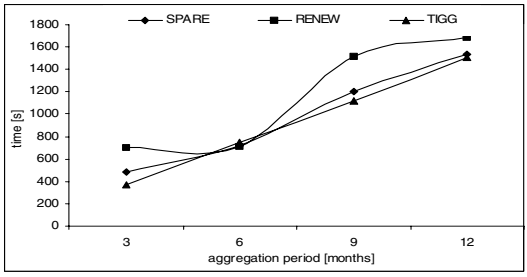


Fig. 2. Comparison of the page-filling algorithms

The structure consisted of one parent node and $10 \div 20$ child nodes; the query concerned parent aggregates but obviously resulted in creating the aggregates of all the child nodes. Figure 2 shows a graph comparing efficiency of the algorithms applied in the child nodes for browsing the list of aggregates for 3, 6, 8 and 12 months. The lists were configured to use 6 pages, each of size 672. In the graph we observe that SPARE and TRIGG algorithms show similar efficiency. Along with extending the aggregation period the operation time increases; for the TRIGG algorithm the increase is purely linear. The RENEW algorithm shows worse efficiency, especially for long aggregation periods of 6 and 12 months. Reason for that is the significantly higher overhead when compared to the other two algorithms.

Therefore, to summarize the parameters selection we can state that the MAL works efficiently for the following configuration: the TRIGG or SPARE algorithm, number of pages $4 \div 6$, size of a single page 672, with no limit to the number of available database connections and with one iterator table available in the pool.

4.3 Materialization

The aspect last investigated was materialization influence on system efficiency. The results interpretation reveals that materialization strongly improves system efficiency. In every tested combination of page size and page number parameters the benefit of materialization is very similar, and upon analyzing the results, we state that using the materialized data the list operates from 5 to 8 times faster than when no materialized is used. A similar situation is observed for all the page-filling algorithms.

5 Conclusions

In this paper we presented the Materialized Aggregate List (MAL). The MAL is a data structure for storing long aggregate lists. The list can be applied as a component of indexing structure nodes in indexes like an aR-Tree. The aggregators stored in the list can be retrieved from both the database and from other levels of an indexing structure. In our solution we applied the idea of aggregates

materialization. The materialization has a very strong, positive influence on list efficiency.

The data warehouse structure described in [3] applies distributed processing. We suppose that in this aspect introducing the MAL to our system will bring benefits in efficiency. The current approach to sending complete aggregate lists as a partial result from a server to a client results in high, single client module load. When we divide the server response into MAL pages, the data transfer and the overall system operation will presumably be more fluent. Implementation and testing of those theoretical assumptions are our future plans.

References

1. Baralis E., Paraboschi S., Teniente E.: Materialized view selection in multidimensional database. In Proc. 23^{th} VLDB, pages 156-165, Athens, 1997.
2. Golfarelli M., Rizzi S., Saltarelli E.: Index selection for data warehousing. In. Proc. DMDW, Toronto, 2002.
3. Gorawski M., Malczok R.: Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree. 5^{th} STDBM_VLDB'04 Workshop, Toronto 2004
4. Gupta H.: Selection of views to materialize in a data warehouse. In. Proc. ICDT, pages 98-112, 1997.
5. Labio W.J., Quass D., Adelberg B.: Physical database design for data warehouses. In. Proc. ICDE, pages 277-288, 1997.
6. Papadias D., Kalnis P., Zhang J., Tao Y.: Effcient OLAP Operations in Spatial Data Warehouses. Spinger Verlag, LNCS 2001
7. Rizzi S., Saltarelli E.: View Materialization vs. Indexing: Balancing Space Constraints in Data Warehouse Design, CAISE, Austria 2003
8. Theodoratos D., Bouzehoub M.: A general framework for the view selection problem for data warehouse design and evolution. In. Proc. DOLAP, McLean, 2000

A New Algorithm for Generation of Exactly M–Block Set Partitions in Associative Model

Zbigniew Kokosiński

Faculty of Electrical & Computer Eng., Cracow University of Technology,
ul. Warszawska 24, 31-155 Kraków, Poland
zk@pk.edu.pl

Abstract. In this paper a new parallel algorithm is presented for generation of all exactly m–block partitions of n–element set. The basic building blocks of the algorithm are an associative generator of combinations and a complex parallel counter. Consecutive objects are generated in lexicographic order, with O(1) time per object. The algorithm can be used for generation of all partitions within the given range of the parameter m, where $1 \leq m_1 \leq m \leq m_2 \leq n$.

1 Introduction

The first known algorithm for generating (n,m)–partitions, $1 \leq m \leq n$, *i.e.* partitions of n–element set into at least n non–empty blocks — published in 1963 — is due to Hutchinson [7]. In the following years a number of sequential algorithms was developed [3, 5, 10, 21]. An increasing interest in parallel computation systems resulted also in development of many parallel algorithms for generation of combinatorial objects. Parallel solutions to the partition generation problem were published for various models of computations [4, 12, 13, 14, 16, 22]. The structure of the set of partitions was investigated and new ranking/unranking techniques were developed satisfying various requirements [24].

In associative generator of exactly m–block partitions of n–element set described in this paper consecutive objects are generated in lexicographic order, with O(1) time per object. The basic building blocks of the algorithm are an associative generator of combinations and a specialized complex parallel counter.

The associative hardware generator of combinations and combinations with repetitions is described in [15], where consecutive objects are generated in lexicographic order, with O(1) time per object, in two different representations. Complex parallel counter with programmable capacity and some additional features provides generation of the assigned code in constant time.

The rest of the paper is organized as follows. The next section introduces combinatorial objects representations. Section 3 describes models of computations used throughout this paper. In the next section a combination generation algorithm is shown. An associative algorithm for generation of exactly m–block partitions is presented in section 5. Section 6 contains concluding remarks.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 67–74, 2006.

2 Representation of Combinatorial Objects

Let us introduce basic notions used throughout this paper.

The index set will be denoted by I. If $I = 1, \ldots, v$ it will be denoted by I_v.

Let $< A_i >_{i \in I_v}$ denote an indexed family of sets $A_i = A$, where: $A = \{1, \ldots, n\}$, $1 \leq v, n$. Any mapping f which "chooses" one element from each set $A_1, \ldots, A_v$ is called a *choice function* of the family $< A_i >_{i \in I_v}$ [20].

Any mapping f which "chooses" one element from a subset $A' \subset \{A_1, \ldots, A_v\}$ is called a *partial choice function* of the family $< A'_i >_{i \in I}$, $I \subset I_v$.

With additional restrictions we can model by choice functions various classes of combinatorial objects [8, 9, 11, 12, 13].

If a suplementary conditions: $a_i < a_j$, for i < j, and i, j $\in I_k$, k≤n, are satisfied then any choice function $\kappa =< a_i >_{i \in I_k}$, that belongs to the indexed family $< A_i >_{i \in I_n}$, is called *increasing choice function* of this family. Set of all increasing choice functions κ is a representation of the set of all (n,k)-combinations of the set A. In the conventional representation of combinations we deal in fact with indexed sets $C_i = \{i, \ldots, n\text{-}k+i\} \subset A_i$, $i \in I_k$. The number of all choice functions κ is $\binom{n}{k}$.

If a suplementary conditions: $a_i \leq a_j$, for i < j, and i, j $\in I_k$, are satisfied then any choice function $\lambda =< a_i >_{i \in I_k}$, that belongs to the indexed family $< A_i >_{i \in I_k}$, is called *nondecreasing choice function* of this family. Set of all nondecreasing choice functions λ is a representation of the set of all (n,k)-combinations with repetitions of the set A. In the conventional representation of combinations with repetitions we deal in fact with indexed sets $D_i = \{1, \ldots, n\text{-}k+1\} \subset A_i$. The number of all choice functions λ is $\binom{n+k-1}{k}$.

Parallel algorithm COMBGEN for generation of choice functions λ and κ with O(1) time per object presented in section 4 is described in [15].

If a suplementary conditions: $a_1 = 1$ and $a_i \in \{1, ..., max[a_1, \ldots, a_{i-1}] + 1\}$, for $2 \leq i \leq n$, and $i \in I_n$, is satisfied then any choice function $\rho =< a_i >_{i \in I_n}$ that belongs to the indexed family $< A_i >_{i \in I_n}$, is called a *partitioning choice function* of this family. Sets of all partitioning choice functions are representations of the set of all m-block partitions of the set A, $1 \leq m \leq n$. In the Hutchinson's representation of partitions we deal in fact with indexed sets $R_i = \{1, \ldots, i\} \subset A_i$. The number of all choice functions ρ is the Bell number B(n). Parallel algorithm PARTGEN for generation of choice functions ρ with O(1) time per object is described in [14].

Let $\kappa_e =< a_i >_{i \in I_k}$, $k \leq n$, be an increasing choice function of the indexed family $< C_i >_{i \in I_n}$ with $a_1 = 1$. The number of all choice functions κ_e is $\binom{(n-1)}{(k-1)}$.

Any choice function $\rho_e =< a_i >_{i \in I_n}$ that belongs to indexed family $< R_i >_{i \in I_n} = < R_{a_1}, .., R_{i1}, .., R_{a_2}, .., R_{i2}, .., R_{i(k-1)}, .., R_{a_k}, .., R_{ik}, .., R_n >$, where:

$R_{a_i} = i$, for $i \in I_k$,
$R_{i1} = \{1\}$, for $a_1 < i1 < a2$,
$R_{i2} = \{1, 2\}$, for $a_2 < i2 < a3$,
...

$R_{i(k-1)} = \{1, \ldots, k-1\}$, for $a_{k-1} < i(k-1) < a_k$,
$R_{ik} = \{1, \ldots, k\}$, for $a_k < ik < n$,
is called an*exactly k-block partitioning choice function* of this family.

Partial choice functions ρ_{ij} are defined for indexed families $< R_{a_j+1}, \ldots, R_{a_{j+1}-1} >$, if $1 \leq j \leq k-1$, and $< R_{a_j+1}, \ldots, R_n >$, if $j = k$.

Any concatenation of partial choice functions ρ_{ij}, $1 \leq j \leq k$, is called a partial choice function α_e.

The number of choice functions ρ_e for the given choice function κ_e matches the number of all its partial choice function α_e, i.e. $\prod_{i=1}^{k} i^{(a_{i+1}-a_i-1)}$, assuming that $a_{k+1} = n+1$.

The lexicographically ordered set of all choice functions ρ_e for all lexicographically ordered choice functions κ_e correspond one–to–one to the lexicographically ordered set of all choice functions ρ. The number of all choice functions ρ_e for all choice functions κ_e is the Stirling number S(n,k) and corresponds to the number of all exactly k–block partitions of n–element set.

Let us introduce now the lexicographic order on the set of all choice functions of the family $< A_i >_{i \in I}$.

For given choice functions $\delta =< d_1, \ldots, d_k >$ and $\gamma =< g_1, \ldots, g_k >$, we say that δ is less then γ according to the increasing lexicographic order, if and only if there exists $i \in \{1, \ldots, k\}$, satisfying $d_i < g_i$, and $d_j = g_j$, for every j < i.

3 Model of Computations

The basic model of computations of choice functions κ_e consists of single memory cell S and associative memory block A of size n, with memory cells lineary ordered, and containing respective order numbers as their address keys. Cell S and cells in block A are considered to be elementary processors. As most parallel algorithms, the generation algorithms presented in this paper require an interprocessor communication pattern. In particular, we need a single source processor (cell S) to sent identical data to a subset of destination processors (cells) in the block A. This sort of communication is called one–to–subset broadcast.

All processors of the block A and processor S are connected to a bus which is used for data transmission. In order to perform one–to–subset broadcast operation (i.e. to determine destination processors) the memory cells have to execute

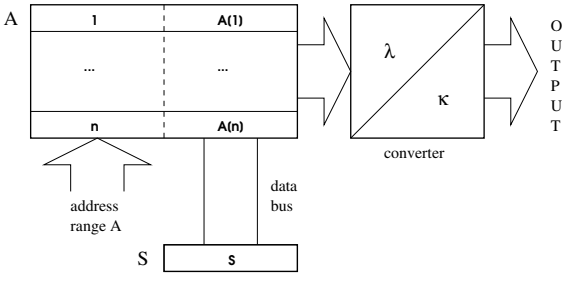


Fig. 1. Basic model of associative computations

associative range matching in processor block A. Only one one–to–subset broadcast is performed at a time.

Generation of choice functions α_e is performed in a complex parallel counter model that is built of n parallel counters with variable capacity. Each choice function κ_e computed in the associative model is used for setting exactly k counter positions while all corresponding choice functions α_e are generated on the remaining (n-k) positions (positions of κ_e are skipped by counter carries). Due to space limitations other details of the model will be omitted here.

The algorithms for parallel generation of κ_e and ρ_e sequences in lexicographic order are presented in sections 4 and 5.

4 The Combination Generation Algorithm

Construction of the presented algorithm is based on the observation that despite of applying different approaches to the generation task various generation algorithms for given class of objects reveal a common control structure.

In this paper we assume that the control structure of (n,k)–combinations is the structure of (n-k+1,k)–combinations with repetitions. The properties of the sequence of combinations with repetitions as nondecreasing choice functions λ are a key factor of our parallelization method. Therefore the sequence of choice functions λ has been chosen as a basic control sequence for the generation.

The presented algorithm uses consequently a uniform one-to-subset broadcast operations described in section 3. In order to produce control sequences the algorithm operates on the set of associative memory locations A and single memory cell S. The range of the subset of destination cells in set A is determined in parallel by associative *compare range* operation which requires O(1) time. In the algorithm COMBGEN the procedure OUTPUT performs a conversion which is required to produce from the control sequence in table A the output sequence. A pseudocode of the parallel algorithm COMBGEN for generating combinations is presented in Fig.2.

Exemplary sequences generated by the algorithm, for n=6, k=3, are depicted in Table 1. In tables S and A transformations of the control sequence are shown. The **bold** font in these columns points out the source and the destination memory cells in all one-to-subset broadcasts between S and A. Exemplary output sequences λ and κ are shown in tables A and K, respectively.

Table 1. Sequences generated by algorithms COMBGEN (n=6, k=3)

No.	IND	S	A=L	K	No.	IND	S	A	K	No.	IND	S	A	K
1	1	**1**	**1 1 1**	1 2 3	8	2	**3**	1 **3 3**	1 4 5	15	3	**4**	2 3 **4**	2 4 6
2	3	**2**	1 1 **2**	1 2 4	9	3	**4**	1 3 **4**	1 4 6	16	2	**4**	2 **4 4**	2 5 6
3	3	**3**	1 1 **3**	1 2 5	10	2	**4**	1 **4 4**	1 5 6	17	1	**3**	**3 3 3**	3 4 5
4	3	**4**	1 1 **4**	1 2 6	11	1	**2**	**2 2 2**	2 3 4	18	3	**4**	3 3 **4**	3 4 6
5	2	**2**	1 **2 2**	1 3 4	12	3	**3**	2 2 **3**	2 3 5	19	2	**4**	3 **4 4**	3 5 6
6	3	**3**	1 2 **3**	1 3 5	13	3	**4**	2 2 **4**	2 3 6	20	1	**4**	**4 4 4**	4 5 6
7	3	**4**	1 2 **4**	1 3 6	14	2	**3**	2 **3 3**	2 4 5		0			

Input : n – size of the set, k – size of the subsets.
Output: Table K with the consecutive choice functions κ.
Method: In table S future values of A subsequences are computed and stored in advance. Computations begin with S=1. Then, the first function λ in the table A is obtained (steps 1-2), and next value of S is determined (step 3). In step 4 the first output is produced. Next, consecutive values A and S are produced and output sequences are computed (step 5). Computations run until the last c.f. is generated, *i.e.* IND=0.

```
/1-3 initialization phase/
1. MAX:=n-k+1; IND:=1; S:=1;
2. ONE2SUBSET(S,A,IND,k);
3. S:= A[IND]+1;
4. do in parallel
   4.1. OUTPUT;
   4.2. IND:=k;
5. while IND>0 do
   5.1. ONE2SUBSET(S,A,IND,k);
   5.2. do in parallel
     5.2.1. OUTPUT;
     5.2.2. if A[IND]<MAX then
       5.2.2.1. S:= A[IND]+1;
       5.2.2.2. IND:=k;
       else
       5.2.2.3. IND:=IND-1;
       5.2.2.4. S:=A[IND]+1;

ONE2SUBSET(ONE,SET,LEFT,RIGHT)
/one-to-subset broadcast/
1. for I:=LEFT to RIGHT do in parallel
   SET[I]:=ONE;

OUTPUT /conversion and output/
case of object
K: for I:=1 to k do in parallel K[I]:=A[I]+I-1;
output K;
```

Fig. 2. The algorithm COMBGEN

A constant delay between objects is provided by execution of conditional step 5.2.2 in constant time. If constant delay is not essential further speedup may be achieved through a hardware implemention if, for IND=k, all one-to-one broadcasts are replaced by increment operations.

Theorem 1. *Algorithm COMBGEN generates, in the conventional representation, all (n,k)–combinations, $1 \leq k \leq n$, in the lexicographic order with constant time per combination in an associative model with at most n+1 processors, each of constant size. Thus, the algorithm COMBGEN is optimal.*

5 The Partition Generation Algorithm

In this paper we assume that the control structure for (n,m)–partitions consists of the structure of the set of (n,k)–combinations with $a_1 = 1$ whose elements represent positions of initial elements of m=k partition blocks in Hutchinson representation and, in general, the structure of the set of choice functions representing elements of the remaining partition blocks. The pseudocode of the parallel algorithm M-PARTGEN for generating exactly m–block set partitions in the Hutchinson's representation is shown in Fig.3.

Exemplary sequences generated by the algorithm M-PARTGEN, for n=6, m=3, are depicted in Table 2.

Input : n — size of the set, m — the number of partition blocks.
Output: Table R with the consecutive choice functions ρ_e.
Method: For each c.f. κ_e generated by algorithm COMBGEN in lexicographic order compute all partial choice functions α_e and generate in table R all partitioning choice functions ρ_e in Hutchinson representation and in lexicographic order.

/1-2 initialization phase/
1. k:=m;
2. R[1]:=1;
/3 generation of all choice functions ρ_e/
3. **for** i:= 1 **to** $\binom{n-1}{k-1}$ **do**
   3.1. **for** j:= 2 **to** n **do in parallel** R[j]:= 1;
   3.2. generate next κ_e using COMBGEN(n-1,k-1,K);
   3.3. **for** j:= 1 **to** (k-1) **do in parallel** R[K[j]+1]:= j+1;
   3.4. **repeat**
      3.4.1. generate the next partial choice function α_e on the (n–k) positions of the table R in the complex parallel counter model;
      3.4.2. output table R with the next choice function ρ_e
   until all partial choice functions α_e for the given κ_e are generated;

Fig. 3. The algorithm M-PARTGEN

Table 2. Sequences generated by algorithm M-PARTGEN (n=6, m=3)

No.	R	No.	R	No.	R
1	1 1 1 1 2 3	11	1 1 2 1 3 1	34	1 2 1 1 3 1
2	1 1 1 2 1 3	...	...	...	...
3	1 1 1 2 2 3	16	1 1 2 2 3 3	45	1 2 2 2 3 3
4	1 1 1 2 3 1	17	1 1 2 3 1 1	46	1 2 1 3 1 1
...	...	...	...	...	...
6	1 1 1 2 3 3	25	1 1 2 3 3 3	63	1 2 2 3 3 3
7	1 1 2 1 1 3	26	1 2 1 1 1 3	64	1 2 3 1 1 1
...	...	...	...	...	...
10	1 1 2 2 2 3	33	1 2 2 2 2 3	90	1 2 3 3 3 3

Theorem 2. *Algorithm M-PARTGEN generates, in Hutchinson's representation, all exactly m–block partitions of n-element set in the lexicographic order with O(1) time per partition in an associative model extended by complex parallel counter model, both with O(n) hardware complexity.*

6 Concluding Remarks

A new O(1) algorithm for the generation of exactly m–block partitions has been described in this paper. The algorithm can be used for generation of all partitions within the given range of the parameter m, where $1 \leq m_1 \leq m \leq m_2 \leq n$.

The algorithm provides the parallelization of computations at the level of single combinatorial object. However, it can be used in adaptive partition generation too, enabling further parallelization on the set of objects level. In this case standard unranking techniques for partitions may be applied with a little effort for programming a number of generators working in parallel [24]. Splitting the generation task in adaptive generation algorithm is much easier to accomplish in associative model then in linear array model, since in the latest model it is necessary to know states of all registers involved in computations and message passing in each of n systolic processors in order to program one partition generator [2, 22].

References

1. Akl S.G., Gries D., Stojmenović I.: An optimal parallel algorithm for generating combinations. Information Processing Letters 33 (1989/90) 135–139
2. Akl S.G., Stojmenović I.: Generating combinatorial objects on a linear array of processors. [in:] Zomaya A.Y. (editor): Parallel Computing. Paradigms and Applications, Int. Thompson Comp. Press (1996)639–670
3. Djokić B. *et al.*: A fast iterative algorithm for generating set partitions. The Computer Journal 32 (1989) 281–282
4. Djokić B. *et al.*: Parallel algorithms for generating subset and set partitions. Proc. SIGAL Int. Symposium on Algorithms, Tokyo, Japan (1990)
5. Er M.C.: A fast algorithm for generating set partitions. The Computer Journal 31 (1988) 283–284
6. Even S.: Algorithmic Combinatorics. Macmillan, New York (1973)
7. Hutchinson G.: Partitioning algorithms for finite sets. Comm. ACM 6 (1963) 613–614
8. Kapralski A.: New methods for generation permutations, combinations and other combinatorial objects in parallel. J. Parallel and Distrib. Computing 17 (1993) 315–326.
9. Kapralski A.: Modeling arbitrary sets of combinatorial objects and their sequential and parallel generation. Studia Informatica 21, No.2 (40) (2000)
10. Kaye R.: A Gray code for set partitions. Inform. Process. Letters 5 (1976) 171–173
11. Kokosiński Z.: On generation of permutations through decomposition of symmetric groups into cosets. BIT 30 (1990) 583–591
12. Kokosiński Z.: Circuits generating combinatorial configurations for sequential and parallel computer systems. Monografia 160, Politechnika Krakowska, Kraków, Poland (1993) (in Polish)

13. Kokosiński Z.: Mask and pattern generation for associative supercomputing. Proc. IASTED Int. Conference AI'94, Annecy, France (1994) 324–326
14. Kokosiński Z.: On parallel generation of set partitions in associative processor architectures. Proc. Int. Conf. PDPTA'99, Las Vegas, USA (1999) 1257–1262
15. Kokosiński Z.: On parallel generation of combinations in associative processor architectures. Proc. IASTED Int. Conf. Euro–PDS'97, Barcelona, Spain (1997) 283–289
16. Lee W.-T., Tsay J.-C., Chen H.-S., Tseng T.-J.: An optimal systolic algorithm for the set partitioning problem. Parallel Algorithm and Applications 10 (1997) 301–314
17. Lehmer D.H.: Teaching combinatorial tricks to a computer. Proc. of Symposium Appl. Math., [in:] Combinatorial Analysis 10, Amer. Math. Society, Providence, R.I. (1960) 179–193
18. Lehmer D.H.: The machine tools of combinatorics, [in:] Beckenbach E.F. (editor): Applied combinatorial mathematics. John Wiley, N.Y. (1964) 5–31
19. Lin C.J., Tsay J.C.: A systolic generation of combinations. BIT 29 (1989) 23–36
20. Mirsky L.: Transversal theory. Academic Press, N.Y. (1971)
21. Semba I.: An efficient algorithm for generating all partitions of the set {1,2,...,n}. Journal of Information Processing, 7 (1984) 41–42
22. Stojmenović I.: An optimal algorithm for generating equivalence relations on a linear array of processors. BIT 30 (1990) 424–436
23. von zur Gathen J.: Parallel linear algebra. [in:] Reiff J.H. (editor): Synthesis of parallel algorithms, Morgan Kaufman (1993) 573–617
24. Williamson S.G.: Ranking algorithms for lists of partitions. SIAM Journal of Computing 5 (1976) 602–617

A Self-stabilizing Algorithm for Finding a Spanning Tree in a Polynomial Number of Moves

Adrian Kosowski and Łukasz Kuszner

Department of Algorithms and System Modeling,
Gdańsk University of Technology, Poland
{kosowski, kuszner}@sphere.pl

Abstract. In the self-stabilizing model each node has only local information about the system. Regardless of the initial state, the system must achieve a desirable global state. We discuss the construction of a solution to the spanning tree problem in this model. To our knowledge we give the first self-stabilizing algorithm working in a polynomial number of moves, without any fairness assumptions. Additionally we show that this approach can be applied under a distributed daemon. We briefly discuss implementation aspects of the proposed algorithm and its application in broadcast routing and in distributed computing.

1 Introduction

A distributed system consists of nodes that are pairwise connected by communication channels. Each node maintains variables which determine its *local state*. The *global state* of the system is the union of all local states. Such a model is seen to be a good abstraction for real objects such as peer-to-peer networks.

The system is constructed in such a way as to guarantee that it works correctly, i.e. persists in a legitimate state, even though some perturbations can bring it to an illegitimate state. It is desirable that it returns to a legitimate state without any external intervention. Self-stabilization, a concept introduced by Dijkstra [4] in 1974, can be thought of as a technique for designing such resilient systems. A *self-stabilizing system* is one which is able to achieve a legitimate global state starting from any possible global state.

A distributed system can be modeled by a connected graph $G = (V, E)$, where vertex set V corresponds to system nodes and the set of edges E denotes communication links between them. A vertex can change its local state by making a *move*. The algorithm for each vertex v is given as a set of rules of the form **if** $p(v)$ **then** A, where $p(v)$ is a predicate over local states of v and its neighbors, and A is an action changing a local state of v (a move of v). A vertex v becomes *active* when $p(v)$ is true, otherwise v is *stable*. The execution of the algorithm is controlled by a *scheduler* which allows some non-empty subset of active vertices to perform a simultaneous move defined by the rules for the respective nodes; this is referred to as a single *action*. If all vertices in a graph are stable, we

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 75–82, 2006.

say that the system is *stable*. In order to measure the time complexity of self-stabilizing algorithms we often use the number of moves or, more commonly in synchronized systems, the number of rounds.

Finding a spanning tree in such a system can be the basis of many complex distributed protocols like broadcasting, token circulation or code assignment. Predictably, many publications on the subject have appeared in recent years. A self-stabilizing algorithm for a BFS spanning tree in a semi-uniform system with a central daemon under read/write atomicity was described in [6]. Afek, Kutten and Yung in [1] gave a similar algorithm but for a uniform network, which stabilizes in $O(n^2)$ asynchronous rounds. Another algorithm was given by Arora and Gouda in [3] as a part of a more complex algorithm. In this paper authors assumed unique identifiers for vertices and a bound on the graph size known to all nodes. A very simple algorithm was presented by Huang and Chan in [8]. Yet another, which also needs a bound on n known to all vertices, was invented by Sur and Srimani in [10].

Later on, many other papers appeared on the subject, describing various approaches, considering time effectiveness [2], memory requirements [5, 9] or communication costs. But, to the best of our knowledge, no algorithm working in a polynomial number of moves without any assumptions on scheduler fairness has ever before been described in literature, even for the case when the scheduler selects exactly one of the active nodes at a time to make a move. We give a general solution to the considered problem for arbitrary schedulers.

2 An Algorithm for Finding a Spanning Tree in $O(n \operatorname{diam}(G))$ Moves

Let $G = (V, E)$ be a system graph, where vertex set V corresponds to system nodes and the set of edges E denotes communication links between them. By $n = |V|$ and $m = |E|$ we denote the number of vertices and the number of edges, respectively. In addition, let $N(v) = \{u : (u, v) \in E\}$ be the *open neighborhood* of v, and let $\deg(v) = |N(v)|$ be the *degree* of v. A spanning tree $T = (V, E')$ of $G = (V, E)$ is a subgraph of G consisting of the same set of nodes V, but only a subset $E' \subseteq E$ of edges such that there exists exactly one path between every pair of nodes in T. To ensure the existence of a spanning tree, graph G must be connected, so in this paper we restrict our considerations to connected graphs only.

In our first approach we provide a simple semi-uniform algorithm, in which each node has only one local variable f, a non-negative integer. Semi-uniformity means that exactly one of the nodes, called a *root*, needs to be distinguished. We will denote it by r. The interpretation of state variable f is as follows. Consider node v and let us choose u such that $f(u) = \min_{w \in N(v)} f(w)$ and u is the first[1]

[1] To be able to say "first" we must assume that the neighbors are somehow ordered. This is not a strong assumption as long as a node is able to distinguish between its neighbors. For example, if the neighbors of v are stored in the form of a list, the order can be given according to the list sequence.

vertex among the neighbors of v with such a property. If $f(u) < f(v)$ then we say that u is the *parent* of v. For the distinguished vertex $f(r)$ is permanently equal to 0.

We now proceed to show an algorithm which will guarantee that the parent of each vertex (with the exception of the root) is uniquely defined, thus implying an ordering of the vertices equivalent to a spanning tree.

Algorithm 1:

R: **if** $v \neq r \wedge f(v) \leq \min_{u \in N(v)} f(u)$
then $f(v) = \max_{u \in N(v)} f(u) + 1$

Let $T = (V, E_T)$ be an arbitrary spanning tree of G. Consider an edge $e = \{u, v\} \in E_T$, where vertex u is closer to root r in tree T than vertex v, $d_T(u, r) < d_T(v, r)$ ($d_T(u, v)$ is the length of path $[u, v]_T$). We will call edge e *correctly directed* if $f(u) < f(v)$. When analyzing Algorithm 1 it is useful to bear in mind the following observation.

Corollary 1. *If for a given state of the system running Algorithm 1 there exists a spanning tree T such that all its edges are correctly directed, then the system is stable.*

Theorem 2. *Algorithm 1 stabilizes in $O(n \operatorname{diam}(G))$ moves.*

Proof. We now select an arbitrary spanning tree T of G and define the following function:

$$S_T(v) = \sum_{(w,w') \in E([r,v]_T)} \max\{f(w) - f(w'), 0\} \tag{1}$$

Intuitively, $S_T(v)$ can be thought of as the number of edges lying on the path $[r, v]_T$ which are correctly directed. Obviously:

$$0 \leq S_T(v) \leq d_T(r, v) \tag{2}$$

Let us consider the effect of a parallel action of a set of vertices $X = \{x_1, \ldots, x_k\} \subset V$ on the values $S_T(v)$. As the root r cannot make any move, thus $r \notin X$. Without loss of generality we can assume that the subgraph H of G induced by X is connected, since otherwise the same actions can be performed by the scheduler in several successive actions, without any time gain. The structure of Algorithm 1 implies that before the action, for all i we have $f(x_i) \leq \min_{u \in N(x)} f(u)$, and consequently $f(x_i) = f(x_j)$ for all i, j. Consider an arbitrary connected component P of the forest $T \cap H$. The change of values of f within component P affects which edges of T are correctly directed. Notice that this operation does not affect the edges of $T \setminus P$. Let e_P be the edge connecting P with the component of $T \setminus P$ containing root r. Before the action, no edge of P was correctly directed, and edge e_P was not correctly directed either. After the action, edge e_P is correctly directed. Taking into account the fact that for any vertex $v \in V$ the value $S_T(v)$ depends only on which edges of the path $[r, v]_T$ are correctly directed, we immediately obtain that the value $S_T(x)$ increases by at least 1 for

every vertex $x \in P$, and does not decrease for any other vertex of T. Since this reasoning can be repeated for all other connected components of $T \cap H$, it is evident that the value $S_T(x)$ increases by at least 1 for every vertex which is making a move, and does not decrease for any other vertex.

Thus, by inequality (2) and Corollary 1 we obtain that a vertex v may make at most $d_T(r, v)$ moves while the algorithm is stabilizing. Since the above reasoning holds for any tree T, it is also correct for the BFS spanning tree T_B of graph G rooted at vertex r. The following inequality is true for any vertex v: $d_{T_B}(r, v) \leq \mathrm{diam}(G)$. All vertices become stable after making at most $\mathrm{diam}(G)$ moves each, which completes the proof. □

Theorem 3. *Algorithm 1 finds a spanning tree.*

Proof. Suppose that the system is stable, so each node is stable. Hence according to our definition of the parenthood relation every vertex except root r has a unique parent node (otherwise, such a vertex would have a locally minimal value and rule `R` would be active for it). By including all edges $\{u, v\}$ such that v is a parent of u we obtain graph T, a subgraph of $G = (V, E)$. It is easy to observe that r is a vertex of T, thus T has n vertices and $n-1$ edges. Moreover, by definition of parenthood, T may not contain any cycles, so T is a spanning tree of G. □

The algorithm given above is extremely simple and fast. However, at this point we can observe certain inconveniences. First, we cannot give an upper bound on the value of $f(v)$ and secondly, we do not provide sufficient local state information to allow a vertex to recognize its child nodes in the tree (only its parent). The former problem is addressed in detail in Section 4. The latter may be easily solved

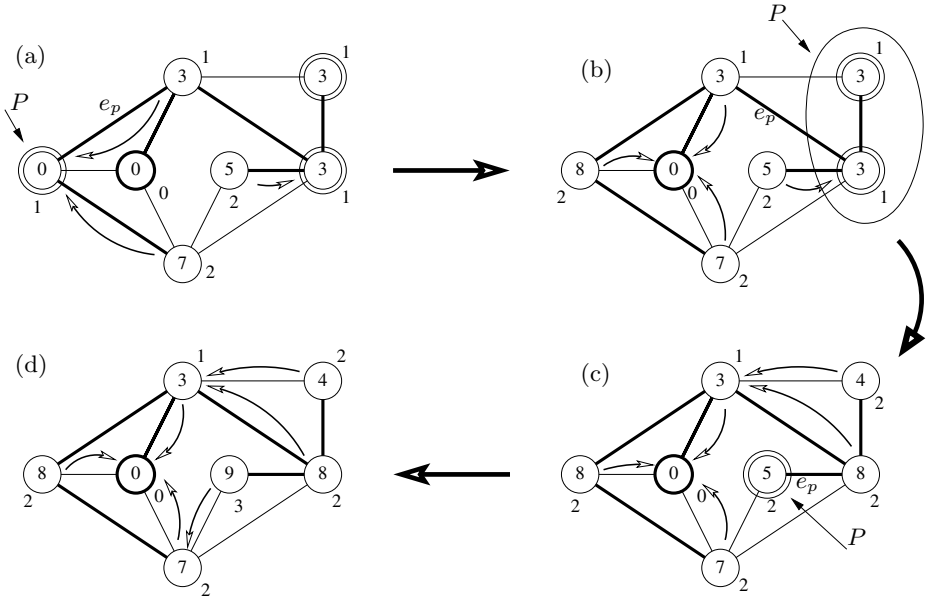


Fig. 1. Illustration of a process of Algorithm 1 (consult Subsection 2.1)

by adding a simple rule to Algorithm 1, which has no effect on its polynomial pessimistic stabilization time.

2.1 Example

In Figure 1 each of the pictures illustrates the states of the vertices in successive moves. The root vertex r is distinguished by a bold circle and active vertices are marked with double circles. The state value f of each vertex is given inside the circle, while the value S_T, used in the proof of theorem 2, is marked close to the circle. An exemplary chosen spanning tree T is denoted by bold lines, while additional arcs show the parenthood relation between vertices. A small pointer indicates the vertices to perform the next action. Symbols P and e_p have the same meaning as in the proof. The algorithm computes a spanning tree in three actions, and the result is visible (in the form of the parenthood relation) in Figure 1(d).

3 Fault Containment for Limited Perturbations

In practical applications of spanning tree construction it is often desirable for the stabilization time of the algorithm to be dependent on the severity of the faults which appeared in the system, i.e. a minor perturbation ought to result in quicker stabilization. This notion is referred to as *fault-containing* spanning tree construction, and was studied in the round-based model by Ghosh, Gupta and Pemmaraju [7]. We say that the system starts in a *k-faulty state* if the initial state can be transformed into a state representing some valid spanning tree of G (rooted at r) by changing local vertex states of not more than k vertices.

Algorithm 1 proves extremely stable when considered from the point of view of fault containment.

Property 4. *If the system starts in a k-faulty state, then Algorithm 1 stabilizes in $O(kn)$ moves.*

Proof. The proof is based on a similar method as that applied in the proof of Theorem 2. Suppose that the system starts in a k-faulty state with tree T used as the reference solution. It suffices to notice that the introduction of a single fault results in a change of values $S_T(v)$ by no more than 1 for all vertices. Consequently, in a k-faulty state the initial value of $S_T(v)$ is never smaller than the final value of $S_T(v)$ by more than k. Since every move of a vertex increases its current value $S_T(v)$ by at least 1, no vertex will ever move more than k times while Algorithm 1 is stabilizing. Of course, the solution obtained by the algorithm need not be the same as the reference tree T. □

Finally, it is interesting to observe the way in which Algorithm 1 builds its spanning tree. At every stage of execution, each vertex of the graph is capable of indicating its direct parent or stating that in the current arrangement it has no parent (at the end of the process the only vertex left without a parent is the

root r). This feature is of some importance in networking applications, since it guarantees that at every stage of execution the temporary solution is a directed spanning forest, and is always acyclic.

4 Memory Usage for Local States

The local state of Algorithm 1 consists of one local variable f and one implied pointer, indicating the parent of the vertex. The values $f(v)$ are non-negative in the initial state and increase throughout the operation of the algorithm. It is easy to show that if their initial value is polynomial with respect to n, then their final value is also polynomial with respect to n. Indeed, let F_t denote the maximum of $f(v)$ taken over all vertices, for the state of the system after exactly t actions. Initially, $F_t = F_0$ and with every action F_t increases by at most 1. Since by Theorem 2 the algorithm stabilizes after f actions, for some $f \leq n \operatorname{diam}(G)$, the final value F_f fulfills the inequality $F_f \leq F_0 + n^2$.

However, in practical applications it is often useful to give up the classical model of a local state understood as a memory cell of dynamically expandable size, and assume that the size of the local state is limited by physical storage constraints. In order to achieve this, we propose the following modification of Algorithm 1, in which the local state variable $f(v)$ is understood as a memory cell capable of storing any integer from the range $[0, N]$, for some known constant value $N \geq n$. The only exception is the local state of the root, as the value $f(r)$ is always fixed and equal to 0 (as in the case of Algorithm 1).

Algorithm 2:

R: **if** $v \neq r \wedge f(v) \leq \min_{u \in N(v)} f(u) \wedge \max_{u \in N(v)} f(u) < N$
then $f(v) = \max_{u \in N(v)} f(u) + 1$

R1: **if** $v \neq r \wedge f(v) > \max_{u \in N(v)\,:\, f(u) < f(v)} (f(u) + 1)$
then $f(v) = \max_{u \in N(v)\,:\, f(u) < f(v)} (f(u) + 1)$

Theorem 5. *Algorithm 2 stabilizes in $O(Nn \operatorname{diam}(G))$ moves and uses $O(\log N)$ storage space per vertex.*

Proof. The $O(\log N)$ storage space required by the algorithm is an obvious consequence of the structure of the local state. We will concentrate on proving the bound on the number of moves required by the algorithm.

Rule R of Algorithm 2 is a copy of rule R of Algorithm 1, with the constraint that the rule will only activate provided the resultant value $f(v)$ does not exceed the imposed upper bound N. Rule R1 reduces the value $f(v)$ for the active vertex v to the smallest possible value which does not make any correctly directed edge in graph G (with respect to any spanning tree) lose its correct direction (consult the proof of Theorem 2), even if such a rule is executed in parallel with rule R on other vertices.

It suffices to show that before the algorithm stabilizes rule R activates in at most $n \operatorname{diam}(G)$ moves, and rule R1 activates in at most $O(Nn \operatorname{diam}(G))$ moves.

The first part of the statement is true by Theorem 2, since the bound on the number of activations of rule `R` is only dependent on the correct directions of edges in some spanning tree (proof of Theorem 2), and the number of such moves made by a single vertex is bounded by $\mathrm{diam}(G)$. To prove the second part of the statement, we consider the sum Σ_t of values $f(v)$ taken over all vertices, in the state of the system after exactly t actions. The change of value Σ_t from Σ_0 in the first considered state to Σ_f in the last considered state is the result of the total change $\Delta\Sigma_{\mathtt{R}}$ caused by moves using rule `R` and the total change $\Delta\Sigma_{\mathtt{R1}}$ caused by activations of rule `R1`, i.e.:

$$\Sigma_f = \Sigma_0 + \Delta\Sigma_{\mathtt{R}} + \Delta\Sigma_{\mathtt{R1}} \tag{3}$$

Since at every stage of the algorithm all values $f(v)$ are bounded ($0 \leq f(v) \leq N$), we have $0 \leq \Sigma_0 \leq Nn$ and $0 \leq \Sigma_f \leq Nn$. A single move using rule `R` increases Σ_t by not more than N, and the number of activations of rule `R` is bounded by $n\,\mathrm{diam}(G)$, hence $0 \leq \Delta\Sigma_{\mathtt{R}} \leq Nn\,\mathrm{diam}(G)$. From (3) and the above observations we have:

$$|\Delta\Sigma_{\mathtt{R1}}| \leq |\Sigma_f| + |\Sigma_0| + |\Delta\Sigma_{\mathtt{R}}| \leq Nn(\mathrm{diam}(G) + 2) \in O(Nn\,\mathrm{diam}(G)) \tag{4}$$

By studying rule `R1` it is easy to observe that any activation of this rule strictly decreases the value Σ, and consequently the number of activations of this rule does not exceed $|\Delta\Sigma_{\mathtt{R1}}|$. By applying (4) we obtain the desired bound on the number of moves. □

Theorem 6. *Algorithm 2 finds a spanning tree.*

Proof. By studying the proof of Theorem 3, we observe that rule `R` is inactive for all vertices in one of two cases: (1) the current state is already stable, or (2) there is a vertex v with all edges directed towards it which has a neighbor u such that $f(u) = N$. It suffices to prove that the latter case is not a final state of Algorithm 2. The proof proceeds by contradiction. Suppose that rule `R1` is also inactive for all vertices. This means that for any vertex w we either have $f(w) = 0$, or there exists a neighbor x of w such that $f(w) = f(x) + 1$. Consequently, for any vertex w the inequality $f(w) < n$ holds, a contradiction with $f(u) = N \geq n$. □

It is interesting to study the effect of the chosen bound N on the performance of Algorithm 2. If N is polynomial with respect to n, $N \in O(\mathrm{poly}(n))$, then the memory required for storing a local state is $O(\log n)$, and the number of moves made by Algorithm 2 is $O(\mathrm{poly}(n))$. In particular, if N is a constant-factor bound on the value of n, i.e. $N \in O(n)$, then the number of moves of Algorithm 2 may be written as $O(n^2\,\mathrm{diam}(G))$. On the other hand, it has to be remembered that the value N needs to be stored in all vertices, and consequently selecting a value too close to n may decrease the scalability of the system.

5 Final Remarks

When considering the model of self-stabilization without fairness guarantees, the spanning tree algorithm presented in this paper has numerous and profound implications.

First of all, the presented algorithm may be easily and efficiently applied in a broadcasting protocol for a distributed network. In such a network, broadcast packets may only be routed along edges of some spanning tree (to avoid infinite transmission loops) and the spanning tree may need to be dynamically recreated in case of temporary malfunction without the intervention of a central agent. A spanning tree suitable for such a protocol can be constructed using an algorithm based on Algorithm 2 with one additional rule (allowing a vertex to know not only its parent, but also its children), whose polynomial stabilization time is immediately evident.

Moreover, the existence of the discussed algorithm shows that it is possible to use an algorithm stabilizing in a polynomial number of moves to determine a structure defined by a global property in the graph, but using only local neighborhood information. Such an approach opens up new possibilities for polynomial-time self-stabilizing distributed computing without fairness assumptions when solving problems related to code assignment.

References

1. Y. Afek, S. Kutten and M. Yung, *Memory efficient self-stabilizing protocols for general networks*, Proceedings of the 4th International Workshop on Distributed Algorithms, 1991, 15–28.
2. S. Aggarwal and S. Kutten, *Time optimal self-stabilizing spanning tree algorithms*, LNCS **761** (1993), 400–410.
3. A. Arora and M. Gouda, *Distributed reset*, IEEE Transactions on Computers, **43** (1994), 1026–1038.
4. E. W. Dijkstra, *Self-stabilizing systems in spite of distributed control*, Communication of the ACM **17** (1974), 643–644.
5. S. Dolev, M. Gouda and Marco Schneider, *Memory requirements for silent stabilization.* Acta Informatica **36** (1999), 447–462
6. S. Dolev, A. Israeli and S. Moran, *Self-stabilization of dynamic system assuming only read/write atomicity*, Distributed Computing **7** (1993), 3–16.
7. S. Ghosh, A. Gupta, and S. Pemmaraju, *A fault-containing self-stabilizing algorithm for spanning trees*, Journal of Computing and Information **2** (1996), 322-338.
8. S. Huang and N. Chen, *A self-stabilizing algorithm for constructing breadth-first trees*, Inform. Process. Lett. **41** (1992), 109–117.
9. C. Johnen, *Memory Efficient, Self-Stabilizing algorithm to construct BFS spanning trees*, Proc. of the third Workshop on Self-Stabilizing System (1997), 125–140,
10. S. Sur and P. K. Srimani, *A self-stabilizing distributed algorithm to construct BFS spanning trees of a symmetric graph*, Parallel Process. Lett. **2** (1992), 171–179.

Massive Concurrent Deletion of Keys in B*-Tree

S. Arash Ostadzadeh[1], M. Amir Moulavi[2], and Zeinab Zeinalpour[2]

[1] Islamic Azad University of Mashhad, Computer Engineering Department,
Ostad Yousefi St. Ghasem Abad, Mashhad, Iran
ostadzadeh@mshdiau.ac.ir
www.mshdiau.ac.ir
[2] Young Researchers Club, Islamic Azad University of Mashhad,
Mashhad, Iran
amir.moulavi@computer.org,
zeinabzt@gmail.com

Abstract. B*-tree is an improved variant of well known B-tree data structure which has extensive applications in data storage and retrieval systems including parallel database systems. In this paper, we present an algorithm for deleting keys of B*-tree concurrently in the case that the number of to-be-deleted keys is more than a half of the total keys in the tree. The proposed algorithm can be implemented on CREW PRAM model in optimal $O(\log_2 n + B\log_B n)$ time with the total processors equal to the keys to be deleted. n is the total number of keys in B*-tree and B is equal to half of the keys in an internal node containing maximum keys. It counts as an improvement upon the previous comparable known algorithms by a reduction of factor B in the $(\log_2 n)$-term of the time complexity.

1 Introduction

B*-tree is one kind of balanced search trees and an improved variant of B-tree [3] according to Knuth [10]. It has been extensively employed in standard index-based systems such as Disk Database Systems and also recent Parallel Database Systems. Although the B-tree structure is mainly concerned with storage and retrieval in secondary memory resident databases, there exists a variant which is well suited for the main memory called cache-conscious B^+-tree [7,16].

The height of B*-tree is $O(\log_U n)$ and dictionary operations i.e. search, insertion and deletion are performed in $O(U\log_U n)$ time in sequential computation model, where n equals the number of keys in B*-tree and U is the minimum degree of B*-tree.

Many sequential [1,2,4,5,13,17] parallel and concurrent [6,9,12,14,18,19,20] analysis and computational algorithms and data structures on B-tree and its variants like B*-tree have been studied. Since synchronous and asynchronous environments differ from each other, the methods and techniques developed in one of them can not be used in the other one and vice versa. Our algorithm is developed for synchronous environment but in a different case of parallel deletion compared to the previous related proposed algorithms, so that makes the

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 83–91, 2006.

thorough and accurate comparison irrelevant in the general case. The algorithm is based on rebuilding the modified B*-tree after deletion rather than rectifying the old B*-tree. Previous results in similar variants of B-trees on different models and environments can also be studied [8,14,18]. For general B-trees, Higham and Schenk [8] achieved optimal $O(\log_2 s + B\log_B n)$ time for search, optimal $O(B(\log_2 s + \log_B n))$ time for insertion and $O(B^2(\log_B s + \log_B n))$ time for deletion with s processors on the EREW PRAM, where s is the number of keys for the concurrent dictionary operations. It should be noted that the time complexity attained for the deletion algorithm is by a factor B slower than the optimality computed theoretically. Later, Heejin Park et al. [14] presented an optimal parallel deletion algorithm on B-trees that runs in $O(B(\log_2 s + \log_B n))$ time with s processors on the EREW PRAM. Both of the algorithms proposed in [8] and [14] adopt pipelining to achieve improved time complexities.

In this paper, we present a new parallel deletion algorithm for B*-tree that runs in optimal $O(\log_2 n + B\log_B n)$ time with s processors on the CREW PRAM model. n is the total number of keys in B*-tree, B is equal to half of the keys in an internal node containing maximum keys and s is equal to the number of keys to be deleted.

The rest of this paper is organized as follows. In section 2, the B*-tree definition is stated and we also describe terms and facts that are used throughout the algorithm. In section 3, we present our parallel deletion algorithm. We conclude in section 4.

2 Preliminaries

A B*-tree of maximum degree $m \geq 3$ is a rooted tree which must satisfy the following properties according to Knuth [10]. The properties are similar to B-tree with the distinction that the B*-tree will guarantee every node except for the root will be two-thirds full instead of the conventional half in B-trees. For the sake of similarity with B-tree, we have adopted parameter B in our algorithm which is equal to $\lfloor (m+1)/2 \rfloor$. The construction of such a tree will also follow two full nodes combination divided into three approximately equal-sized nodes scheme, in order to satisfy the restriction which defines the miminum number of keys in a node.

1. Every node except the root has at most m children.
2. Every node except for the root and the leaves (external nodes) has at least $(2m-1)/3$ children.
3. The root has at least 2 and at most $2 * \lfloor (2m-2)/3 \rfloor + 1$ children.
4. All leaves appear on the same level.
5. A nonleaf node with k children contains k -1 keys.

We consider the following data structure for each node of B*-tree. All nodes use an array for storing their keys, only leaves are linked together. For each key, the array contains a data element and a bit to indicate marked / unmarked keys. It means that we can perform a linear traversal of leaves without using

their parents. Besides, we need an extra array *CopyOfKeys* to keep a copy of each key inserted to the tree. The array also contains the data element and a bit to indicate marked / unmarked keys similar to the structure of nodes in the B*-tree. The extra array must be built while creating the B*-tree. If we perform inorder traversal of B*-tree, the successor of key x, denoted by *successor(x)*, is the key which we visit right after x. The node of key x, denoted by *node(x)*, locates the node where x is placed in. Variable $P_x.index$ is used to indicate the rank of the processor associated with the key x.

3 Deletion

Our deletion algorithm consists of five major stages. We first present a concise pseudocode and then describe each stage in details and for some non-trivial stages we will demonstrate time complexity analysis.

Algorithm Outline

1. Search for s keys and mark the found keys.
2. A copy of each unmarked key x in an internal node will be added to the beginning of $node(successor(x))$.
3. Compute parallel prefix sum for the keys in B*-tree leaves.
4. Perform a kind of packing algorithm on all marked / unmarked keys in B*-tree leaves in order to separate the two classes of marked and unmarked keys.
5. Reconstruct B*-tree from the remaining (unmarked) keys of the previous stage.

Stage 1: In this stage, s predetermined keys that must be deleted will be searched for and marked synchronously. We use CREW PRAM model, therefore the keys can be found concurrently straight and simple. Fig. 1(a) shows an example of B*-tree of maximum degree 6 storing 52 keys. Fig. 1(b) shows the B*-tree with the set of keys 1, 8, 10, 19, 20, 25, 26, 34, 40, 44, 57, 109, 120, 122, and 150 marked to be deleted. Since the CREW[1] model is exploited which allows us concurrent reads without memory access conflicts, a processor finds its corresponding key in $O(B\log_B n)$. Each processor in the worst case must traverse the height of the B*-tree that is equal to $O(\log_B n)$ and pursue a sequential search in a candidate node containing at most $2B-1$ keys which is a linear function with respect to B.

Marked keys are indicated by underlines with light color in Fig. 1(b). Note that the links connecting the leaves of the B*-tree are not depicted in the figure for the sake of simplicity.

Stage 2: We assign $n/2$ processors to array *CopyOfKeys* according to $P_x.index$. Each processor reads key x from array *CopyOfKeys* and finds it in B*-tree. Each processor P_x in an internal node whose key x is unmarked, finds the $successor(x)$

[1] For this stage on EREW model, we suggest using the parallel search algorithm proposed by Higham and Schenk [8].

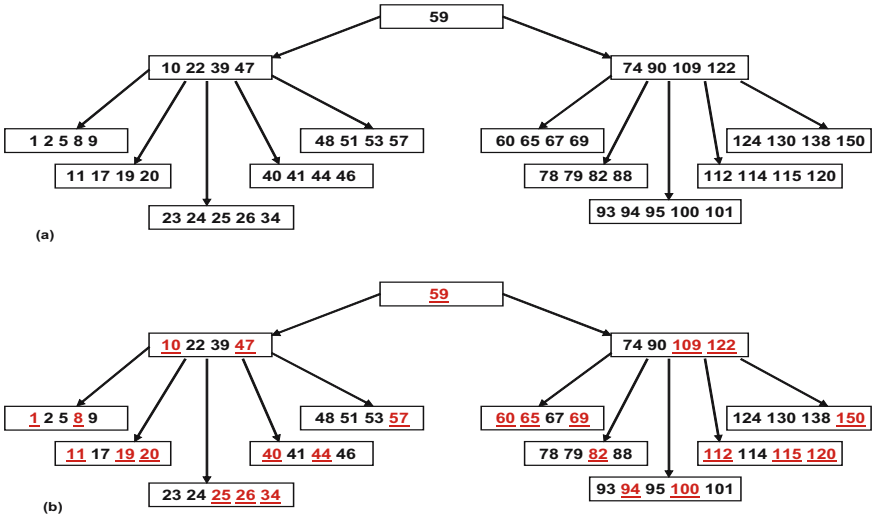


Fig. 1. Example of (a) B*-tree with m=6 (b) B*-tree after stage 1 of the algorithm

and adds a copy of key x to the beginning of the *node(successor(x))*. It's easy to find out that no more than exactly one processor attempts to push a key in front of keys contained in a specific leaf node. In other words, if x_1 is not equal to x_2 then $node(successor(x_1))$ won't be equal to $node(successor(x_2))$ either. After this step, the same processors read from array $CopyOfKeys[P_x.index + (n/2)]$ again. A similar procedure is repeated once more for these new assigned keys. At this point there is no unmarked key left unprocessed in the internal nodes and a copy of each unmarked key resides in a leaf node. Assigning each element of the array *CopyOfKeys* to a processor P_x is performed in O(1) because of using $P_x.index$ as a location indicator to assign keys in the array to the processors and finding the keys in B*-tree takes $O(B\log_B n)$ time. It takes $O(\log_B n)$ time to find $successor(x)$ [14]. Therefore this stage is totally performed in $O(B\log_B n)$ time. Afterward, we make a copy of all keys residing in leaves to a temporary array called $Temp$ for subsequent processing.

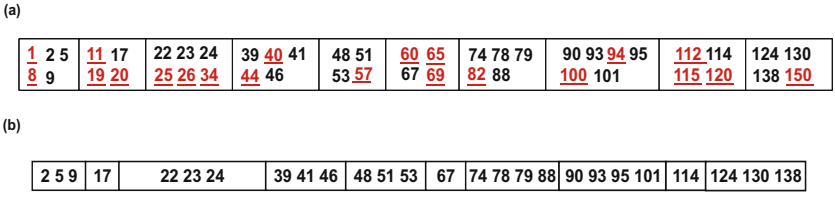


Fig. 2. (a) The contents of array $Temp$ after stage 2 (b) stage 4 performance of deletion algorithm ($PackedArray$)

Fig. 2(a) shows this stage performance on the B*-tree of Fig. 1(b) which copies each unmarked key in an internal node to the beginning of its successor node. For example key 22 goes in front of key 23; key 39 goes in front of key 40 and so on.

Stage 3: In this stage we perform a parallel prefix [11] sum on marked / unmarked bits for each key x_i in the array *Temp* in order to accomplish the next stage that is packing the array *Temp*. The packing procedure separates the class of unmarked keys from the marked one and accumulates the unmarked keys at the beginning of the array *Temp*. A marked bit of key x_i denotes a 1 and an unmarked bit denotes a 0. The parallel prefix sum algorithm on the marked / unmarked bits acts as follows.

1. Each processor P_i such that $0 \leq i \leq (p-1)$ computes $\sum_{j=0}^{n/p-1} x_j.bit$.
2. For $l = 0$ to $\lceil \log p \rceil$-1 do in parallel P_i and P_{i+2^l} exchange total sums T_1 and T_2 and both replace their total sum T with $T_1 + T_2$ and P_{i+2^l} also keeps a copy of T_1. (T_1 is the local sum of the bit elements of P_i's partition and T_2 is the local sum of the bit elements of P_{i+2^l}'s partition in array *Temp*).
3. Each processor P_i computes the total sum of all stored values and adds it to each partial sum.

This stage can be performed in $O(n/p + \log p)$ time and we have more than $n/2$ processors, thus the time complexity is $O(\log n)$. Note that the general parallel prefix algorithm presented can be run on EREW, as no need for CR or CW is arisen. More details can be found in a report presented by Qui and Akl [15].

Stage 4: In this stage we consider a problem that results in global rearrangement of keys. The problem consists of taking an input data set (array *Temp*), in which a subset of the items are marked and rearranging the keys so that all of the marked items succeed all of the unmarked items. Each key in the array *Temp* has an associated bit that is initially set to 1 for marked and 0 for unmarked keys. Note that this problem is equivalent to sorting a set of 0s and 1s into nondecreasing order.

The problem is solved using a parallel prefix sum to determine the rank of each 0 with respect to all 0s and the rank of each 1 with respect to all 1s. In fact we need not to compute the ranks for keys with a flag bit set to 1 because they are already keys to be removed from the B*-tree. If array *PPS* contains the parallel prefix sum on marked / unmarked bits for each key x_i in the array *Temp*, we can pack the unmarked keys to another array *PackedArray* with the following rule:

For each P_i such that $0 \leq i \leq (s-1)$ and *Temp*[i]'s bit is set to 0, copy the key in $Temp[i]$ to $PackedArray[i - PPS_i]$ otherwise do nothing (The processors responsible for marked keys will be idle). The same operation will be repeated for the remaining keys in the array $Temp$ if needed.

So this stage needs constant time due to availability of more than $n/2$ processors. Note that the resulting array *PackedArray* is sorted because the keys in

the leaves were sorted before this stage and the packing algorithm preserves the order of the keys. Fig. 2(b) shows the result of the packing algorithm on the keys in the leaves of the B*-tree depicted in Fig. 2(a).

Stage 5: After performing the previous stage, we are left with the array *PackedArray* of unmarked keys which is sorted. We assign a processor to each element in the array (we discard extra processors if such processors exist). Each processor P_i that is assigned to a key x_i in the array *PackedArray* participates in a parallel prefix sum operation similar to the procedure described in the stage 3 of the algorithm. This action is performed because we need to compute the total number of the keys in the array to make a new B*-tree.

If all the processors perform the sum operation on a dedicated memory location filled with the value of 1 then the last processor holds a resulting value equal to the number of all the participants i.e. the number of all the remained unmarked keys. The same time complexity analysis of stage 3 holds here. Therefore this action is performed in $O(\log n)$. We now give a brief description of the algorithm which builds the new B*-tree (reconstruction) and then we give an accurate pseudocode to show how it works.

In this stage we set off to build the new B*-tree from the unmarked keys by means of the *PackedArray*. This reconstruction is performed in a bottom-up fashion from the leaves to the root of the B*-tree with the structure explained in section 2. Although we attempt to build the B*-tree nodes in each step as full as possible, but there may exist such nodes that violate the minimum keys rule. After the completion of each iteration of this algorithm, one level of the B*-tree is built and linked to the upper level of the B*-tree. For constructing the nodes as much full as possible, partition boundaries (the keys that go up to the upper level) are located at $2B$ distance intervals. There is an exception to this case; the last two partitions should be dealt with in a different manner according to the minimum keys rule.

We must check to see whether or not the minimum keys rule is violated for the last partition. If the number of elements in the *PackedArray* is divisible by the maximum keys contained in a B*-tree node $(2B - 1)$ then the last two partitions need not to be manipulated, otherwise we compute the remainder of the division which is an integer called *Remain*. The decision how to revise the last two partitions depends on the value of *Remain*. If *Remain* is between $1/3(2B-1)$ and $2/3(2B - 1)$, only the lower (left) boundary of the last partition should go back $1/3(2B - 1)$ in order to prevent the violation of minimum keys rule in the last partition. If Remain is between 0 and $1/3(2B - 1)$, the lower boundary of the last partition should go back $2/3(2B - 1)$ and the lower boundary of the partition right before the last one should go back $1/3(2B - 1)$.

This revision should guarantee all the partitions i.e. B*-tree nodes conform to the minimum keys rule. During the promotion process of boundary keys to the upper level, they store their children addresses. This is necessary for organizing the links in each iteration of the algorithm. This way the B*-tree can be reconstructed in a recursive manner. If the number of elements in the *PackedArray* is less than or equal to $2 * \lfloor (4B-1)/3 \rfloor + 1$ then the algorithm is stopped because

we have reached the root and no further promotion of keys is required. Here is a detailed pseudocode of the described reconstruction phase.

Considerations:

- $P_x.index$ is equal to the index of each element x in the array $PackedArray$.
- $M[i]$ is a specified memory location in the shared memory.
- $Remain$ is the number of keys in the last partition of the $PackedArray$.
- A' is a temporary array which stores the keys that get promoted to the upper level.

Outline :

1. All processors compute the PPS for a value of 1 and the final result is written to $M[i]$
2. If $M[i] \leq 2 * \lfloor (4B-1)/3 \rfloor$ then go to 10
3. All processors read $M[i]$ and store the value to their local variable v
4. All processors compute $v/(2B-1)$ and store the result to v
5. All processors initialize their local variable $IsPartition$ to $P_x.index$ mod $(2B-1)$
6. For all processors that $IsPartition = 0$
   (a) Compute $NumberOfPartition \leftarrow P_x.index/(2B-1)$.
   (b) Compute $TotalNumberPartition \leftarrow \lfloor M[i]/(2B-1) \rfloor$
   (c) If $NumberOfPartition < TotalNumberPartition - 1$ then
      i. Write key to $A'[NumberOfPartition]$
   (d) Else
      i. Initialize local variable $Remain$ to $(M[i] - TotalNumberPartition * (2B-1))$
      ii. If $1/3(2B-1) \leq Remain < 2/3(2B-1)$
         A. If $NumberOfPartition = TotalNumberPartition - 1$ write the key to $A'[NumberOfPartition]$
         B. For all processors that are involved in this step, if $P_x.index = TotalNumberPartition * (2B-1) - 1/3(2B-1)$, the processor pushes its variable x back to the end of array A'.
      iii. Else if $0 < Remain < 1/3(2B-1)$
         A. For all processors that are involved in this step, if $P_x.index = TotalNumberPartition * (2B-1) - 1/3(2B-1)$, the processor pushes its varable x to $A'[TotalNumberPartition]$.
         Or if $P_x.index = (TotalNumberPartition - 1) * (2B-1) - 2/3(2B-1)$, the processor pushes its varable x to $A'[Total\ NumberPartition - 1]$
      iv. Else write key to $A'[NumberOfPartition]$
7. For all processors in array A', $Px.index \leftarrow NumberOfPartition$. $PackedArray$ replaces A'.
8. P_1 sets $M[i] \leftarrow TotalNumberPartition$
9. Go to step 2
10. End

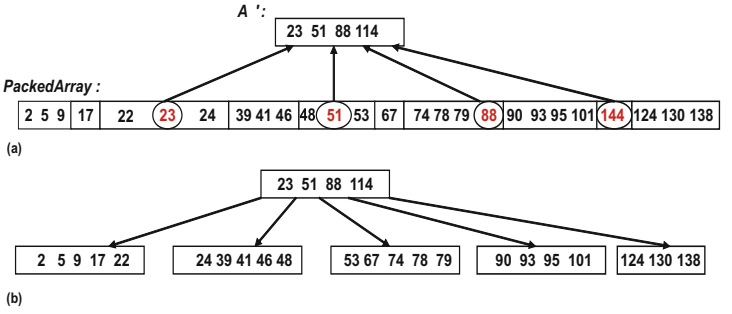


Fig. 3. (a),(b) Stage 5 of the deletion algorithm for rebuilding a new B*-tree

The final stage of the algorithm is performed in O ($\log_2 n + \log_B n$) time and makes a new B*-tree from scratch in a bottom up style from leaves to the root. It starts by making leaves then makes their parents and it repeats until the root is reached. Each processor which transfers its key to the parent's node links the key's pointer to its children nodes. Fig. 3(a) and (b) illustrate the steps of rebuilding the B*-tree. For deleting less than a half of all keys in B*-tree, we suggest a variant optimal algorithm proposed by H. Park, K. Park and Y. Cho [14]. Eventually, the following theorem is reached.

Theorem 1. *Concurrent deletion of more than a half of unsorted keys in a B*-tree of maximum degree $m \geq 3$ with the total of n keys can be done in optimal $O(\log_2 n + B \log_B n)$ time on the CREW PRAM with s processors where s is equal to the number of to-be-deleted keys and B is equal to half of the keys in an internal node containing maximum keys.*

4 Conclusion

We have proposed a new parallel algorithm on CREW PRAM that deletes large number of keys in a B*-tree. It takes O($\log_2 n + B\log_B n$) time with s processors, where n is the total number of keys in B*-tree, B equals to half of the keys in an internal node containing maximum keys and s is the number of unsorted keys to be deleted, which is at least half of the total keys in the B*-tree. Our proposed algorithm shows a reduction of factor B in the ($\log_2 n$)-term of the time complexity compared to the deletion algorithm presented in [14]. The algorithm presented here outperforms the previous ones due to the huge cost of refinement associated with them when the number of to-be-deleted keys in the B*-tree is large.

References

1. D.S Batory, B^+-trees and indexed sequential files: A performance comparison, ACM SIGMOD (1981), pp. 30-39.
2. R. Bayer, K.Unterauer, Prefix B-trees, ACM Trans. on Database System 2, no. 1 (March 1977), pp. 11-26.

3. R. Bayer, E. McCreight, Organization and maintenance of large ordered indexes, Acta Inform. 1 (3), 1972, pp. 173-189.
4. H. Berliner, The B*-tree search algorithm: a best first proof procedure, Tech. Rep. CMU-CA-78-112 Computer Sci. Depart. Carnegie-Mellon Univ., Pittsburg, 1978.
5. D. Comer, The ubiquitous B-tree, ACM Comput. Surveys 11, no. 2 (June 1979).
6. S. Das, M. Demuynck, Concurrent algorithms for B-trees, Tech. Rep. CRPDC-92-9, Center for Research in Parallel and Distributed Comput., Dept. Of Computer Sci., Univ. of North Texas, 3, 1992.
7. R. A. Hankins, J. M. Patel, Effect of node size on the performance of cache-conscious B^+-trees, in ACM SIGMETRICS'03, 2003.
8. L. Higham, E. Schenk, Maintaining B-trees on an EREW PRAM, J. Parallel Distributed Comput.22 (2), 1994, pp. 329-335
9. T. Johnson, D. Shasha, The performance of concurrent B-tree algorithms, ACM Trans. Database Systems 18(1), 1993, 51-101.
10. D. E. Knuth, The Art of Computer programming, Vol 3: sorting and searching 2^{nd} ed., Addison-Wesley Publ. Co., 1998.
11. R. E Ladner, M. J. Fischer, Parallel prefix computation, J. ACM, Vol. 27, no. 4, pp. 831-838, Oct. 1980.
12. V. Lanin, D. Shasha: A Symmetric concurrent B-tree algorithm, Proc. of fall joint computer conference, 1986, pp 380-389.
13. E. McCreight, Pagination of B-trees with variable length records, Comm. of the ACM 20, no. 9 (Sep. 1977).
14. H. Park, K. Park, and Y. Cho, Deleting keys of B-trees in parallel, J Parallel Distributed Comput. 64 (2004), pp. 1041-1050.
15. K. Qui, S. G. Akl, Parallel Maximum Sum Algorithms on Interconnection Network, Tech. Rep. No. 99-431.
16. J. Rao, K.A. Ross, Making B^+-tree cache conscious in main memory, in: SIGMOD'00,2000, pp. 475-486.
17. A.L Rosenberg, L. Synder, Time and Space Optimality in B-trees, ACM Trans. on Database System 6, no. 1 (Mar. 1981).
18. Y. Sagiv, Concurrent operation on B*-tree with overtaking, J. Comput. System Sci. (33) (1986) pp. 275-296.
19. V. Srinivasan, M. Carey, Performance of B-tree concurrency control algorithms, in: Proceedings of SIGMOD'91, 1991, pp. 416-425.
20. B. Wang, G. Chen, Cost-optimal parallel algorithms for constructing B-trees, in: Proceedings of the 20th Annual International Conference on Parallel Processing, 1991, pp. 294-295.

The Distributed Stigmergic Algorithm for Multi-parameter Optimization

Jurij Šilc and Peter Korošec

Jožef Stefan Institute, Computer Systems Department,
Jamova 19, 1000 Ljubljana, Slovenia
{jurij.silc, peter.korosec}@ijs.si
http://csd.ijs.si

Abstract. The paper presents a new distributed Multilevel Ant Stigmergy Algorithm (MASA) for minimizing the power losses in an electric motor by optimizing the independent geometrical parameters of the rotor and the stator. The efficiency of the algorithm, in sequential form, to solve that particular optimization problem has already been shown in literature. However, even if this method offers good quality of solution, it still needs considerable computational time. With distributed implementation of the MASA the computation time is drastically decreased (from one day to few hours) without any noticeable loss in solution quality.

1 Introduction

Multi-parameter optimization is the process of finding the point in the parameter space $P = \{p_1, p_2, \ldots, p_D\}$ where a cost function $f(P)$ is minimized according to the feasible set of parameters p_i, $i = 1, 2, \ldots, D$, that satisfy the constraints. Very often this cost function contains information about the problem target and the constraints that the solution has to meet (constrained optimization). These constraints define the region of the design space where the solution has to be comprised – called the feasibility region. Optimizing a multi-parameter function is usually a continuous problem.

There is no universal optimization algorithm to solve such an optimization problems. Many of the problems arising in real-life applications are NP-hard. Hence, one usually solves large instances with the use of approximate methods that return near-optimal solutions in a relatively short time. Algorithms of this type are called *heuristics*. The upgrade of a heuristic is a *metaheuristic* [1]: a set of algorithmic concepts that can be used to define a heuristic method applicable to a wider set of different problems. A particularly successful metaheuristic based on *stigmergy* is observed in colonies of real ants [2]. Stigmergy is a method of communication in decentralized systems in which the individual parts of the system communicate with one another by modifying their local environment. It was first observed in nature as a class of mechanisms that mediate animal-animal interactions (e.g., ant trails, termite nest-building, ant corpse-gathering). The term stigmergy (from the Greek $\sigma\tau\iota\gamma\mu\alpha$ = sting, and $\varepsilon\rho\gamma o\varsigma$ = work) was originally defined by the French biologist Pierre-Paul Grassé in his pioneering

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 92–99, 2006.

studies on the reconstruction of termite nests [4]. He defined it as: "Stimulation of workers by the performance they have achieved." Ants communicate with one another by laying down pheromones along their trails, so an ant colony is a stigmergic system.

An ant-colony metaheuristic is normally used for solving discrete, combinatorial optimization problems, but in this paper we will show a successful implementation on a numerical, multi-parameter optimization problem that is often solved by algorithms for continuous optimization. Because of the nature of the ant-based algorithms we first had to discretize the continuous, multi-parameter problem and translate it into graph representation.

The rest of the paper is organized as follows. The Ant Stigmergy Algorithm is defined in Section 2. In Section 3, the multilevel approach is explained. We round up with the Distributed Multilevel Ant Stigmergy Algorithm is described in Section 4, followed by application of the algorithm to the electric motor design in Section 5. Finally, we conclude the paper in Section 6.

2 The Ant Stigmergy Algorithm

In this section, we introduce the basic concept and major issues pertaining to ant stigmergy algorithm. First, we translate the multi-parameter problem into a directed graph and then use some sort of optimization technique to find the cheapest path in the constructed graph; this path consists of the values of the optimized parameters. In our case, we use an optimization algorithm, the routes of which can be found in the ant-colony optimization (ACO) method [3]. The so-called *Ant Stigmergy Algorithm* (ASA) consists of two main phases: (a) initialization and (b) optimization (see Fig. 1).

```
graph = Initialization(parameters)
GraphInitialization(initial pheromone amount)
WHILE NOT ending condition DO
    FOR all ants in colony DO
        path = FindPath(probability rule)
        Evaluate(path)
    END FOR
    UpdatePheromone(all found paths vertices)
    DaemonAction(best path)
    EvaporatePheromone(all vertices)
END WHILE
```

Fig. 1. The pseudocode of the Ant Stigmergy Algorithm

(a) *Initialization* consists of translation the parameters of the problem into a search graph. This way we translate the multi-parameter problem into a problem of finding the cheapest path. For each parameter $p_d, d = 1, \ldots, D$, parameter value $v_{\langle d,i \rangle}, i = 1, \ldots, n_d$, $n_d = |p_d|$, represents one vertex in a search graph, and each vertex is connected to all the vertices that belong to the next parameter

p_{d+1}. Once we have translated the multi-parameter problem into one of finding the cheapest path, we can deploy the initial pheromone values on all the vertices.

(b) *Optimization* consists of finding the cheapest path. We have a number of ants that all simultaneously start from the *start* vertex. The probability with which they choose the next vertex depends on the amount of pheromone on the vertices. The ants use a probability rule to determine which vertex will be chosen next. The ants repeat this action until they get to the ending vertex. Now we evaluate the gathered parameter values of each ant (that can be found on its path). Then each ant returns to the *start* vertex and on the way it deposits pheromones on the vertices according to the evaluation result: the better the result the more pheromone is deposited on the vertices, and vice versa. After all the ants return to the start vertex we do a so-called daemon action; this consists of depositing some additional pheromones on what is currently the best path, and also a smaller amount on a neighboring path. Afterwards, the pheromones are evaporated on all vertices, i.e., the amount of pheromones is decreased by some predetermined percentage on each vertex. The whole procedure is repeated until some ending condition is met.

3 Multilevel Approach

We considered the multilevel approach and its potential to aid the solution of optimization problems. The multilevel approach is a simple one, which at its most basic involves recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is found (sometimes for the original problem, sometimes at the coarsest level) and then iteratively refined at each level. As a general solution strategy the multilevel procedure has been in use for many years and has been applied to many problem areas [9]. The multilevel approach consists of two main phases: (a) coarsening and (b) refinement.

(a) *Coarsening* is done by merging two or more neighboring vertices into one vertex; this is done in L iterations (we call them levels $\ell = 1, 2, \ldots, L$). Let us consider coarsening from level ℓ to level $\ell + 1$ at a distance d: $V_d^\ell \xrightarrow{coars} V_d^{\ell+1}$. Here $V_d^\ell = \{v_{\langle d,1\rangle}^\ell, \ldots, v_{\langle d,n_d^\ell\rangle}^\ell\}$ is a set of vertices at level ℓ and distance d of the search graph $\mathcal{G}$, where $1 \leq d \leq D$. If n_d^1 is the number of vertices at a starting level of coarsening and distance d, then for every level ℓ the equation $n_d^{\ell+1} = \lceil \frac{n_d^\ell}{s_d^\ell} \rceil$ is true, where s_d^ℓ is number of vertices at level ℓ, which merge into one vertex at level $\ell + 1$. So what we do is we divide V_d^ℓ into $n_d^{\ell+1}$ subsets, where $V_d^\ell = \bigcup_{k=1}^{n_d^{\ell+1}} V_{\langle d,k\rangle}^\ell, \forall i, j \in \{1, \ldots, n_d^{\ell+1}\} \wedge i \neq j : V_{\langle d,i\rangle}^\ell \cap V_{\langle d,j\rangle}^\ell = \emptyset$. Each subset is defined as follows: $V_{\langle d,1\rangle}^\ell = \{v_{\langle d,1\rangle}^\ell, \ldots, v_{\langle d,s_d^\ell\rangle}^\ell\}, V_{\langle d,2\rangle}^\ell = \{v_{\langle d,s_d^\ell+1\rangle}^\ell, \ldots, v_{\langle d,2s_d^\ell\rangle}^\ell\}, \ldots, V_{\langle d,n_d^{\ell+1}\rangle}^\ell = \{v_{\langle d,(n_d^{\ell+1}-1)s_d^\ell+1\rangle}^\ell, \ldots, v_{\langle d,n_d^\ell\rangle}^\ell\}$.

Set $V_d^{\ell+1} = \{v_{\langle d,1\rangle}^{\ell+1}, \ldots, v_{\langle d,n_d^{\ell+1}\rangle}^{\ell+1}\}$ is a set of vertices at distance d at level $\ell+1$, where $v_{\langle d,k\rangle}^{\ell+1} \in V_{\langle d,k\rangle}^{\ell}$ is selected on some predetermined principle. For example, random pick, the most left/right/centered vertex in the subset, etc. The outline of the Coarsening() pseudo code is as follows:

$$\text{FOR } k = 1 \text{ TO } n_d^{\ell+1} \text{ DO } v_{\langle d,k\rangle}^{\ell+1} = \mathsf{SelectOneVertex}(V_{\langle d,k\rangle}^{\ell})$$

(b) *Refinement:* Because of the simplicity of the coarsening, the refinement itself is trivial. Let us consider refinement from level l to level $l-1$ at distance d: $V_d^{\ell} \overset{refin}{\longrightarrow} V_d^{\ell-1}$. The outline of the Refinement() pseudo code is as follows:

```
FOR k = 1 TO n_d^ℓ DO
    FOR each v_⟨d,i⟩^(ℓ-1) ∈ V_⟨d,k⟩^(ℓ-1) DO v_⟨d,i⟩^(ℓ-1) = v_⟨d,k⟩^ℓ
END FOR
```

Now, we merge the ASA and multilevel approach into one. This method is called the *Multilevel Ant Stigmergy Algorithm* (MASA) (see Fig. 2).

```
graph = Initialization(parameters)
FOR ℓ = 1 TO L DO
    Coarsening(graph[ℓ])
END FOR
GraphInitialization(initial pheromone amount)
FOR ℓ = L DOWNTO 1 DO
    WHILE NOT current level ending condition DO
        FOR all ants in colony DO
            path = FindPath(probability rule)
            Evaluate(path)
        END FOR
        UpdatePheromone(all found paths vertices)
        DaemonAction(best path)
        EvaporatePheromone(all vertices)
    END WHILE
    Refinement(graph[ℓ])
END FOR
```

Fig. 2. The pseudocode of the Multilevel Ant Stigmergy Algorithm

4 Distributed Algorithm

Like many other metaheuristic approaches, the MASA admits direct parallelization schemes and parallelism can be exploited at one or more scales [8]. In our implementation, we decide on the largest scale where entire searches can be performed concurrently. Such implementation, called *parallel interacting ant colonies* [7], is based on the well-known server/client approach. Non-distributed MASA approach presented in previous section base on single ant colony. But in distributed approach, we split this colony into N sub-colonies, where N represents number of processors. Each sub-colony searches for a solution according to the following algorithm:

```
Server:  StartAllClients()
         WHILE NOT ending condition DO
             IF ReceivedEvaluatedPaths(client) THEN
                 BroadcastPaths(all other clients)
             END IF
         END WHILE
         StopAllClients()

Client:  graph = Initialization(parameters)
         FOR ℓ = 1 TO L DO
             Coarsening(graph[ℓ])
         END FOR
         GraphInitialization(initial pheromone amount)
         FOR ℓ = L DOWNTO 1 DO
             WHILE NOT current level ending condition DO
                 FOR all ants in sub-colony DO
                     path = FindPath(probability rule)
                     Evaluate(path)
                 END FOR
                 SendEvaluatedPathsToServer(all ants)
                 ReceivePathsFromServer(from all other clients)
                 UpdatePheromone(all found and received paths vertices)
                 DaemonAction(best path)
                 EvaporatePheromone(all vertices)
             END WHILE
             Refinement(graph[ℓ])
         END FOR
```

Fig. 3. The pseudocode of the Distributed Multilevel Ant Stigmergy Algorithm

This algorithm is similar to the MASA in previous section, except that at given iterations, an exchange of information between the sub-colonies occurs. With the use of SendEvaluatedPathsToServer() function the paths with update pheromone amount is send to server which then with the use of BroadcastPaths() function broadcasts this information to all other clients (sub-colonies). The update pheromone amount is determined by Evaluate() function. On the other hand, the information (paths with updated pheromone amount) is gathered from other sub-colonies with the use of ReceivePathsFromServer() function. The last difference is in the UpdatePheromone() function. Here the pheromone is not deposited only on found paths but also on a received ones.

5 The Case Study

In a conventional design procedure for an electric motor the initial estimation for the geometry of the rotor and the stator is made by an experienced engineer. The suitability of this geometry is then usually analyzed by means of a numerical simulation of the electromagnetic field. The manual procedure is repeated until the satisfied evaluation results are obtained. The advantage of this

approach is that with their experience the engineers can significantly influence the progress of the design process and react intelligently to any noticeable electromagnetic response with proper geometry redesign. However, this conventional design approach can be upgraded with stochastic optimization techniques which, in connection with reliable numerical simulators, allow for highly automated design process where the need for an experienced engineer to navigate the process is significantly reduced. Actually this is multi-parameter optimization. The usefulness and efficiency of the ant-based metaheuristic, in sequential form, to solve the problem of minimizing the losses in an electric motor has already been shown in our previous work [5]. The average solution obtained with the algorithm is 25% better than a solution recently found using a genetic approach [6].

The efficiency of an electric motor is defined as the ratio of the output power to the input power and depends on various power losses. They include copper losses, iron losses, and additional losses, such as brush losses, and losses due to ventilation and friction, but the last three losses are not significantly affected by the geometry of the rotor and the stator. The optimization task is to find the geometry parameter values that would generate the rotor and the stator geometry with minimum power losses.

There are several invariable and variable parameters that define the rotor and the stator geometry [6]. Invariable parameters are fixed; they cannot be altered, either for technical reasons (e.g., the air gap) or because of the physical constraints on the motor (e.g., the radius of the rotor's shaft). Variable parameters do not have predefined optimum values. Some variable parameters are mutually independent and without any constraints. Others are dependent, either on some invariable parameters or on mutually independent ones. In our case, 10 mutually independent variable parameters defining the rotor and the stator geometry are subject to optimization. The optimization task is to find the geometry parameter values that would generate the rotor and the stator geometry with minimum power losses.

To evaluate settings of the rotor and the stator geometry parameters with respect to the resulting power losses, we used the ANSYS finite-element method simulation package. The evaluation of a single solution is time consuming task. For example, evaluation through ANSYS simulation on an AMD Opteron^{TM} 1.8 GHz computer takes approximately one minute. To find an acceptable solution a few thousand evaluations are needed. These two facts alone motivated us in implementation of distributed MASA.

The computer platform used to perform the experiments was 8-node cluster connected via Giga-bit switch, each node consisted of two AMD Opteron^{TM} 1.8 GHz processors and 2 GB of RAM.

In the experiments, the stopping criterion for the MASA was given by the number of solutions to be evaluated. It was set to 2240 evaluations per run and this value was chosen considering the computational complexity of the optimization procedure. Other algorithm parameters were set as follows. The MASA operated with seven levels, at each level all ants in sub-colony climbs down the graph 20 times ("level ending condition"). Total number of ants in all

Table 1. An optimized electric motor by the distributed MASA (power losses [W])

	Number of processors N			
	1	2	4	8
Best	120.63	122.26	122.86	129.30
Worst	128.71	129.74	138.46	136.47
Mean ± Std	124.45 ± 2.33	125.28 ± 2.83	129.36 ± 4.85	133.34 ± 3.18

sub-colonies is 16, while the number of ants in each sub-colony depends on the number of processors: eight ants for $N = 2$, four for $N = 4$, and two for $N = 8$. For each distribution ($N = 1$, 2, 4 or 8), we ran our algorithm 10 times where we compared solution quality and computational time.

In Table 1 we can see that the MASA optimizes electric motor design with power losses in range from 120.63 W to 138.46 W. Even the worst found solution is still much better that the one found by an expert designer (177.9 W) and it is used in production. More detailed examination of Table 1 reveals a slight degradation of solution quality with increasing number of processors. The main reason for this is in huge percent (approx. 99.98%) of infeasible solutions in the search space and in fact that the evaluation of feasible solution is 60 times more time consuming than the evaluation of infeasible solution. Let us consider example with one and eight processors. In the first case ($N = 1$) 16 ants climb down simultaneously while in the second one ($N = 8$) two ants climb down simultaneously in each processors. If in the first case only one ant finds a feasible solution, the rest of 15 ants waits for this ant to finish solution evaluation and do not search for new feasible solutions. But already in the next climb down all 16 ants use this newly acquired information(gathered from the ant from previous climb down). So, 15 searches for feasible solution were done without this information at the most. Now, let us take into account the second case. If again only one ant finds a feasible solution then just the ant on the same processor waits, but all the rest of the ants (14) search for new feasible solutions undisturbed. Here it is possible that none of 14 ants do not use this information. For example, if in the first climb down of the current level one ant finds a feasible solution it takes approx. 60 seconds to evaluate it. In the mean time on all the other processors the climb downs continue and if none of the ants find a feasible solution all of the predetermined number of evaluations per level is used without the use of knowledge gathered by the evaluation of the solution of the first ant. This scenario can not happen in the first case (where $N = 1$) where 15 searches were lost at the most.

Table 2. Computation time in [min]

	Number of processors N			
	1	2	4	8
Best	1295	623	254	113
Worst	1552	811	396	198
Mean ± Std	1481 ± 81	702 ± 61	332 ± 40	182 ± 11

In Table 2 we see that speed-up is greater than N, which only confirms our statement from previous paragraph that number of infeasible solutions increases with the number of processors.

6 Conclusion

In this paper we presented a new distributed multilevel ant stigmergy algorithm for minimizing the power losses in an electric motor by optimizing the independent geometrical parameters of the rotor and the stator. We have showed that with distributed computing we can drastically decrease the computation time (from one day to few hours) without any noticeable loss in solution quality.

References

1. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. **35** (2003) 268–308
2. Dorigo, M., Bonabeau, E., Theraulaz, G.: Ant algorithm and stigmergy. Future Gener. Comp. Sy. **16** (2000) 851–871
3. Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization. Artif. Life **5** (1999) 137–172
4. Grassé, P.-P.: La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis et Cubitermes* sp. La théorie de la stigmergie. Ins. Soc. **6** (1959) 41–81
5. Korošec, P., Šilc, J.: The multilevel ant stigmergy algorithm: An industrial case study. Proc. 8th Joint Conference on Information Sciences, Salt Lake City, UT, July 2005, pp. 475–478
6. Papa, G., Koroušić-Seljak, B.: An artificial intelligence approach to the efficiency improvement of a universal motor. Eng. Appl. Artif. Intel. **18** (2005) 47–55
7. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. J. Parallel Distr. Com. **62** (2002) 1421–1432
8. Stützle, T.: Parallelization strategies for ant colony optimization. Lect. Notes Comp. Sc. **1498** (1998) 722–741
9. Walshaw, C.: Multilevel refinement for combinatorial optimisation problems. Ann. Oper. Res. **131** (1-4) (2004) 325–372

Total Exchange Performance Modelling Under Network Contention

Luiz Angelo Barchet-Steffenel* and Grégory Mounié

ID - IMAG Laboratory, MOAIS Team**,
ZIRST 51, Avenue Jean Kuntzmann, F-38330 Montbonnot St. Martin, France
{Luiz-Angelo.Steffenel, Gregory.Mounie}@imag.fr

Abstract. One of the most important collective communication patterns for scientific applications is the *many to many*, also called *complete exchange*. Although efficient All-to-All algorithms have been studied for specific networks, general solutions like those found in well known MPI distributions are strongly influenced by the congestion of network resources. In this paper we present our approach to model the performance of the All-to-All collective operation. Our approach consists in identifying a contention factor that characterises the network environment, and using it to augment a contention-free communication model. This approach allows an accurate prediction of the performance of the All-to-All operation over different network environments with a small cost. Indeed, we demonstrate the accuracy of our approach by presenting our experiments with three different network environments, Fast Ethernet, Giga Ethernet and Myrinet.

1 Introduction

One of the most important collective communication patterns for scientific applications is the *many to many* (also called *complete exchange* [1]), in which each process holds P different data items that should be distributed among the P processes, including itself. An important example of this communication pattern is the All-to-All collective operation, where all messages have the same size m. The All-to-All operation is frequently used for matrix transposition, two-dimensional Fourier Transform, conversion between storage schemes (remapping of arrays in HPF compilers), shuffle permutation, N body problems and matrix-vector multiplication.

Although efficient All-to-All algorithms have been studied for specific networks structures like meshes, hypercubes, tori and circuit-switched butterflies, general solutions like those found in well-known MPI distributions rely on direct point-to-point communication among the processes. Because all communications are started simultaneously, the overall communication time of the MPI_AlltoAll operation is strongly influenced by the congestion of network resources.

* Supported by grant BEX 1364/00-6 from CAPES - Brazil.
** This project is supported by CNRS, INPG, INRIA and UJF.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 100–107, 2006.

Consequently, the performance modelling of the All-to-All operation is not a simple task. Indeed, most existing performance models are unable to reflect the impact of network contention, while others are too complex to be used in real situations.

In this paper we present a new approach to model the performance of the All-to-All collective operation. Our strategy consists in identifying a *contention factor* that characterises the network environment, and using it to augment a contention-free performance model. This approach allows the prediction of the performance of the All-to-All operation with efficiency and reduced cost. Indeed, to demonstrate our approach, we present the results we obtained with three different network environments, Fast Ethernet, Giga Ethernet and Myrinet.

The rest of this paper is organised as follows: Section 2 presents the definitions and the test environment we will consider along this paper. Section 3 presents a survey of performance modelling under communication contention. Section 4 presents our approach to model the performance of the All-to-All operation, validating it against experimental data. Finally, Section 5 presents our conclusions as well as the future directions of our work.

2 Performance Models and System Definitions

There are several performance models adequate to represent message-passing parallel programs, most of them based on *delay* [2], BSP [3] or LogP [4]. Although these last two performance models are equivalent in most circumstances [5], LogP is slightly more adapted to our problem as it includes the notion of finite network capacity, which is especially useful to reflect the network contention. Hence, in this paper we model collective communications using the *parameterised LogP* model (*pLogP*) [6], an extension of the LogP performance model that can accurately handle both small and large messages with a low complexity.

All along this paper we use $\boldsymbol{g(m)}$, $\boldsymbol{os(m)}$ and $\boldsymbol{or(m)}$ to respectively represent the communication gap, the send and the receive overheads of a message of size m, $\boldsymbol{L}$ as the communication latency between two nodes, and $\boldsymbol{P}$ as the number of nodes involved in the operation. The *pLogP* parameters used to feed our models were obtained with the MPI LogP Benchmark tool [7], and are presented in Figure 1.

The experiments were conducted on the **icluster-2**[1] cluster at the INRIA Rhône-Alpes computing centre and on the **IDPOT**[2] cluster at the ID-IMAG Laboratory. The icluster-2 contains 104 Bi-Itanium2 machines (900MHz, 3GB, Red Hat AS 3.0 with kernel 2.4.21smp) interconnected by a switched Fast Ethernet and a Myrinet network. The IDPOT cluster contains 48 Bi-Xeon machines (2.5GHz, 1.5GB, Debian with kernel 2.4.26smp) interconnected by a Giga Ethernet network. The experiments used LAM-MPI 7.0.4 and consisted on 100 measures for each set of parameters (message size, number of processes), from which the average values are considered in this paper.

[1] *http://i-cluster2.inrialpes.fr/*
[2] *http://idpot.imag.fr/* or *http://frontal38.imag.fr*

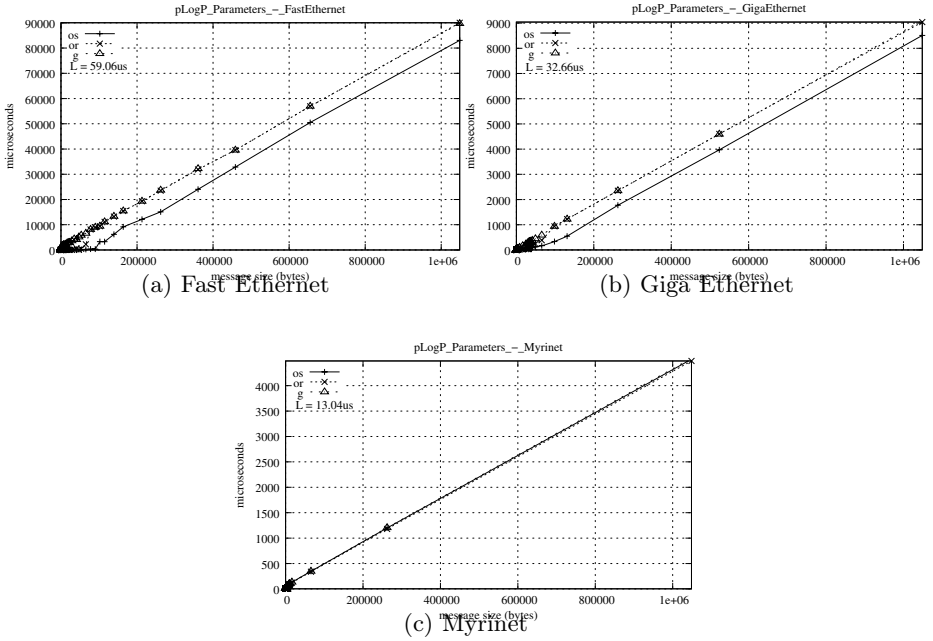


Fig. 1. pLogP parameters for the icluster-2 and IDPOT networks

3 Modelling the All-to-All Operation

In the *All-to-All* operation, every process holds $m \times P$ data items that should be equally distributed among the P processes, including itself. Because general implementations of the All-to-All collective communication rely on direct point-to-point communication among the processes, the network can easily become saturated, and by consequence, degrade the communication performance. Indeed, Chun and Wang [8] demonstrated, that the overall execution time of intensive exchange collective communications are strongly dominated by the network contention and congestive packet losses, two aspects that are very difficult to quantify. As a result, a major challenge on modelling the communication performance of the All-to-All operation is to represent the impact of network contention.

Unfortunately, most communication models like those presented by [1] are simple extensions of the *one-to-many* communication pattern (e.g. the Scatter operation, where a single process sends different messages of size m to each other process). By consequence, such models do not take into account the influence of the network contention, and therefore, are not accurate enough to predict the performances of an All-to-All operation.

Indeed, the development of contention-aware communication models is relatively recent, as shown by Grove [9]. For instance, one of the first models that considered the effects of resource contention was presented by Adve [10].

This model considers that the total execution time of a parallel program is the sum of four components, namely:

$$T = t_{computation} + t_{communication} + t_{resource-contention} + t_{synchronisation}$$

While conceptually simple, this model was non-trivial in practice because of the non-deterministic nature of resource contention, and because of the difficulty to estimate average synchronisation delays.

In fact, the non-deterministic behaviour of the network contention is a major obstacle to modelling communication performance. A proposal to circumvent this limitation was introduced by Clement *et al.* [11], which suggested a way to account contention in shared networks such as non-switched Ethernet consisting in a contention factor γ that augments the linear communication model T:

$$T = l + \frac{b\gamma}{W}$$

where l is the link latency, b is the message size and W is the bandwidth of the link, and γ is equal to the number of processes. A restriction on this model is that it assumes that all processes communicate simultaneously, which is only true for a few collective communication patterns. Anyway, in the cases where this assumption holds, they found that this simple contention model greatly enhanced the accuracy of their predictions for essentially zero extra effort.

The principle of a contention factor is complemented by the work of Labarta et *al.* [12], that tried to approximate the behaviour of the network contention by considering that if there are m messages ready to transit, and only b available buses, then the messages are serialised in $\lceil \frac{m}{b} \rceil$ communication waves.

Most recently, some works on contention-aware performance models have been published. LoGPC [13] is an extension of the LogP model that tries to determine the impact of network contention through the analysis of k-ary n-cubes. Unfortunately, the complexity of this analysis makes too hard the application of such model in practical situations.

Another approach was presented by Chun [8], in which the contention is considered as a component of the communication latency, and by consequence, their model uses different latency values according to the message size. Although easier to use than LoGPC, the model from Tam does not take into account the number of communicating processes, which is clearly related to the occurrence of network contention.

4 A Different Approach

Similarly to Clement *et al.* [11], we assume that the contention is sufficiently linear to be modelled. Our approach, however, consists on identifying theoretical performance bounds for the All-to-All operation and deriving a contention factor that fits our predictions with pre-existent experimental results. We consider that the network contention depends mostly on the physical characteristics of the

network (network cards, links, switches), and consequently, the ratio between the theoretical bounds and the practical results represents a "*contention signature*" of the network. Once identified the *signature* of a network, it can be used in further experiments to predict the communication performance.

In the case of the All-to-All operation, we explore the limitations of the *1-port* communication model. For instance, although a process can only send a message to a process each time, the *1-port* model allows a process to simultaneously send a message to one process and receive a message from another one. Hence, in a contention-free situation, a process would be able to access the network interface as soon as the precedent *send* operation returned (while the *receive* operation runs simultaneously in the background). In terms of pLogP parameters, this means that a contention-free process needs only g time units to simultaneously send a message to a process and receive a message from another one.

At the other hand, processes subjected to network contention may not be able to send and receive messages simultaneously. Due to the congestion of network resources, a process may not be able to overlap send and receive, and therefore, can be forced to serialise its communications. In pLogP terms, such processes need g time units to send each message, plus or time units to receive a message.

By consequence, a *Contention-Free* situation represents the capability to overlap send and receive operations with no extra cost, while in a *Contention* situation the processes need to serialise their transmissions due to the network contention. Thus, we model the All-to-All operation using these two situations as represented on Table 1. It worth noting that in real situations the performances of the All-to-All operation may exceed the predictions for the *Contention* case, as there are other factors that can influence the communications besides the physical environment. Even though, by defining a network signature based on the theoretical bounds, we are able to quantify the network contention effects regardless the sources of contention.

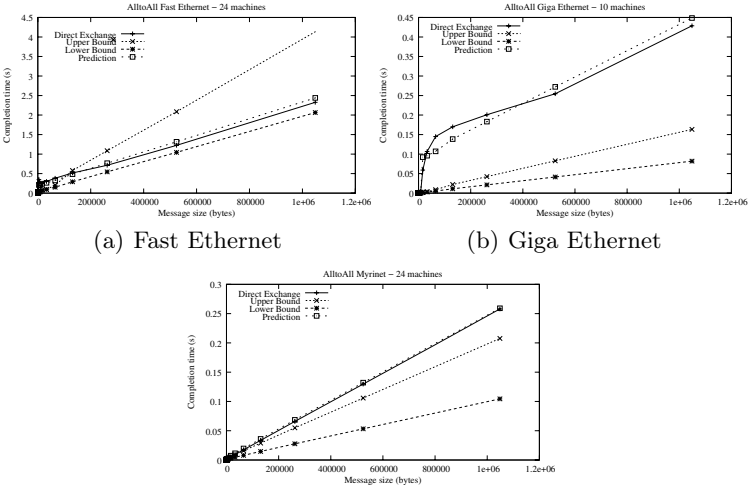
Table 1. Theoretical performance bounds for the *All-to-All* operation

	Communication Models
Under Contention	$(P-1) \times g(m) + (P-1) \times or(m) + L$
Contention-free	$(P-1) \times g(m) + L$

4.1 Practical Results

To illustrate our approach to predict the performance of the All-to-All operation in an environment subjected to network contention, we use a *Direct Exchange (DE)* algorithm similar to the current MPI_AlltoAll implementation from both LAM 7.0.4 and MPICH 1.2.5. In this algorithm, all nodes start to communicate simultaneously, but the contact list of each node is rotated to avoid overloading a single process each "round".

We present in Figure 2 an example with the measured performance for the *DE* algorithm as well as the predicted performance bounds. It can be observed that



(a) Fast Ethernet (b) Giga Ethernet

(c) Myrinet

Fig. 2. Theoretical bounds and performance predictions

the completion time for the *DE* algorithm usually differs from the *Contention-free* case in a non-negligible amount, which clearly indicates the presence of network contention.

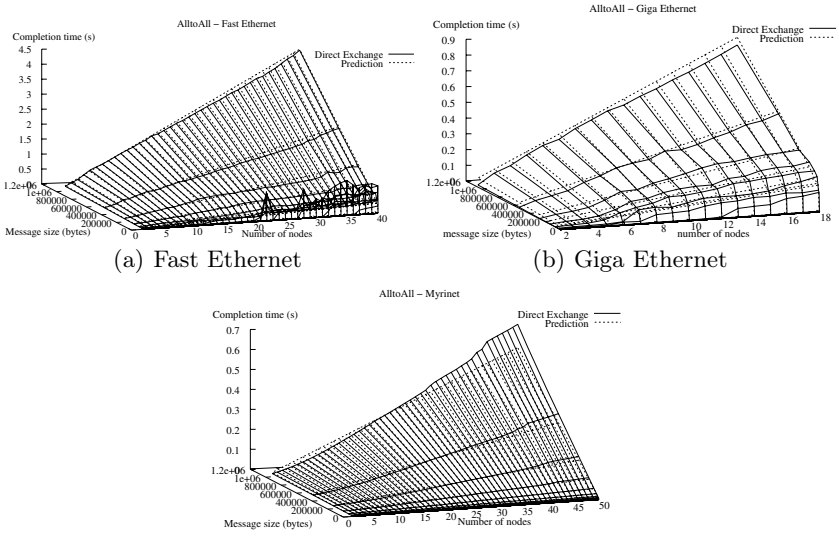
Next, we determine a contention factor γ between the predicted *Contention-free* and *Contention* performances such that the predictions fit the performance of the All-to-All operation. This contention factor γ is constant and depends only on the network characteristics (the *network signature*), whilst the *Contention-free* and *Contention* bounds depend on the number of processes and on the message size. In some cases, a supplementary factor δ, dependent on the number of processes P, may be necessary to represent additional costs like, for example, the overhead of message segmentation, buffer overflow or residual synchronisation delays. Hence, a performance model for the All-to-All operation can be defined by:

$$T = Free + (Contention - Free) \times \gamma + (P-1) \times \delta$$

$$= (P-1) \times g(m) + L + (P-1) \times or(m) \times \gamma + (P-1) \times \delta$$

$$= (P-1) \times (g(m) + or(m) \times \gamma + \delta) + L$$

Taking as basis the data from Figure 2, a contention factor γ that fits those performances is $\gamma = \frac{1}{10}$ for the Fast Ethernet network, $\gamma = \frac{9}{2}$ For the Giga Ethernet network and $\gamma = \frac{3}{2}$ for the Myrinet network. In the case of Fast Ethernet and Giga Ethernet networks we need a supplementary δ_P for messages larger than 2kB, mostly due to reception buffers overflow. Hence, we approximate the behaviour of Fast Ethernet networks with $\delta_P = 7ms * P$ while Giga Ethernet needs $\delta_P = 9ms * P$. Using these contention factor values as the *network signatures*



(a) Fast Ethernet (b) Giga Ethernet

(c) Myrinet

Fig. 3. Performance predictions for the *All-to-All* operation

of our clusters, we could accurately predict the performance of the All-to-All operation for a wide range of processes with no extra cost. Therefore, Figure 3 presents a comparison between our predictions and the measured performances for the All-to-All operation with both Fast Ethernet, Giga Ethernet and Myrinet networks.

It also worth noting the instabilities observed in the case of the Fast Ethernet network when dealing with small messages and a large number of processes. We believe that these instabilities are due to a problem with the TCP implementation on Linux, as previously discussed in a precedent work [14].

5 Conclusions and Future Work

In this paper we present a new approach to model the performance of the All-to-All collective operation that is both simple and precise. Our method consists on identifying a contention factor that characterises the network environment, and using it to augment a contention-free performance model. This approach allows the prediction of the performance of the All-to-All operation over different network environments, with accuracy and reduced cost. Indeed, to demonstrate our approach, we present the results we obtained with three different network environments, Fast Ethernet, Giga Ethernet and Myrinet.

We intend to pursue our experiments on communication modelling using the GRID5000[3] facility, investigating the behaviour of collective communications with a larger number of machines and with other network environments such

[3] *http://www.grid5000.org*

as InfiniBand, and more specifically to the complete exchange operations, to automate the definition of γ and δ for a given network. We are also interested in the study of contention effects in the domain of small messages, subjected to important performance variations as represented by the δ factor itself.

References

1. Christara, C., Ding, X., Jackson, K.: An efficient transposition algorithm for distributed memory computers. In: High Performance Computing Systems and Applications. (1999) 349–368
2. Rayward-Smith, V.J.: UET scheduling with unit interprocessor communication delays. Discrete Applied Mathematics **1**(18) (1987) 55–71
3. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM **33**(8) (1990) 103–111
4. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eicken, T.: LogP - a practical model of parallel computing. Communication of the ACM **39**(11) (1996) 78–85
5. Skillicorn, D., Hill, J., McColl, W.: Questions and answers about BSP. Scientific Programming **6**(3) (1997) 249–274
6. Kielmann, T., Bal, H., Gorlatch, S., Verstoep, K., Hofman, R.: Network performance-aware collective communication for clustered wide area systems. Parallel Computing **27**(11) (2001) 1431–1456
7. Kielmann, T., Bal, H., Verstoep, K.: Fast measurement of LogP parameters for message passing platforms. In: 4th Workshop on Runtime Systems for Parallel Programming. LNCS Vol. 1800 (2000) 1176–1183
8. Chun, A.T.T., Wang, C.L.: Contention-aware communication schedule for high-speed communication. Cluster Computing: The Journal of Networks, Software Tools and Application **6**(4) (2003) 337–351
9. Grove, D.: Performance Modelling of Message-Passing Parallel Programs. PhD thesis, University of Adelaide (2003)
10. Adve, V.: Analysing the Behavior and Performance of Parallel Programs. PhD thesis, University of Wisconsin, Computer Sciences Department (1993)
11. Clement, M., Steed, M., Crandall, P.: Network performance modelling for PM clusters. In: Proceedings of Supercomputing. (1996)
12. Labarta, J., Girona, S., Pillet, V., Cortes, T., Gregoris, L.: DiP: A parallel program development environment. In: 2nd Euro-Par Conference. Volume 2. (1996) 665–674
13. Moritz, C.A., Frank, M.I.: LoGPC: Modeling network contention in message-passing programs. IEEE Transactions on Parallel and Distributed Systems **12**(4) (2001) 404–415
14. Barchet-Estefanel, L., Mounié, G.: Fast tuning of intra-cluster collective communications. In: Proceedings of the Euro PVM/MPI 2004. LNCS Vol. 3241 (2004) 28–35

Towards the Performance Visualization of Web-Service Based Applications

Marian Bubak[1,2], Wlodzimierz Funika[1], Marcin Koch[1], Dominik Dziok[1], Allen D. Malony[3], Marcin Smetek[1], and Roland Wismüller[4]

[1] Inst. Comp. Science, AGH, Krakow, Poland
[2] ACC CYFRONET-AGH, Krakow, Poland
[3] Dept. of Computer and Information Science, University of Oregon, Eugene, USA
[4] Fachgruppe BVS - Universität Siegen, Siegen, Germany
Phone: (+48 12) 617 44 66; Fax: (+48 12) 633 80 54
{bubak, funika, smetek}@uci.agh.edu.pl, malony@cs.uoregon.edu, ivn@icslab.agh.edu.pl, domin@student.uci.agh.edu.pl, roland.wismueller@uni-siegen.de

Abstract. In this paper we present an approach to building a monitoring environment which underlies performance visualization for distributed applications. Our focus is to make the J-OCM monitoring system and the TAU-Paravis performance visualization system to collaborate. J-OCM, based on the J-OMIS interface, provides services for on-line monitoring of distributed Java applications. The system uses J-OCM to supply monitoring data on the distributed application, whereas TAU-Paravis provides advanced visualization of performance data. We managed to integrate J-OCM into TAU/Paravis by developing additional software providing access to the monitor and transformation of raw monitor data into performance data which is presented with 3-D charts. At the end we present an extension, which introduces Web Service monitoring into the integrated environment.

Keywords: performance visualization, monitoring tools, OMIS, TAU, web service.

1 Introduction

The ability to monitor the execution of a distributed application and to measure its performance is a key issue in designing and deploying such applications [1]. A standard approach assumes a kind of pre-execution instrumentation of the source code. During execution the instrumented code generates monitoring information which are stored and presented to the user either in semi-on-line or off-line mode. Such an approach has many limitations: it requires often source recompilation, does not work with applications that execute very long, does not allow to control the execution of the application. As a result, there is a distinct need for performance analysis systems that can perform on-the-fly monitoring and allow for application control and interaction at run-time. In this paper we

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 108–115, 2006.

present our approach to building a monitoring environment and also the way of extending it to support the monitoring of Web Services which become an increasingly popular technology of distributed programming.

Our approach is based on the integration of J-OCM [2], a flexible monitoring system into the TAU-Paravis visualization package [3]. Up to now there were no tools which used J-OCM for performance analysis, so it was impossible to fully use this on-line monitoring system for performance analysis. We use J-OCM to supply on-the-fly monitoring data from a distributed application, while TAU-Paravis provides advanced visualization of performance data. As J-OCM and TAU-Paravis use different data models, our work is aimed at the development of additional packages to make them cooperate.

The paper is organized as follows: Section 2 introduces the main features of J-OCM. Next, Section 3 gives some details on SCIRun and TAU. Section 4 presents the concept and structure of the integrated monitoring environment. Next, Section 5 explains a way of extending the environment to support Web Services monitoring, followed by the features of 3-D performance visualization in Section 6. Then we give a short overview of related work. Finally, we sum up the work done and show some plans for further research.

2 J-OCM Monitoring System

J-OCM is a monitoring system for Java applications, compliant with the OMIS specification [4] extended by a support for Java, in form of the J-OMIS extension [5]. The idea of OMIS (On-line Monitoring Interface Specification) is to separate the functionality of a monitoring system from monitoring tools. The OMIS specification defines an interface that is an intermediate layer between them. The communication is based on the request-reply mechanism realized as a set of *services* while the processing of events uses the event-action paradigm. OMIS enables convenient access to performance objects like classes, methods, threads, or web services; they are identified by tokens. All performance object types, observable by the monitor, form an *objects hierarchy*. The OMIS concept allows multiple monitoring-based tools like profilers, debuggers, etc. to use a single monitoring system at the same time.

The J-OMIS specification, which underlies J-OCM, extends OMIS to match the monitoring of Java applications. It introduces new, specific for Java types of objects and services, to form an object hierarchy relevant to Java. It also divides the new object hierarchy into two kinds of objects – execution ones: nodes, JVMs, threads, and application ones: interfaces, classes, objects, methods. Each object has its own set of services divided to three groups: information services - to provide information about objects, manipulation services - to allow to change objects' states, and event services to trigger some actions when matching events occur. Event services are used by tools to program the monitoring system for getting specific data from a monitored application or manipulating it. J-OCM (see Fig. 1) is implemented as an extension to the OCM monitoring system [6].

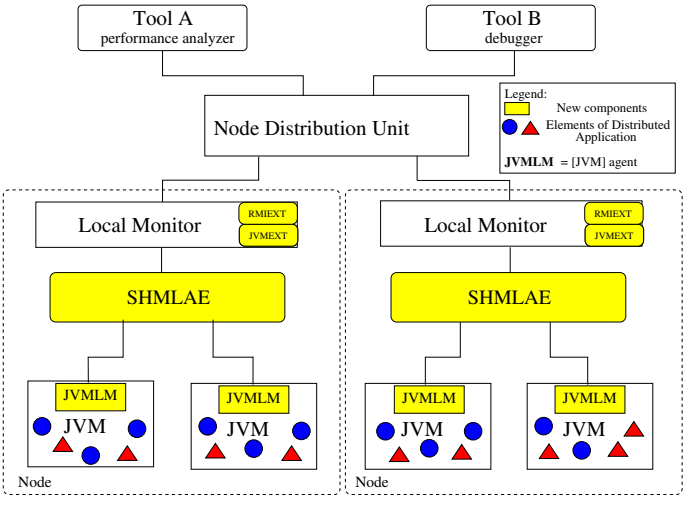


Fig. 1. Architecture of J-OCM

Recently, J-OCM was extended to enable the monitoring of web-service based applications written in Java. Some modifications have been introduced to the J-OMIS interface specification. The extension provides means for accessing web services and forwarding web-service specific events.

3 SCIRun and TAU

Within our work we are integrating J-OCM with TAU-Paravis visualization package. Paravis, which is a part of the TAU(Tuning And Analysis Utilities) monitoring environment , is developed at the University of Oregon in Eugene [7, 8]. Paravis introduces advanced 3-D visualization into TAU.

TAU-Paravis is built within SCIRun, a powerful Problem Solving Environment [9]. It is developed at the University of Utah, as an open source software. SCIRun can be used widely for solving various scientific problems. It consists of the modules that allow to perform complicated computations, data transformations, and provide advanced visualization. An application built within SCIRun is composed from the modules which can be connected one to another through pipes carrying data. A module usually gets input data, performs some computations on it and sends the results to another module. Through various configurations of modules the user can solve many complicated scientific problems.

The concept of providing monitoring data by J-OCM to a tool required additional software to be written. We had to develop a new package in SCIRun to make TAU/Paravis work with J-OCM. The new set of modules is responsible for: programming the J-OCM monitoring system, receiving the events from it, selecting a performance object to be monitored, controlling the execution of application, and processing the data which are passed to TAU-Paravis.

4 Design of Monitoring Environment

The new monitoring environment can be divided into two parts: a monitoring subsystem and an SCIRun compliant tool. The tool is responsible for the programming monitoring activities, handling monitor events, measuring and visualizing. The tool runs inside SCIRun and consists of several packages. Its most important components are the TAU package and JOCM package. We focused on the second one. The JOCM package contains several SCIRun modules, data types and ports definitions. They provide access to the monitor and produce a TAU compatible data structure on the output. The structure is a 3-D matrix. Having passed such matrices TAU can be used to provide matrix specific operations on the data.

Fig. 2 presents an overview of the system architecture. J-OCM and SCIRun are two separate systems which communicate using the J-OMIS interface. SCIRun *access modules* execute J-OCM specific services and handle responses and events. In principle, it is possible to use any monitoring system other than J-OCM, which complies to the J-OMIS interface. A monitoring system can be developed or evolve separately without influencing the whole environment.

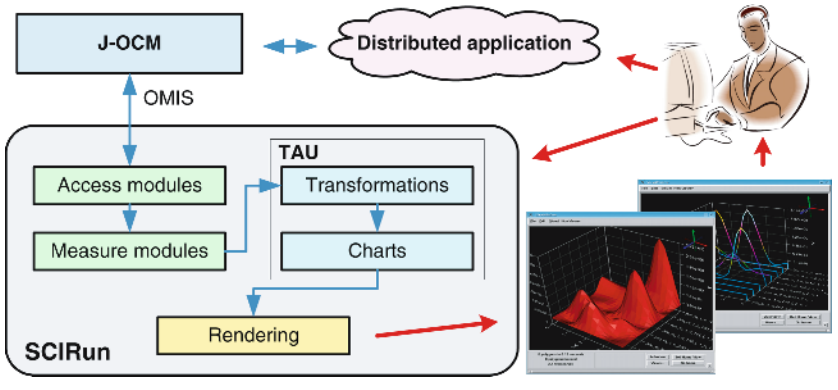


Fig. 2. System architecture

In the JOCM package we define the following kinds of objects which can be monitored: *Node* (physical machine), *JVM* (Java Virtual Machine), *Class* (Java class), *Method* (class method), and *Thread* (Java thread)

Each performance object has its representation within J-OCM in form of *token*. Objects form a hierarchy with nodes on the top and methods and threads at the bottom. Each performance object is associated with a specific SCIRun access module. The monitoring environment can be easily configured by placing modules and connecting them according to the objects hierarchy. Each access module can be attached to one or more J-OCM tokens. It is possible to attach multiple modules to the same token.

The access modules perform monitor programming, but they also handle events from J-OCM. The module on top of the hierarchy receives all event

notifications and passes them down until the event comes to the module which is intended to handle it. Each access module has also a performance output which can be connected to some measurement module's input port.

The second category of modules contains *measurement modules*. They are responsible for building measurements from monitoring events and gathering measurement results. Measurements are connected to some metrics. We have defined the following metrics:

- *method execution time* (aggregate or momentary, inclusive or exclusive, in context of a thread)
- *thread status over time*
- *garbage collector activity* (execution time, released memory size)

The "method execution time" can be aggregated at the class level. In this case, the "class execution time" is a sum of all class methods execution times. The monitoring of thread status provides important information about the activity of a thread. This information can be useful in solving the issues of thread synchronization.

The measurement modules gather performance data and transform it into 3-D matrix structures. These can be passed to TAU modules which perform some additional transformations like aggregation or scaling. The results are being used to produce 3-D charts which are finally rendered by an SCIRun built-in rendering module.

The functionality of the system is much broader that performance visualization only. It can also take control over an application execution. Now this ability is limited to threads only. The user can make use of this feature to change a thread status during the execution, without modifying source code.

5 Web Services Monitoring

The Web Services approach to building distributed applications is getting more and more popular. The advantages of using web-based components are very difficult to overestimate. Such components have well-defined interfaces and can be easily accessed. An issue when using Web Services is their performance, due to the use of XML-based communication protocols.

Debugging and optimizing a single Web service is quite an easy task. Problems occur when Web services start to interact while forming a working web application. In this case the ability to do performance analysis is extremely important. It is still very difficult to find a complete solution for the performance monitoring and visualization of web services.

Following the extension done to J-OCM, we have developed additional modules for accessing the new types of performance objects. These modules are:

- *Web Service access module*
- *Web Service's port access module*
- *Port's operation access module*

The Web Service access module can be connected to the Node access module. In this way the new modules extend the existing modules hierarchy (it is related to the extended J-OCM tokens hierarchy).

We have defined relevant metrics:

- *Resources usage* (CPU, memory)
- *Communication* (throughput, latency, reply time)
- *Requests* (frequency, SOAP message processing time, message size)
- *Run-time specific* (operation execution time)

These metrics define the new measurements which require new modules for gathering the measurement results. The important thing is that the output of these modules is still TAU's 3D matrix. As a result the visualization engine is invariant to changes.

Within this concept, a very important part of the work are extensions to the monitoring system. J-OCM must be able to manage a web service's distribution which differs from that on a cluster of nodes supported by J-OCM. Since nodes can be grouped into sites, the monitoring system architecture must comprise *Service Managers* above Local Monitors. Service Managers are managed by Service Distribution Unit instead of the Node Distribution Unit as it was in the cluster-oriented version of J-OCM.

6 Performance Visualization

The advantages of 3-D visualization over 2-D are obvious. 3-D charts are much more readable and can provide more information at the same time. One of the most important advantages of TAU is the ability to perform such visualization in real time. The user is enabled to see what performance problems occur during an application's execution. The user is enabled to access various performance

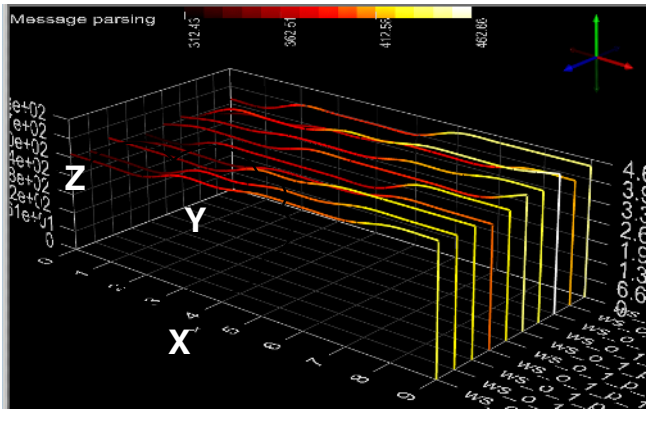


Fig. 3. Summarized message parsing time (X-time; Y-web services; Z-measured value)

data on different objects at the same time. In this case it is much easier to figure out dependencies between performance phenomena (e.g. bottlenecks).

In Fig. 3 we show an example performance visualization session of several similar web services ("Hello world"). The monitoring environment is configured to measure the SOAP messages parsing time. Each "Hello world" Web service is located on a different physical machine. The client application randomly sends requests to the Web services. With the tool we can observe the dynamics of increase in the aggregate parsing time over the execution.

7 Related Work

There are a large number of monitoring tools for Java distributed programs. Commercial tools like AmberPoint Express[1] for .NET platform or open source ones like Apache jMeter[2] for Java provide for the user a lot of useful features. They support advanced visualization and can point at application bottlenecks. However, each of them runs as a client application. They test Web Services by sending requests and counting the response time or number of fails. They can also inspect SOAP packages. However, an issue is that they do not allow the user to get insight into what really happens inside the Web Service.

Our goal is to overcome this constraint and to provide that our approach can point at the part of the Web Service (initialization, request processing, operation invocation or response) responsible for the performance problem. Moreover, the system under discussion supplies more advanced visualization and can be easily extended by new functionality.

8 Conclusion

Performance visualization is important for several reasons. Efficient application programming requires efficient techniques for performance analysis and visualization. When dealing with distributed systems the need in such a functionality is even stronger.

It is still difficult to find comprehensive open source solutions for performance monitoring, analysis and visualization of web services. Most of available tools are limited to the SOAP messages analysis. Our approach is to attach the monitoring tool to the Web service container and obtain from it needed monitoring data.

The system under discussion is open source and offers a functionality required in case of the monitoring of distributed Java applications and also web services implemented in Java. It uses a powerful scientific visualization environment - SCIRun which is widely used due to it's open architecture. As a result, the system's functionality can be easily extended by adding new modules.

As any monitoring system, J-OCM induces some overhead into the monitored application performance, due to the local agents which may do dynamical instrumentation of the monitored application. In other cases, agents use the JVM

[1] [http://www.amberpoint.com/solutions/express.shtml]
[2] [http://jakarta.apache.org/jmeter]

built-it Tool Interface which influences the performance to a small extent. The results of overhead measurements will be presented in the final version of the paper. As well we will provide the results of scalability research.

One of the important issues of the whole environment we will work on in further research is to optimize the resources usage needed by SCIRun (memory, CPUs, efficient graphics accelerator), which is crucial in large distributed applications.

The web page of the monitoring environment is being worked on.

Acknowledgements

This research was partially supported by the KBN grant 4 T11C 032 23.

References

1. M. Gerndt, Automatic performance analysis tools for the Grid, Concurrency and Computation: Pract. Exper, Vol. 17, pp. 99-115, 2005
2. W. Funika, M. Bubak, M.Smętek, and R. Wismüller. An OMIS-based Approach to Monitoring Distributed Java Applications. In Yuen Chung Kwong, editor, Annual Review of Scalable Computing, volume 6, chapter 1. pp. 1-29, World Scientific Publishing Co. and Singapore University Press, 2004.
3. TAU's 3-D Profile Visualizer - ParaVis, University of Oregon, Computer and Information Science
   http://www.cs.uoregon.edu/research/paracomp/tau/tauprofile/dist/paravis/
4. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series. 1997
   http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz
5. Bubak, M., Funika, W., Wismüller, R., Mętel, P., Orłowski. Monitoring of Distributed Java Applications. In: Future Generation Computer Systems, 2003, no. 19, pp. 651-663. Elsevier Publishers, 2003
6. R. Wismüller, J. Trinitis and T. Ludwig: A Universal Infrastructure for the Runtime Monitoring of Parallel and Distributed Applications. In Proc. Euro-Par'98, Southampton, UK, September 1998, LNCS 1470, pp. 173-180. Springer-Verlag, 1998
7. A. D. Malony, S. Shende, and R. Bell, "Online Performance Observation of Large-Scale Parallel Applications", Proc. Parco 2003 Symposium, Elsevier B.V., Sept. 2003.
8. A. D. Malony, S. Shende, R. Bell, K. Li, L. Li, and N. Trebon, "Advances in the TAU Performance System," Chapter, "Performance Analysis and Grid Computing," Kluwer, Norwell, MA, 129-144, 2003.
9. C. Johnson, S. Parker, "The SCIRun Parallel Scientific Computing Problem Solving Environment" Proc. Ninth SIAM Conference on Parallel Processing for Scientific Computing, 1999.
   http://software.sci.utah.edu/scirun.html

Parallel Machine Scheduling of Deteriorating Jobs by Modified Steepest Descent Search*

Stanisław Gawiejnowicz, Wiesław Kurc, and Lidia Pankowska

Adam Mickiewicz University,
Faculty of Mathematics and Computer Science,
Umultowska 87, 61-614 Poznań, Poland
{stgawiej, wkurc, lpankow}@amu.edu.pl

Abstract. In the paper two problems of scheduling time-dependent jobs on parallel machines are considered. In both problems the processing time of each job is described by a linear function of the starting time of the job. The criterion of optimality of a schedule is either the total completion time or the total machine load. First, an equivalence of these two problems is proved. Next, several properties of the problems are shown. Finally, two heuristic algorithms based on the steepest descent search are proposed and results of their experimental evaluation are reported.

Keywords: deteriorating jobs, parallel machine scheduling, total completion time, total machine load; heuristic algorithms, steepest descent search.

1 Introduction

We consider the following two scheduling problems. A set $J = \{J_1, J_2, \ldots, J_n\}$ of time-dependent jobs, which are simultaneously available at time $t_0 = 1$, is to be processed on a set $M = \{M_1, M_2, \ldots, M_m\}$ of identical parallel machines. Jobs are independent and job preemption is not allowed, i.e. once the processing of a job is started, it cannot be interrupted until it is entirely completed.

In the first problem the actual processing time p_i of job J_i, $1 \leq i \leq n$, is a proportional function of the starting time t of the job, $p_i = \alpha_i t$, where $\alpha_i > 0$ is a positive *deterioration rate* of the job and $t \geq t_0$. The objective is to minimize the total completion time, $\sum C_i$, of jobs from the set J.

In the second problem the actual job processing time is a linear function of time, $p_i = 1 + \alpha_i t$, where $\alpha_i > 0$ for $1 \leq i \leq n$. The objective is to minimize the total machine load, $\sum C_{\max}^{(k)} = \sum_{k=1}^{m} \sum_{i=0}^{n_k} \prod_{j=1}^{i} (1 + \alpha_{n_k - j + 1}^{k})$, where n_k is the number of jobs assigned to machine M_k and α_j^k is the deterioration rate of the j-th job assigned to machine M_k, $1 \leq k \leq m$, $1 \leq j \leq n_k$ and $\sum_{k=1}^{m} n_k = n$.

Using the $\alpha|\beta|\gamma$ notation, [4], these problems can be denoted in short as $Pm|p_j = \alpha_j t| \sum C_j$ and $Pm|p_j = 1 + \alpha_j t| \sum C_{\max}^{(k)}$, respectively.

* The research has been partially supported by grant no. 4T11C 03925 of the Ministry of Science and Information Society Technologies of Poland.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 116–123, 2006.

2 Previous Research

In time-dependent scheduling we deal with proportional job processing times, when the processing time of each job is described by proportional function of the starting time of the job, [8]. Nowe we review complexity results from the area.

Mosheiov [8] proved that problem $1|p_j = \alpha_j t|\psi$ is polynomially solvable, if the optimality criterion $\psi \in \{C_{\max}, \sum C_j\}$. Chen [2] and Kononov [7] proved that problem $P2|p_j = \alpha_j t|\sum C_j$ is ordinarily NP-hard. Mosheiov [9] proved that the problem with the total machine load criterion, $P2|p_j = \alpha_j t|\sum C^{(k)}_{\max}$, is ordinarily NP-hard. Kononov [7] and Mosheiov [9, 10] proved several other results concerning scheduling proportionally deteriorating jobs on dedicated machines.

The second group of results concerns lower bounds on the value of $C_{\max}$, $\sum C^{(k)}_{\max}$ or $\sum C_j$ criterion. Let $F^\star(J)$ denote a lower bound on the value of criterion F for job set J. For problem $P2|p_j = \alpha_j t|\sum C_j$, Chen [2] established the following lower bound: $\sum C^\star_j(J) \geq 2\sum_{i=1}^{k}\sqrt{\prod_{j=1}^{2i+r} a_j}$, where $a_j = 1 + \alpha_j$ for $1 \leq j \leq n$, all a_j's are ordered nondecreasingly, $r = 0$ if $n = 2k$ and $r = 1$ if $n = 2k + 1$ for some k. For the problem with arbitrary number of machines and with the total machine load criterion, $Pm|p_j = \alpha_j t|\sum C^{(k)}_{\max}$, Mosheiov [9] suggested the lower bound in the form of $\sum C^{(k)\star}_{\max}(J) \geq m\sqrt[m]{\prod_{j=1}^{n} a_j}$. Chen's lower bound has been tightened by Jeng and Lin [6], who have shown that for problem $Pm|p_j = \alpha_j t|\sum C_j$, $k = \lfloor \frac{n}{m} \rfloor$ and $r = n - km$ there holds the inequality

$$\sum C^\star_j(J) \geq LB_1 = m\sum_{i=1}^{k}\sqrt[m]{\prod_{j=1}^{im+r} a_j} + \sum_{j=1}^{r} a_j. \tag{1}$$

The reader is referred to reviews by Alidaee and Womer [1] and Cheng et al [3] for more details on time-dependent scheduling.

3 Equivalence of Problems $Pm|p_j = \alpha_j t|\sum C_j$ and $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$

We now prove the equivalence of the two problems and some of their properties.

Property 1. In an optimal schedule for problem $Pm|p_j = \alpha_j t|\sum C_j$ jobs assigned to a machine are arranged in the nondecreasing order of deterioration rates and scheduled without idle times.

Proof. Assume that σ is an optimal schedule in which jobs assigned to a machine are not in a nondecreasing order of deterioration rates α_i, $1 \leq i \leq n$. By changing the order into a nondecreasing order, we obtain a schedule σ' such that $\sum C_j(\sigma') \leq \sum C_j(\sigma)$, since in an optimal schedule for a single machine jobs have to be in nondecreasing order of α_i's, [8]. Changing, if necessary, the order of jobs on other machines, we obtain such an optimal schedule that on each machine jobs are arranged in nondecreasing order of α_i's. Since any idle time increases job completion times, the optimal schedule cannot contain idle times. □

Theorem 1. *Let* $\sigma^i = (\alpha^i_1, \alpha^i_2, \ldots, \alpha^i_{n_i})$ *and* $\bar{\sigma}^i = (\alpha^i_{n_i}, \alpha^i_{n_i-1}, \ldots, \alpha^i_1)$ *denote a sequence of jobs assigned to machine* M_i *and the sequence reversed to* σ^i*, respectively, where* α^i_j *denotes deterioration rate of job* J_j *assigned to machine* M_i*,* $1 \leq j \leq n_i$*,* $1 \leq i \leq m$ *and* $\sum_{i=1}^{m} n_i = n$*. Then for every schedule* $\sigma = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ *for problem* $Pm|p_j = \alpha_j t|\sum C_j$ *there exists a corresponding schedule* $\bar{\sigma} = (\bar{\sigma}^1, \bar{\sigma}^2, \ldots, \bar{\sigma}^m)$ *for problem* $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$ *and for every schedule* $\bar{\sigma}$ *for problem* $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$ *there exists a corresponding schedule* σ *for problem* $Pm|p_j = \alpha_j t|\sum C_j$ *such that* $\sum C_j(\sigma) = \sum C^{(k)}_{\max}(\bar{\sigma}) - m$*, provided that both these schedules start at time* $t_0 = 1$*.*

Proof. (Sufficiency) Assume that $t_0 = 1$ and let $\sigma = (\sigma^1, \sigma^2, \ldots, \sigma^m)$ be a schedule for problem $Pm|p_j = \alpha_j t|\sum C_j$, where $\sigma^i = (\alpha^i_1, \alpha^i_2, \ldots, \alpha^i_{n_i})$, $1 \leq i \leq m$. Then we have $\sum C_j(\sigma) = \sum_{k=1}^{m} \sum_{i=1}^{n_k} \prod_{j=1}^{i} a^k_j = \sum_{k=1}^{m} \sum_{i=0}^{n_k} \prod_{j=1}^{i} a^k_j - m = \sum_{k=1}^{m} \sum_{i=0}^{n_k} \prod_{j=1}^{i} b^k_{n_k-j+1} - m = \sum C^{(k)}_{\max}(\bar{\sigma}) - m$, where $a^k_j = 1 + \alpha^k_j$, $b^k_{n_k-j+1} = a^k_j$ and $\bar{\sigma}^i = (\alpha^i_{n_k}, \alpha^i_{n_k-1}, \ldots, \alpha^i_1) = (b^i_1, b^i_2, \ldots, b^i_{n_k})$.

The proof for necessity can be done in an analogous way. □

Property 2. A schedule $\sigma^\star$ is optimal for problem $Pm|p_j = \alpha_j t|\sum C_j$ if and only if the schedule $\bar{\sigma}^\star$ is optimal for problem $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$. Moreover, $\sum C_j(\sigma^\star) = \sum C^{(k)}_{\max}(\bar{\sigma}^\star) - m$.

Proof. Let $\sigma^\star$ be an optimal schedule for problem $Pm|p_j = \alpha_j t|\sum C_j$ and let $\bar{\sigma}^\star$ denote the schedule obtained from $\sigma^\star$ by reversing the order of jobs on each machine. First, note that equality $\sum C_j(\sigma^\star) = \sum C^{(k)}_{\max}(\bar{\sigma}^\star) - m$ must hold, since Theorem 1 concerns any schedule, the optimal one in particular.

Assume now that $\bar{\sigma}^\star$ is not optimal for problem $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$. Hence there must exist a schedule τ such that $\sum C^{(k)}_{\max}(\tau) < \sum C^{(k)}_{\max}(\bar{\sigma}^\star) = \sum C_j(\sigma^\star) + m$. On the other hand, $\sum C^{(k)}_{\max}(\tau) = \sum C_j(\bar{\tau}) + m$. Hence we have that $\sum C_j(\bar{\tau}) < \sum C_j(\sigma^\star)$. This, however, is impossible since $\sigma^\star$ is optimal with respect to $\sum C_j$ criterion. A contradiction. □

Throughout the paper the problems for which the equivalence described in Theorem 1 holds, will be called *equivalent* problems.

Property 3. Problems $1|p_j = \alpha_j t|\sum C_j$ and $1|p_j = 1 + \alpha_j t|C_{\max}$ are equivalent.

Proof. Let $m = 1$. Then, by definition of the total machine load, we have $\sum C^{(k)}_{\max} = \sum_{k=1}^{m} C_{\max} \equiv C_{\max}$. Applying Theorem 1, the result follows. □

Property 4. In optimal schedule for problem $Pm|p_j = 1 + \alpha_j t|\sum C^{(k)}_{\max}$, jobs assigned to a machine are arranged in a nonincreasing order of deterioration rates and scheduled without idle times.

Proof. By Property 2, in any optimal schedule for $Pm|p_j = 1+\alpha_j t|\sum C^{(k)}_{\max}$ jobs are scheduled in the reversed order, comparing to the order of jobs in the optimal schedule for $Pm|p_j = \alpha_j t|\sum C_j$. Applying Property 1, the result follows. □

Property 5. The optimal value $\sum C_{\max}^{(k)\star}$ of the total machine load for problem $Pm|p_j = 1+\alpha_j t|\sum C_{\max}^{(k)}$ is no less than $LB_2 = m\sum_{i=1}^{k} \sqrt[m]{\prod_{j=1}^{im+r} a_j} + \sum_{j=1}^{r} a_j + m$, if $t_0 = 1$, $k = \lfloor \frac{n}{m} \rfloor$ and $r = n - km$.

Proof. Let τ be any schedule for problem $Pm|p_j = 1 + \alpha_j t|\sum C_{\max}^{(k)}$. Then by Theorem 1 and formula (1) for schedule $\bar{\tau}$ we have $LB_1 \leq \sum C_j(\bar{\tau}) = \sum C_{\max}^{(k)}(\tau) - m$. Hence, for any schedule τ there holds inequality $LB_1 + m = LB_2 \leq \sum C_{\max}^{(k)}(\tau)$. □

Theorem 2. *If $t_0 = 1$, problem $Pm|p_j = 1 + \alpha_j t|\sum C_{\max}^{(k)}$ is NP-hard in the ordinary sense, even if $m = 2$.*

Proof. First, note that any instance of problem $P2|p_j = \alpha_j t|\sum C_j$ is an instance of problem $P2|p_j = 1 + \alpha_j t|\sum C_{\max}^{(k)}$, too. Second, by Theorem 1, if $t_0 = 1$ then problem $P2|p_j = \alpha_j t|\sum C_j$ is equivalent to problem $P2|p_j = 1 + \alpha_j t|\sum C_{\max}^{(k)}$. This means that for each schedule σ for the first problem there exists a schedule $\bar{\sigma}$ for the second problem such that $\sum C_j(\sigma) = \sum C_{\max}^{(k)}(\bar{\sigma}) - m$ and vice versa. Thus, since problem $P2|p_j = \alpha_j t|\sum C_j$ is NP-hard in the ordinary sense ([2, 7]), problem $P2|p_j = 1 + \alpha_j t|\sum C_{\max}^{(k)}$ is NP-hard in the ordinary sense as well. □

4 The Modified Steepest Descent Search Heuristics

In the section we formulate two heuristics based on the steepest descent search.

4.1 Previously Known Heuristic Algorithms

For problem $1|p_j = \alpha_j t|\sum C_j$ Mosheiov, [8], proposed the *SGR* (*Smallest Growth Rate* first) rule that sequences the jobs in a nondecreasing order of deterioration rates and leads to the minimum total completion time. This algorithm has been also proposed by Mosheiov [9] for the two-machine case, $P2|p_j = \alpha_j t|\sum C_j$.

ALGORITHM *SGR*

Input: sequence $a = (a_1, a_2, \ldots, a_n)$, where $a_j = 1 + \alpha_j$ for $1 \leq j \leq n$
Output: sub-optimal schedule $\sigma = (\sigma^1, \sigma^2)$

Step 1: Re-index jobs in nonincreasing order of a_j; set $t_{M_1} = t_{M_2} = t_0$;
Step 2: Let k be the index of job with the smallest index. If $t_{M_1} \leq t_{M_2}$, then assign job J_k to machine M_1 and set $t_{M_1} = (1 + \alpha_k)t_{M_1}$; else assign job J_k to machine M_2 and set $t_{M_2} = (1 + \alpha_k)t_{M_2}$. Continue until all jobs are assigned.

For the total machine load problem, $Pm|p_j = \alpha_j t|\sum C_{\max}^{(k)}$, Mosheiov [9] proposed the *LDR* (*Largest Deterioration Rate* first) heuristic. He also proved that if $n \to \infty$, then the performance ratio of the LDR heuristic is bounded and is asymptotically close to one. For problem $Pm|p_j = \alpha_j t|\sum C_j$, Jeng and Lin [6]

proposed the heuristic *RLDR* (*Reverse LDR*), adopted from LDR (in fact, the LDR algorithm is composed of the first two steps of the RLDR algorithm).

ALGORITHM *RLDR*

Input: sequence $a = (a_1, a_2, \ldots, a_n)$, where $a_j = 1 + \alpha_j$ for $1 \leq j \leq n$
Output: sub-optimal schedule $\sigma = (\sigma^1, \sigma^2, \ldots, \sigma^m)$

Step 1: Re-index jobs in nonincreasing order of a_j;
Step 2: Assign, one at a time, jobs $J_1, J_2, \ldots, J_n$ to the machine with the smallest completion time;
Step 3: Reverse the job sequence on each machine, i.e. put the jobs in nondecreasing order of a_j.

Note that all algorithms mentioned above have $O(n \log n)$ time complexity, since the most time-consuming step is the re-indexing of jobs. Computational experiments have also shown (see Jeng and Lin [6] for details) that the RLDR algorithm gives better results than the SGR algorithm.

4.2 Auxiliary Result

Now we pass to the result which will allow us to formulate a new polynomial-time heuristic algorithm for the problems under consideration. (We omit the proof.)

Let jobs $J_{1,p}, J_{2,p}, \ldots, J_{n_p,p}$ with deterioration rates $a_1, a_2, \ldots, a_{n_p}$ be scheduled on machine $M_p, 1 \leq p \leq m$, and let jobs $J_{1,q}, J_{2,q}, \ldots, J_{n_q,q}$ with deterioration rates $b_1, b_2, \ldots, b_{n_q}$ be scheduled on machine $M_q, 1 \leq q \leq m$, $q \neq p$.

Theorem 3. *Consider schedule σ for problem $Pm|p_j = \alpha_j t|\sum C_j$ and schedule τ obtained by moving job $J_{k,q}$ from machine M_q and inserting it after job $J_{r,p}$ on machine M_p, $1 \leq p, q \leq m$, $p \neq q$, $0 \leq r \leq n_p$, $1 \leq k \leq n_q$. Then the difference between total completion time of σ and total completion time of τ is equal to*

$$\Delta_1(\sigma, \tau) = \sum C_j(\sigma) - \sum C_j(\tau) = \left(1 + (b_k - 1) \sum_{j=k}^{n_q} \prod_{i=k+1}^{j} b_i\right) \prod_{i=1}^{k-1} b_i \quad (2)$$
$$- \left(1 + (b_k - 1) \sum_{j=r}^{n_p} \prod_{i=r+1}^{j} a_i\right) \prod_{i=1}^{r} a_i.$$

From Theorem 3 it follows that the necessary condition for optimality of a schedule $\sigma^\star$ is satisfying inequality $\Delta_1(\sigma^\star, \tau) \leq 0$ for any $\tau \neq \sigma^\star$.

4.3 Algorithm MSD_1

Recall that the steepest descent search in each iteration evaluates all *moves* which can be done from the *neighbourhood* of the currently best *solution* and chooses the one which minimizes the criterion function.

Let J_{a_i} denote the job corresponding to deterioration rate a_i. Let variables σ_0, σ_{best} and σ_{last} denote the starting schedule, the last best and the last but one best schedule, respectively, and let variable τ denote the best schedule in set $\mathcal{S}$ of solutions constructed at some iteration of the algorithm. The proposed algorithm MSD_1 (*Modified Steepest Descent* search) can be formulated as follows (note that for simplicity we use an indentation for denoting complex instructions).

ALGORITHM MSD_1

Input: sequence $a = (a_1, a_2, \ldots, a_n)$, where $a_j = 1 + \alpha_j$ for $1 \leq j \leq n$
Output: sub-optimal schedule $\sigma_{best} = (\sigma^1_{best}, \sigma^2_{best}, \ldots, \sigma^m_{best})$

Step 1: { `Construction of the starting schedule` σ_0}
Arrange all jobs in nonincreasing order of a_i;
Assign $m-1$ jobs with greatest a_i to machines $M_2, M_3, \ldots, M_m$;
Assign remaining $n-m$ jobs to machine M_1;
$\sigma_{best} := \sigma_0$;
repeat
$\sigma_{last} := \sigma_{best}$;
Step 2: { `Construction of set` $\mathcal{S}$ `of current solutions` }
$\mathcal{S} := \emptyset$;
for jobs assigned to machine M_1 **do**
Choose a job J_{a_i};
for $\mathcal{M}$ **in** $M_2, M_3, \ldots, M_m$ **do**
Construct schedule σ' by moving job J_{a_i} to machine $\mathcal{M}$;
$\mathcal{S} := \mathcal{S} \cup \sigma'$;
Step 3: { `Selection of the best schedule in` $\mathcal{S}$}
Choose from $\mathcal{S}$ the schedule $\tau = \arg\max\{\Delta_1(\sigma_{last}, \sigma') : \sigma' \in \mathcal{S}\}$;
if $(\Delta_1(\sigma_{last}, \tau) > 0)$ **then** $\sigma_{best} := \tau$;
until $(\Delta_1(\sigma_{best}, \sigma_{last}) = 0)$.

The time complexity of the MSD_1 algorithm depends on the number of iterations of **repeat-until** loop, which is $O(nm)$, and the cost of checking the condition $\Delta_1(\sigma, \tau) > 0$ from formula (2) for $\sigma = \sigma_{last}$ and $\tau = \sigma'$. Since this latter cost is $O(n)$, the overall time complexity of the algorithm is $O(n^2 m) \equiv O(n^2)$ for fixed m.

4.4 Algorithm MSD_2

Assume that a schedule is given. Then we can improve this schedule by successive moving of jobs between machines in order to find such an assignment of jobs which gives the smaller total completion time than the initial one.

Introduce some notation which we will use. Let $\tau = \sigma(J_i \leftrightarrow J_k)$ denote schedule σ in which jobs J_i and J_k have been mutually replaced. Let $ind(\sigma, J_i)$ denote the index of a machine to which job J_i has been assigned in schedule σ. Let $\Delta_2(\sigma, \tau)$ denote the difference $\sum C_j(\sigma) - \sum C_j(\tau)$. Then the MSD_1 algorithm using the above idea of iterative improvement can be formulated as follows.

ALGORITHM MSD_2

Input: sequence $a = (a_1, a_2, \ldots, a_n)$, where $a_j = 1 + \alpha_j$ for $1 \leq j \leq n$
Output: sub-optimal schedule $\sigma_{best} = (\sigma^1_{best}, \sigma^2_{best}, \ldots, \sigma^m_{best})$

Step 1: { `Construction of the starting schedule` σ_0}
$\sigma_0 := MSD_1(a)$; $k := 0$; $\sigma_{best} := \sigma_0$;

Step 2: { `Iterative improvement of the present schedule` }
repeat
$k := k + 1$; $\sigma_{last} := \sigma_{best}$;
for $i := n$ **downto** 2 **do**
for $k := i - 1$ **downto** 1 **do**
if $(ind(\sigma_{last}, J_i) \neq ind(\sigma_{last}, J_k))$ **then** $\tau := \sigma_{last}(J_i \leftrightarrow J_k)$;
if $(\Delta_2(\sigma_{last}, \tau) > 0)$ **then** $\sigma_{best} := \tau$;
until $((\Delta_2(\sigma_{best}, \sigma_{last}) = 0)$ or $(k > n))$.

The time complexity of the MSD_2 algorithm is $O(n^3)$, since in the worst case we have to check n times $O(n^2)$ possibilities of a mutual change of two jobs.

5 The Computational Experiment Results

In order to evaluate the quality of schedules generated by algorithms RLDR, MSD_1 and MSD_2 we conducted a computational experiment for $m = 3$ machines. The rates a_i were generated randomly. The results are presented in Table 1 and Table 2. Each value in these tables is an average of results for 10 instances.

Columns $\texttt{RLDR}^*$, $\texttt{MSD}_1^*$ and $\texttt{MSD}_2^*$ include the average relative error of the total completion time for algorithms RLDR, MSD_1 and MSD_2, respectively, calculated with respect to the optimal value of the total completion time, $\sum C_j$. Columns $\texttt{RLDR}^\circ$, $\texttt{MSD}_1^\circ$ and $\texttt{MSD}_2^\circ$ include the average relative error of the total completion time for algorithms RLDR, MSD_1 and MSD_2, respectively, calculated with respect to lower bound (1) of $\sum C_j$.

Table 1 and Table 2 show that MSD_2 is better than $RLDR$ for $a_i \in (2, 99)$, while for $a_i \in (1, 2)$ algorithms MSD_2 and $RLDR$ are comparable. The

Table 1. Results of Computational Experiment for $a_i \in (2, 99)$

n	RLDR^*	RLDR°	MSD_1^*	MSD_1°	MSD_2^*	MSD_2°
6	0.0	0.186729	0.167711	0.374054	0.0	0.186729
8	0.267105	0.639685	0.167173	0.493273	0.0	0.293706
10	0.366406	0.384822	0.121466	0.127353	0.016173	0.031644
12	0.116080	0.115118	0.459128	0.444614	0.003993	0.014459
14	-	0.476927	-	0.206961	-	0.090330
16	-	0.354809	-	0.237446	-	0.012126
18	-	0.052520	-	0.344081	-	0.054585
20	-	0.475177	-	0.161075	-	0.031898

Table 2. Results of Computational Experiment for $a_i \in (1, 2)$

n	RLDR*	RLDR$^\circ$	MSD$_1^*$	MSD$_1^\circ$	MSD$_2^*$	MSD$_2^\circ$
5	0.0	0.003121	0.0	0.003121	0.0	0.003121
6	0.0	0.002508	0.000693	0.003204	0.0	0.002508
8	0.001603	0.005034	0.004348	0.007798	0.0	0.003425
10	0.001520	0.002659	0.014319	0.015473	0.000026	0.001163
12	0.001170	0.001809	0.020410	0.021059	0.003459	0.004098
14	-	0.002801	-	0.025815	-	0.005598
16	-	0.002348	-	0.031094	-	0.001261
18	-	0.001272	-	0.044117	-	0.013159
20	-	0.003101	-	0.049956	-	0.004320

quality of schedules generated by algorithm MSD_1, in comparison to $RLDR$ algorithm, is an open question and it needs further experiments. Notice also that by Theorem 1 algorithms proposed for $Pm|p_j = \alpha_j t| \sum C_j$ problem can be used for $Pm|p_j = 1 + \alpha_j t| \sum C_{\max}^{(k)}$ problem, too. Finally, it seems that MSD_2 is a promising alternative for such metaheuristics as simulated annealing (see Hindi and Mhlanga [5]) which are more time-consuming and hard to implement.

References

1. B. Alidaee and N.K. Womer, Scheduling with time dependent processing times: Review and extensions. *J. Optl Res. Soc.* **50** (1999), 711-720.
2. Z.-L. Chen, Parallel machine scheduling with time dependent processing times. *Discr. Appl. Math.* **70** (1996), 81–93. Erratum: *Discr. Appl. Math.* **75** (1996), 103.
3. T.C.E. Cheng, Q. Ding and B.M.T. Lin: A concise survey of scheduling with time-dependent processing times. *Euro. J. Optl Res.* **152** (2004), 1–13.
4. R.E. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discr. Math.* **4** (1979), 287-326.
5. K.S. Hindi and S. Mhlanga, Scheduling linearly deteriorating jobs on parallel machines: A simulated annealing approach. *Prod. Planning and Control* **12** (2001), 76–80.
6. A.A-K. Jeng and B.M-T. Lin, Parallel-machine scheduling with deteriorating jobs to minimize total completion time, unpublished manuscript, April 2004.
7. A. Kononov, Combinatorial complexity of scheduling jobs with simple linear deterioration. *Diskret. Anal. Issled. Oper.* **3** (1996), 15–32 (in Russian).
8. G. Mosheiov, Scheduling jobs under simple linear deterioration. *Computers and Oper. Res.* **21** (1994), 653–659.
9. G. Mosheiov, Multi-machine scheduling with linear deterioration, *Infor* **36** (1998), 205–214.
10. G. Mosheiov, Complexity analysis of job-scheduling with deteriorating jobs, *Discr. Appl. Math.* **117** (2002), 195–209.

A Study on Load Imbalance in Parallel Hypermatrix Multiplication Using OpenMP*

José R. Herrero and Juan J. Navarro

Computer Architecture Dept., Univ. Politècnica de Catalunya,
Barcelona, Spain
{josepr, juanjo}@ac.upc.edu

Abstract. In this paper we present our work on the the parallelization of a matrix multiplication code based on the hypermatrix data structure. We have used OpenMP for the parallelization. We have added OpenMP directives to a few loops and experimented with several features available with OpenMP in the Intel Fortran Compiler: scheduling algorithms, chunk sizes and nested parallelism. We found that the load imbalance introduced by the hypermatrix structure could not be solved by any of those OpenMP features.

1 Introduction

We have used a Hypermatrix data structure [1, 2] in sequential linear algebra codes. We could obtain efficient implementations in both sparse and dense codes. Now, we are interested in the parallelization of our dense codes. This data structure, however, presents difficulties when work has to be distributed amongst several processors. Namely, the difficulty to balance the load evenly. In this paper we present the work we have done to produce a hypermatrix multiplication based on OpenMP directives. We wanted to know whether some of the features available in OpenMP could surmount the intrinsic difficulties of parallel codes based on the Hypermatrix data structure. We have chosen matrix multiplication because it is highly parallelizable. Also, it is a very important operation since it appears as a basic kernel in many scientific applications. For this reason it has been studied extensively [3, 4, 5].

1.1 OpenMP

OpenMP [6] provides a set of directives and environment variables to express and control parallelism in the execution of a program. The user can choose the scheduling algorithm. When a *static* scheduling algorithm is used, the distribution of iterations to threads is done before the execution of any of them. When a *dynamic* scheduling algorithm is used, the next piece of work for a thread is assigned when it is needed. It is taken from the remaining operations due. There

* This work was supported by the Ministerio de Ciencia y Tecnología of Spain (TIN2004-07739-C02-01).

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 124–131, 2006.

is a default value for the number of iterations assigned to each processor which can be changed by the user explicitly. We refer to the *chunk* size. The default for the static scheduling is to split the work in as many parts as the number of threads defined. The default for the dynamic scheduling is to take one iteration each time.

Several nested loops can be parallelized. OpenMP permits this fact with a feature known as *nested parallelism*. When nested parallelism is activated, parallel constructs can be used within other parallel constructs.

In this paper we have used OpenMP for the parallelization of a matrix multiplication code based on the hypermatrix data structure.

1.2 Hypermatrix Data Structure

Our application uses a data structure based on a hypermatrix (HM) scheme [1, 2], in which a matrix is partitioned recursively into blocks of different sizes. A commercial package known as PERMAS uses the hypermatrix structure for solving very large systems of equations [7]. It can solve very large systems out-of-core and can work in parallel. This approach is also related to a variety of recursive/nonlinear data layouts which have been explored elsewhere for both regular [8, 9, 10, 11] and irregular [12] applications.

The HM structure consists of N levels of submatrices. In order to have a simple HM data structure which is easy to traverse we have chosen to have blocks at each level which are multiples of the lower levels. The top *N-1* levels hold pointer matrices which point to the next lower level submatrices. Only the last (bottom) level holds data matrices. Data matrices are stored as dense matrices and operated on as such. Hypermatrices can be seen as a generalization of quadtrees. The latter partition each matrix precisely into four submatrices [13].

Null pointers in pointer matrices indicate that the corresponding submatrix does not have any non-zero elements and is therefore unnecessary. This is useful when matrices are sparse. Figure 1 shows a sparse matrix and a simple example of a corresponding hypermatrix with 2 levels of pointers.

In the past, we have been working on the sparse Cholesky factorization based on the hypermatrix data structure. We created efficient routines which operate on small matrices. By small we mean matrices which fit in cache. We grouped such routines in a library called the Small Matrix Library (SML). Information

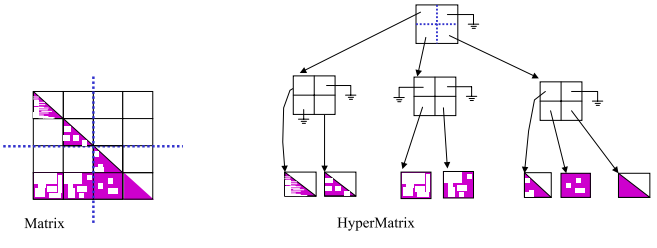


Fig. 1. A sparse matrix and a corresponding hypermatrix

about the creation of the SML can be found in [14]. Further details on the application of SML to sparse hypermatrix Cholesky can be found in [15].

The hypermatrix data structure, however, can also be used for dense matrix computations. Recently, we have applied a similar approach to work on dense matrices. When working on dense matrices in-core, two levels of pointers are enough. To work efficiently on dense matrices we have extended our SML with routines which work with larger sizes than the ones used for the sparse codes. On MIPS, ALPHA and Itanium2 platforms we could obtain very efficient codes for the matrix multiplication.

Now, we are interested in the efficient execution on multiprocessor machines. The hypermatrix data structure, however, presents some difficulties when parallel code is developed. Namely, the partitioning of the matrix is done when the data structure is set. Each pointer in the upper pointer matrix level maps a part of the matrix. If the dimension of such matrix is not a multiple of the number of processors used then the load is not distributed evenly amongst them.

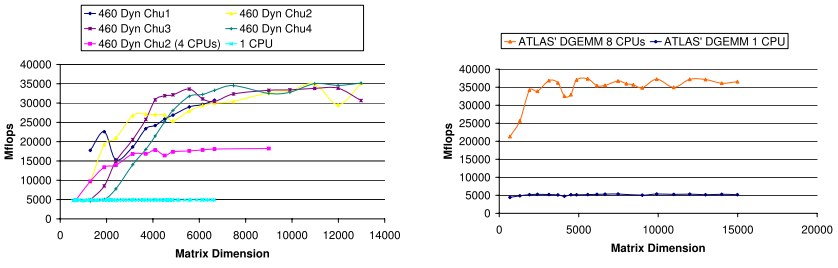
We have started with the study of the hypermatrix multiplication operation, which is very regular and has a high potential for parallelism. We have added OpenMP directives to a few loops and experimented with several features available with OpenMP in the Intel Fortran Compiler: scheduling algorithms, chunk sizes and nested parallelism. We anticipate than none of these features was completely successful for the efficient parallelization of our code.

2 Parallel Dense Hypermatrix Multiplication

The target machine was a 8-way SMP with Intel Itanium2 processors running at 1.5 GHz. The theoretical peak of this machine is 48 Gflops. The Itanium2 has three levels of cache. In the first level it has separate instruction and data caches with 16 Kbytes each. Then, it also has a 256 Kbytes L2 cache and an off-chip L3 cache with possible sizes ranging from 1.5 up to 9 MB.

We have experimented with four and eight processors. In this section we will discuss the results obtained. Our preliminary study on four CPUs provides a speed-up of 3.7 for medium to large matrices. The best combination was that where the two outermost loops were parallelized using nested parallelism and a dynamic scheduling algorithm was used where the chunk size equaled 2. Figure 2a shows the performance of our hypermatrix multiplication code on four processors for the $C = C - A * B^T$ operation for both the sequential and parallel versions of our code. We have used an upper block size of size 460×460, i.e. each upper level pointer maps a block of such size. We then tried the same approach on eight processors. The same figure 2a shows the performance obtained on eight processors with a dynamic scheduling strategy, with the two outermost loops parallelized using nested parallelism. Several chunk sizes were used.

We observe that for small matrix dimensions small chunk sizes provide better results. However, as the matrix gets bigger, larger chunk values can be more effective. This is due to the reduction in the overhead which occurs when one thread searches for new work. Giving a thread more work at once reduces the



a) b)

Fig. 2. a) Two parallel loops with dynamic scheduling and several chunk sizes on 8, 4 and 1 processors. **b)** Performance of ATLAS' DGEMM.

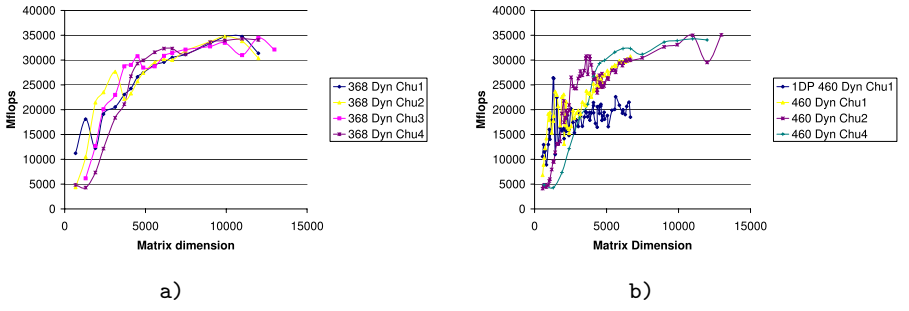
number of times this needs to be done. Also, the memory hierarchy can be better used since contiguous blocks corresponding to consecutive iterations can be reused in the cache. It is important to note that a certain chunk value is effective only when it keeps a good load balancing. Since the loops we have parallelized are the outer loops, they correspond to the upper level pointer matrix. The chunk size times the number of processors should divide the upper level matrix dimension evenly. Otherwise, load imbalance occurs and the performance drops.

We wanted to compare our results to those of ATLAS [3]. Figure 2b shows the performance of the sequential and parallel (on eight processors) versions of ATLAS matrix multiplication routine DGEMM. Their code, starting with the sequential version, outperforms ours. We must note, however, that on this machine ATLAS uses a hand-tuned kernel. The interesting point here comes from the fact that their code achieves high speed-ups sooner than our code. For some large matrix dimensions the speed-up we obtain is similar to theirs (around 7.0). However, for smaller matrices our speed-up is considerably lower. This is due to the load imbalance mentioned above. We have revisited our code and tried several variants aiming to improve its performance, specially when working on smaller matrices.

2.1 Reducing the Block Size

By default we have been using an upper block size of 460×460, i.e. each upper level pointer maps a block of such size. However, we have also reduced the size of the block to 368×368. Figure 3a shows the performance obtained with dynamic scheduling and nested parallelism for this block size. Results are similar to those obtained with our default block size of 460×460.

For the upper levels we have also tried other multiples of the lower levels close to the value $\sqrt{C/2}$, where C is the cache size [16]. Figure 5b shows the performance obtained with several sizes. To simplify the comparison, the maximum value obtained for all chunk sizes for a given block size are presented. Results are similar for all of them. We must note that the smaller block size 276×276 provides the worst performance for larger matrices. This size does



a) b)

Fig. 3. a) Two parallel loops with dynamic scheduling: smaller blocks. **b)** Three loops parallelized (one in the level of pointers to data).

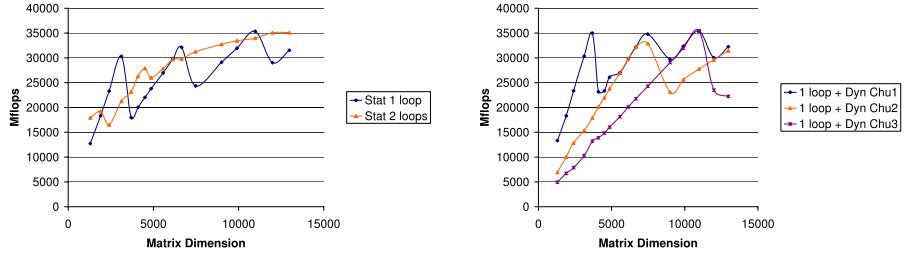
not use the memory hierarchy so effectively. Also, there is more overhead in the parallelization.

2.2 Parallel Loop in Level of Pointers to Data

We have tried another code which parallelizes a third loop in addition to the outermost two loops. This loop is the outermost loop in the level of pointers to data. Figure 3b compares its results to those shown in figure 2a. This code only gets better performance for a few small matrices. For larger matrices, this code does not offer any advantages. The granularity of this third loop is too small and the overhead of the parallelization outweighs any possible advantages.

2.3 Static Scheduling

Figure 4a shows the results obtained with a static scheduling. Results with and without nested parallelism are shown. When only the outermost loop is parallelized we get a saw shape curve. The peaks correspond to sizes which get a perfect partitioning of the hypermatrix, i.e. with a number of pointers in the upper matrix which is a multiple of the number of processors. The use of nested



a) b)

Fig. 4. a) One and two parallel loops with static scheduling. **b)** Parallel outer loop with dynamic scheduling and several chunk sizes.

parallelism introduces some overhead. However, it improves the performance for matrix sizes which are not multiples of the number of CPUs. The performance obtained in both cases is in general worse than that presented in figure 2.

2.4 Dynamic Scheduling with Only 1 Parallel Loop

Figure 4b shows the results obtained when only the outermost loop is parallelized. A dynamic scheduling is used in this case. Again, we get a saw shaped curve. It is quite obvious that larger chunk sizes suffer from load imbalance more often.

2.5 Combining Static and Dynamic Scheduling Algorithms

We have scheduled the outermost loop using a static scheduling and the second outermost loop using a dynamic scheduling. Figure 5a shows the performance obtained. The performance obtained is similar to the one obtained when both loops are scheduled using a dynamic scheduling algorithm as shown in figure 5b.

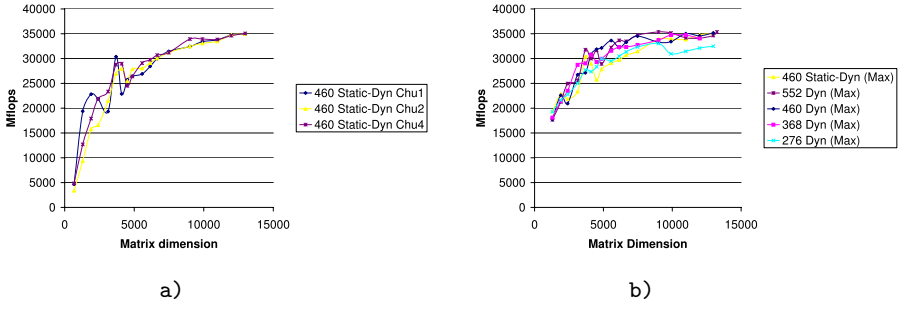


Fig. 5. a) Static scheduling of outermost loop and dynamic scheduling of next inner loop. **b)** Maximum values obtained for each block size and scheduling algorithm.

2.6 Perfect Matrix Partitioning

Figure 6 shows the results obtained when the matrix has been partitioned in a number of parts which is multiple of the number of threads. All partitions have the same size: each upper level pointer maps blocks of 460×460. Thus, a dimension of 3680 produces a hypermatrix with eight pointers in the upper level. The number of pointers in the upper level corresponding to the other three matrix dimensions in the figure are 16, 24 and 32 respectively. All of them are examples where the load can be easily balanced amongst the processors.

These results allow us to study the overhead of each strategy. We use *nest* to identify the use of nested parallelism. A value of 0 means nested parallelism is not allowed while a value of 1 means the opposite. Label *sch* corresponds to the scheduling algorithm used. A value of 0 denotes static scheduling. A value of 1 is used to indicate dynamic scheduling. We use *chu* to specify the chunk size. A value of 0 is used to signify the default value for a given scheduling strategy.

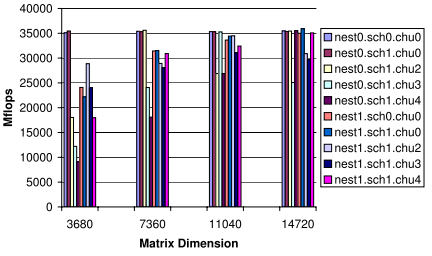


Fig. 6. Experiments with different OpenMP features when the hypermatrix is partitioned for perfect load balancing

On these perfectly partitioned hypermatrices we can observe that, when the matrices are small, the only way to get good speed-ups is via simple strategies: parallelizing only the outermost loop with either static or dynamic scheduling. As the matrices get large there are more strategies which provide good speed-ups. However, in any case it is important to use a chunk size which allows for a good load balancing. The result of dividing the dimension of the upper level pointer matrix by the number of threads must be a multiple of the chunk size.

In a few occasions, a chunk size larger than the default for the dynamic scheduling strategy (which defaults to 1) can improve slightly the performance of our matrix multiplication. This is due to the reduction of the overhead which occurs when one thread takes several iterations at once instead of taking one iteration each time. Also, better use of the memory hierarchy results when one thread reckons several contiguous blocks corresponding to consecutive iterations.

Nested parallelism is not really effective in such situations. It cannot provide any advantages. Instead, it introduces an additional overhead with the creation of the inner parallel construct.

3 Conclusions

We conclude that the best way to parallelize our application is by means of an adequate partitioning of the matrix. If this is possible, a simple scheduling strategy where just the outermost loop is parallelized turns out to be the best solution. Both static and dynamic scheduling algorithms work well and perform in a similar manner.

When data cannot be partitioned adequately we can take advantage of nested parallelism. Despite its overhead, it offers the advantage of being able to open new parallel sections which can employ otherwise idle processors. The resulting performance curves are smoother than the saw shaped curves which result from those cases where only the outer loop was parallelized.

We have conducted experiments with eight processors and found some load imbalance in those cases where the dimension of the matrix in the upper pointer level is low and is not multiple of the number of processors used. Thus, smaller matrices suffer from load imbalance as the number of processors grow. This can

limit the effectivity of parallel codes based on the hypermatrix scheme. In the future, we plan to replace the hypermatrix data structure in our algorithms which deal with dense matrices. We plan to use a plain storage of the data submatrices which can be accessed with a simple indexing scheme. We believe that in this way we can still use our routines which deal with small submatrices and, at the same time, can split the work amongst all processors more effectively.

References

1. Fuchs, G., Roy, J., Schrem, E.: Hypermatrix solution of large sets of symmetric positive-definite linear equations. Comp. Meth. Appl. Mech. Eng. **1** (1972) 197–216
2. Noor, A., Voigt, S.: Hypermatrix scheme for the STAR–100 computer. Comp. & Struct. **5** (1975) 287–296
3. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: Supercomputing '98, IEEE Computer Society (1998) 211–217
4. Choi, J., Dongarra, J., Pozo, R., Walker, D.: ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers. In: Proc. Fourth Symposium on the Frontiers of Massively Parallel Computation, ACM Press (1992) 120–127
5. Chatterjee, S., Lebeck, A.R., Patnala, P.K., Thottethodi, M.: Recursive array layouts and fast parallel matrix multiplication. In: Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures, ACM Press (1999) 222–231
6. OpenMP: (URL) http://www.openmp.org.
7. Ast, M., Fischer, R., Manz, H., Schulz, U.: PERMAS: User's reference manual, INTES publication no. 450, rev.d (1997)
8. Chatterjee, S., Jain, V.V., Lebeck, A.R., Mundhra, S., Thottethodi, M.: Nonlinear array layouts for hierarchical memory systems. In: Proceedings of the 13th international conference on Supercomputing, ACM Press (1999) 444–453
9. Frens, J.D., Wise, D.S.: Auto-blocking matrix multiplication, or tracking BLAS3 performance from source code. Proc. 6th ACM SIGPLAN Symp. on Principles and Practice of Parallel Program., SIGPLAN Not. **32**(7) (1997) 206–216
10. Valsalam, V., Skjellum, A.: A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels. Concurrency and Computation: Practice and Experience **14**(10) (2002) 805–839
11. Wise, D.S.: Ahnentafel indexing into Morton-ordered arrays, or matrix locality for free. In: Euro-Par 2000,LNCS1900. (2000) 774–783
12. Mellor-Crummey, J., Whalley, D., Kennedy, K.: Improving memory hierarchy performance for irregular applications. In: Proceedings of the 13th international conference on Supercomputing, ACM Press (1999) 425–433
13. Wise, D.S.: Representing matrices as quadtrees for parallel processors. Information Processing Letters **20**(4) (1985) 195–199
14. Herrero, J.R., Navarro, J.J.: Automatic benchmarking and optimization of codes: an experience with numerical kernels. In: Proceedings of the 2003 International Conference on Software Engineering Research and Practice, CSREA Press (2003) 701–706
15. Herrero, J.R., Navarro, J.J.: Improving Performance of Hypermatrix Cholesky Factorization. In: Euro-Par 2003,LNCS2790, Springer-Verlag (2003) 461–469
16. Lam, M., Rothberg, E., Wolf, M.: The cache performance and optimizations of blocked algorithms. In: Proceedings of ASPLOS'91. (1991) 67–74

Common Due Window Assignment in Parallel Processor Scheduling Problem with Nonlinear Penalty Functions

Adam Janiak and Marcin Winczaszek

Wroclaw University of Technology, Institute of Engineering Cybernetics, Janiszewskiego 11-17, 50-372 Wrocław, Poland

Abstract. In this paper we study a scheduling problem with earliness and tardiness penalties. The objective is to find a sequence of jobs on parallel identical processors and a common due window. The beginning and the end of the due window are decision variables but the size of the due window is constrained from above and below. A job incurs penalty if it completes before or after the due window. A penalty function is nonlinear and identical for all the jobs. In the considered problem we minimize the sum of earliness and tardiness penalties, due window location penalty and due window size penalty. We proved some properties of an optimal solution and constructed a pseudopolynomial time algorithm based on the dynamic programming method. We also established the computational complexity of the problem.

1 Introduction

Scheduling problems with a common due date assignment were widely studied in the scientific literature over the past decade. This resulted in the literature survey written by Gordon et al. [1]. In this class of problems, the due date itself is a decision variable, and the objective is to schedule the jobs with minimum deviation from the due date.

However, in practice, the completion of a task is usually acceptable without penalty over a time duration. The due time duration is called due window. Some relevant references are among others: Azizoglu and Webster [2], Kramer and Lee [3], Liman et al. [4], [5], Mosheiov [6], Yeung et al. [7].

In this paper we study a parallel processor scheduling problem with a common due window. Earliness and tardiness penalties are given by nonlinear functions, which are job-independent. Moreover, the position and the size of the due window are also penalized. These penalties are described by nonlinear functions, too.

Problem with nonlinear earliness-tardiness penalty functions was also studied by Kahlbacher [8], who designed a pseudopolynomial time algorithm based on the dynamic programming method for a single processor problem with a common due date.

The model under investigation is applicable in many manufacturing systems, where the negotiation between the producer and the customer occurs. The negotiation concerns the delivery time of the final products. The producer should

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 132–139, 2006.

deliver the products before some established moment and, on the other hand the customer will not receive the products before the other established moment. This results in the due window which corresponds to the time frame during which the customer is most willing to take delivery of the products. Products manufactured too early have to be held in inventory until the customer is ready to receive them. This results in such costs as capital, insurance and deterioration costs. On the other hand products manufactured too late incur costs connected with late charges, express delivery, or loss sales.

In the following section, we present the definition of the problem. In Sect. 3 we provide some properties of an optimal solution. A dynamic programming algorithm is described in Sect. 4. Finally, we summarize our paper in Sect. 5.

2 Problem Definition

In this section we define the problem under investigation.

There is a set $\mathbf{J} = \{1, \ldots, n\}$ of n independent and non preemptive jobs to be processed on m parallel identical processors. Each processor can deal with only one job at a time. Job $j \in \mathbf{J}$ with processing time p_j is available at time zero. All jobs share a common due window. The start and the end of the due window are also called the earliest due date, e, and the latest due date, d, respectively. The size of the due window is bounded from below by D_{min} and from above by D_{max}, i.e. $D_{min} \leq d - e \leq D_{max}$.

A schedule specifies an assignment of jobs to processors, a sequence of jobs on each processor, the earliest and the latest due dates. Denote the start and the completion time of job j in the schedule σ by $S_j(\sigma)$ and $C_j(\sigma)$, respectively. (S_j and C_j will be used instead of $S_j(\sigma)$ and $C_j(\sigma)$, respectively, if there is no possible confusion as to the schedule we are referring to). Let us also define: $[x]^+ = \max(x, 0)$. The objective is to find a schedule σ to minimize the following criterion

$$Z(\sigma) = \sum_{j \in \mathbf{J}} (f_E(E_j) + f_T(T_j)) + f_W(d - e) + f_D(e),$$

where $E_j = [e - C_j]^+$ is earliness of job j, $T_j = [C_j - d]^+$ is tardiness of job j and f_E, f_T, f_W and f_D are arbitrary nondecreasing functions such that $f_\bullet(0) = 0$ ($\bullet \in \{E, T, W, D\}$). We assume that all the parameters are positive integers (including e and d). We will refer to this problem as $\mathbf{P}$.

For a given schedule σ, let us define:

- $\mathbf{J}^i$ - the set of jobs processed on processor i,
- H^i - the job from $\mathbf{J^i}$ which straddles the due window, i.e. $S_j < e$ and $C_j > d$ (note, at most one such job exists),
- $\mathbf{E^i} = \{j \in \mathbf{J}^i : S_j < e\} \setminus H^i$ - the set of early jobs on processor i (excluding job H^i),
- $\mathbf{T^i} = \{j \in \mathbf{J}^i : C_j > d\} \setminus H^i$ - the set of tardy jobs on processor i (excluding job H^i),
- $\mathbf{W^i} = \{j \in \mathbf{J}^i : C_j \leq d \wedge S_j \geq e\}$ - the set of due window jobs on processor i.

The jobs in $\mathbf{E}^i$, $\mathbf{W}^i$ and $\mathbf{T}^i$ are called $\mathbf{E}^i$-jobs, $\mathbf{W}^i$-jobs and $\mathbf{T}^i$-jobs, respectively.

Problem **P** is strongly NP-hard because it is a special case of problem $P||C_{max}$ what will be shown in Sect. 4.

3 Structural Properties of Optimal Solution

In this section we present several properties of an optimal solution, which will be used to construct an exact algorithm for problem **P**.

Property 1. There exists an optimal schedule to problem **P** in which at least one job starts at 0 and the jobs are processed without idle times between them.

Proof. Note that we can shift all the jobs and the due window to the left by some value ε and the value of the total earliness-tardiness penalty and due window size penalty will not change and due window location penalty will decrease. In this way we can obtain a schedule in which at least one job starts at 0.

We can always eliminate an idle time on the processor by shifting the jobs preceding the idle period to the right or shifting the jobs following the idle period to the left (recall that early and tardy penalty functions, f_E and f_T, are nondecreasing). □

Property 2. There exists an optimal schedule to **P**, in which $\mathbf{E}^i$-jobs $(i = 1, \ldots, m)$ are sequenced in nonincreasing order of their processing times.

Proof. Assume that in an optimal schedule σ, on some processor i there are two jobs $j, k \in \mathbf{E}^i$ such that $p_j < p_k$ and job j is immediate predecessor of job k. Let σ' denote the schedule obtained from the schedule σ by swapping the jobs j and k. This modification affects only earliness of jobs j and k. See, that $C_k(\sigma) = C_j(\sigma')$ and $S_j(\sigma) = S_k(\sigma')$. Let $x = e - S_j(\sigma)$. Thus, we have:

$$Z(\sigma') - Z(\sigma) = f_E([e - C_k(\sigma')]^+) - f_E([e - C_j(\sigma)]^+) = f_E([x - p_k]^+) - f_E([x - p_j]^+) \leq 0 \ .$$

This result contradicts the assumption that the schedule σ is optimal. □

Property 3. There exists an optimal schedule to **P**, in which $\mathbf{T}^i$-jobs $(i = 1, \ldots, m)$ are sequenced in the nondecreasing order of their processing times.

We omit this proof because it is similar to the proof of Property 2.

Property 4. There exists an optimal schedule to **P**, in which for each $i = 1, \ldots, m$, job H^i (if it exists) is either shorter than all $\mathbf{E}^i$-jobs or shorter than all $\mathbf{T}^i$-jobs.

Proof. Assume that in the optimal schedule σ, for some processor i, job H^i is longer than the shortest $\mathbf{E}^i$-job (denote this job as j) and longer than the shortest $\mathbf{T}^i$-job (denote this job as k). By Properties 1 and 2, job j is immediate predecessor of job k and job k is immediate successor of job H^i. We will show that swapping jobs j and H^i or swapping jobs k and H^i decreases the objective function value.

In order to simplify the notation, let us define:

- H - job H^i in the schedule σ;
- σ_E - the schedule obtained from the schedule σ by swapping jobs H and j;
- σ_T - the schedule obtained from the schedule σ by swapping jobs H and k;
- $x = e - S_H(\sigma)$; $y = C_H(\sigma) - d$; $z = C_H(\sigma_E) - d$; $v = e - S_H(\sigma_T)$.

The notation given above is illustrated in Fig. 1.

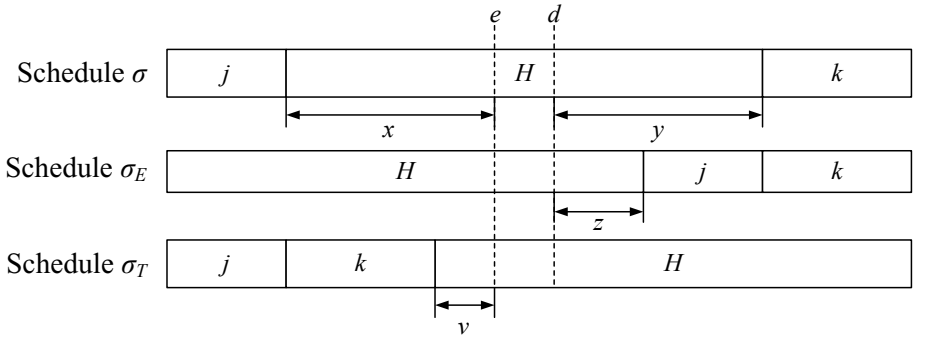


Fig. 1. Schedules σ, σ_E and σ_T with marked distances x, y, z, v

If $z < 0$ then in the schedule σ_E job H is early or completes within the due window and $T_H(\sigma) = T_j(\sigma_E)$ ($T_j(\sigma)$ denotes tardiness of job j in schedule σ). Thus, it can be easily proved by adjacent job interchanging that $Z(\sigma_E) < Z(\sigma)$ (see the proof of Property 2).

Similarly, if $v < 0$ then in the schedule σ_T job k is tardy or completes within the due window and $T_k(\sigma) = T_H(\sigma_T)$. Thus, it can be easily proved by adjacent job interchanging that $Z(\sigma_T) < Z(\sigma)$.

Now, let us investigate the case with $z > 0$ and $v > 0$. See that:

$$Z(\sigma_E) - Z(\sigma) = f_T(z) - f_E(x) \ , \tag{1}$$

$$Z(\sigma_T) - Z(\sigma) = f_E(v) - f_T(y) \ . \tag{2}$$

In order to prove that the schedule σ is not optimal, it is enough to show that (1) or (2) is negative. Thus, we will show that if expression (1) is positive, then expression (2) is negative and vice-versa. It is enough to consider the following two cases:

(*Case 1*) Assume that $Z(\sigma_E) - Z(\sigma) > 0$. Thus, $f_E(x) < f_T(z)$. Recall that $v < x$ and $z < y$. Hence, $f_E(v) \leq f_E(x) < f_T(z) \leq f_T(y)$. Thus, $Z(\sigma_T) - Z(\sigma) = f_E(v) - f_T(y) < 0$. This result contradicts the assumption that the schedule σ is optimal.

(*Case 2*) Assume that $Z(\sigma_T) - Z(\sigma) > 0$. Thus, $f_T(y) < f_E(v)$. Recall that $v < x$ and $z < y$. Hence, $f_T(z) \leq f_T(y) < f_E(v) \leq f_E(x)$. Thus, $Z(\sigma_E) - Z(\sigma) = f_T(z) - f_E(x) < 0$. This result contradicts the assumption that the schedule σ is optimal. □

Property 5. In an optimal schedule of $\mathbf{P}$, the $\mathbf{W}^i$-jobs are sequenced arbitrarily.

Proof. It is enough to see that the value of the objective function does not depend on the sequence of $\mathbf{W}^i$-jobs. □

4 Dynamic Programming Algorithm

In this section, we provide a dynamic programming algorithm $\mathbf{A}$ to solve problem $\mathbf{P}$ optimally.

Hereafter, we assume without loss of generality that the jobs are numbered according to the nondecreasing order of their processing times (i.e. $p_1 \leq p_2 \leq \ldots \leq p_n$).

In order to simplify the notation, let us define:

- $\tau_k = \sum_{j=1}^{k} p_j$;
- $\tilde{\mathbf{E}}^i = \begin{cases} \mathbf{E}^i \cup \{H^i\}, \text{ if } p_{H^i} < \min_{j \in \mathbf{E}^i} p_j, \\ \mathbf{E}^i, \qquad \text{otherwise;} \end{cases}$
- $\tilde{\mathbf{T}}^i = \begin{cases} \mathbf{T}^i \cup \{H^i\}, \text{ if } p_{H^i} < \min_{j \in \mathbf{T}^i} p_j, \\ \mathbf{T}^i, \qquad \text{otherwise;} \end{cases}$
- F^i - the first job processed on processor i;
- L^i - the last job processed on processor i.

By Properties 2, 3, and 4, it follows that $H^i \in \tilde{\mathbf{E}}^i$ or $H^i \in \tilde{\mathbf{T}}^i$ for each $i = 1, \ldots, m$ in an optimal solution of $\mathbf{P}$.

A dynamic programming algorithm $\mathbf{A}$ for problem $\mathbf{P}$ is based on Properties 1-5. We solve problem $\mathbf{P}$ by generating partial schedules. Jobs are considered in the order $1, \ldots, n$ (recall that $p_1 \leq p_2 \leq \ldots \leq p_n$). In iteration k of algorithm $\mathbf{A}$, job k is assigned to some processor i as an $\tilde{\mathbf{E}}^i$-job, $\tilde{\mathbf{T}}^i$-job or $\mathbf{W}^i$-job, $i = 1, \ldots, m$. In each partial schedule, the due window size can be different on different processors but it starts at the same time e. So, let us denote the end of the due window on processor i ($i = 1, \ldots, m$) by d^i. In iteration n, we detect an optimal schedule among schedules, in which the due window size is the same on all processors.

Define a schedule to be in the state $(k, \bar{a}, \bar{w}, \bar{b})$, where $\bar{a} = (a^1, a^2, \ldots, a^m)$, $\bar{w} = (w^1, w^2, \ldots, w^m)$ and $\bar{b} = (b^1, b^2, \ldots, b^m)$, if it includes jobs $1, \ldots, k$, the deviation of the start time of the first job processed on processor i from e is equal to a^i ($a^i = e - S_{F^i}$), the deviation of the completion time of the last job processed on processor i from d^i is equal to b^i ($b^i = C_{L^i} - d^i$), and the due window size on processor i is equal to w^i. The state variables are illustrated in Fig. 2.

Consider a pair of partial schedules in the same state. If one of theses schedules can be extended with unscheduled jobs to a complete schedule σ' with due window starting time $e = \max(a^1, a^2, \ldots, a^m)$ and due window size $w = w^1 = w^2 = \ldots = w^m$, then the schedule with the minimum total earliness-tardiness penalty can also be extended in the same way to a complete schedule σ'' with the same due window starting time and size and the total earliness-tardiness penalty that is smaller than or equal to the one for schedule σ'. Since

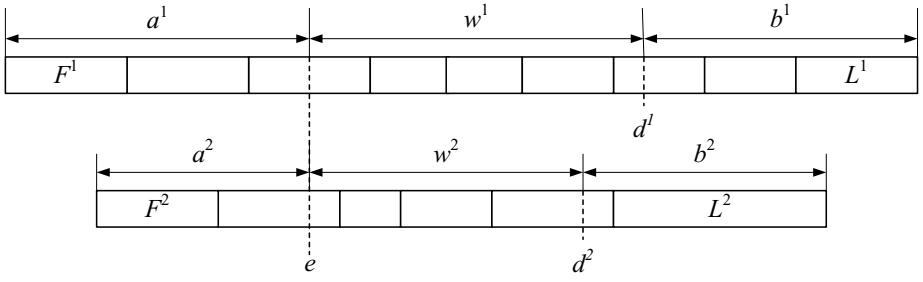


Fig. 2. Two processor partial schedule with marked state variables

both the complete schedules have the same due window starting time and size, schedule σ'' will be as good as schedule σ' with respect to the objective function of problem **P**. These observations show that only the schedule with minimum total earliness-tardiness penalty value among the schedules in the same state can be chosen for further expansion.

In order to simplify the further considerations, define $\bar{x}^i$ as a vector of size m with value 1 on position i and value 0 on the other positions.

Let $F_k(\bar{a}, \bar{w}, \bar{b})$ be the minimum total earliness-tardiness penalty value among the schedules in the state $(k, \bar{a}, \bar{w}, \bar{b})$. A schedule in the state $(k, \bar{a}, \bar{w}, \bar{b})$ with value $F_k(\bar{a}, \bar{w}, \bar{b})$ can be obtained from a schedule in some previous state by taking one of the following decisions about job k.

(i) *Schedule job k as $\tilde{\mathbf{T}}^i$-job* $(i = 1, \ldots, m)$. In this case, the previous state is $\left(k-1, \bar{a}, \bar{w}, \bar{b} - p_k\bar{x}^i\right)$ and $F_k\left(\bar{a}, \bar{w}, \bar{b}\right) = F_{k-1}\left(\bar{a}, \bar{w}, \bar{b} - p_k\bar{x}^i\right) + f_T(b^i)$.

(ii) *Schedule job k as $\mathbf{W}^i$-job* $(i = 1, \ldots, m)$. In this case, the previous state is $\left(k-1, \bar{a}, \bar{w} - p_k\bar{x}^i, \bar{b}\right)$ and $F_k\left(\bar{a}, \bar{w}, \bar{b}\right) = F_{k-1}\left(\bar{a}, \bar{w} - p_k\bar{x}^i, \bar{b}\right)$.

(iii) *Schedule job k as $\tilde{\mathbf{E}}^i$-job* $(i = 1, \ldots, m)$. In this case, the previous state is $\left(k-1, \bar{a} - p_k\bar{x}^i, \bar{w}, \bar{b}\right)$. We have $F_k\left(\bar{a}, \bar{w}, \bar{b}\right) = F_{k-1}\left(\bar{a} - p_k\bar{x}^i, \bar{w}, \bar{b}\right) + f_E([a^i - p_k]^+)$ if $k \neq H^i$ and $F_k\left(\bar{a}, \bar{w}, \bar{b}\right) = F_{k-1}\left(\bar{a} - p_k\bar{x}^i, \bar{w}, \bar{b}\right) + f_T(-a^i - w^i)$ if $k = H^i$ (if $k = H^i$, then job k is tardy and its tardiness is equal to $-a^i - w^i$).

Now, let us investigate what values of state variables a^i, b^i and w^i represent partial schedules which can be extended to feasible schedules. Value a^i $(i = 1, \ldots, m)$ can be positive (if there are some $\tilde{\mathbf{E}}^i$-jobs in the schedule) or negative (if there is no $\tilde{\mathbf{E}}^i$-job in the schedule). If it is negative, then $|a^i| \leq p_n$. Similarly, value b^i $(i = 1, \ldots, m)$ can be positive (if there are some $\tilde{\mathbf{T}}^i$-jobs in the schedule) or negative (if there is no $\tilde{\mathbf{T}}^i$-job in the schedule). If it is negative, then $|b^i| \leq p_n$. State variable w^i $(i = 1, \ldots, m)$ must be always positive or zero. Note that $a^i + b^i + w^i = \sum_{j \in \mathbf{J}^i} p_j$. In consequence, if no job is scheduled on processor i then $a^i + b^i + w^i = 0$. Since we take into account only complete schedules with $w^1 = w^2 = \ldots = w^m$, state variable w^i can not be greater than $\min(\frac{\tau_n}{m}, D_{max})$. Observe that in an optimal schedule $\min_i a^i \geq \max_i a^i + p_n$ must hold. In other case, moving the job which starts at $\max_i a^i$ to another processor would improve the criterion value. Moreover, $\sum a^i \leq \tau_n - mD_{min}$. Thus, state variable a^i can

not be greater than $\frac{\tau_n}{m} - D_{min} + p_n$. Similarly, the state variable b^i can not be greater than $\frac{\tau_n}{m} - D_{min} + p_n$. Thus, let $\hat{w}_k = \min(\tau_k, \lfloor \frac{\tau_n}{m} \rfloor, D_{max})$, $\hat{a}_k = \min(\tau_k + p_n, \lfloor \frac{\tau_n}{m} \rfloor - D_{min} + p_n)$, $\hat{b}_k = \min(\tau_k - \hat{a}_k, \lfloor \frac{\tau_n}{m} \rfloor - D_{min} + p_n)$ denote the maximal acceptable value of w^i, a^i and b^i, respectively, in iteration k of the algorithm.

Below we present a formal description of the algorithm, which was constructed based on the considerations presented above.

Algorithm A

Step 1. (*Initialization*) For $w^i = 0, \ldots, D_{max}$; $a^i = -p_n, \ldots, \tau_n - D_{min}$; $b^i = -p_n, \ldots, \tau_n - D_{min}$ $(i = 1, \ldots, m)$, set:

$$F_0(\bar{a}, \bar{w}, \bar{b}) = \begin{cases} 0, & \text{if } \forall_i \; w^i = -a^i - b^i; \\ \infty, & \text{otherwise } . \end{cases}$$

Set $k := 1$.

Step 2. (*Recursive relations*) For $a^i = -p_n, \ldots, \hat{a}_k$; $b^i = -p_n, \ldots, \hat{b}_k$; $w^i = 0, \ldots, \hat{w}_k$ $(i = 1, \ldots, m)$, if $\sum_{i=1}^{m} \left(a^i + b^i + w^i\right) = \tau_k$ is satisfied, calculate:

$$F_k(\bar{a}, \bar{w}, \bar{b}) = \min_i \begin{cases} F_{k-1}(\bar{a}, \bar{w}, \bar{b} - p_k \bar{x}^i) + f_T(t^i), & \text{if } b^i > 0, \quad (i) \\ F_{k-1}(\bar{a}, \bar{w} - p_k \bar{x}^i, \bar{b}), & (ii) \\ F_{k-1}(\bar{a} - p_k \bar{x}^i, \bar{w}, \bar{b}) + f_E([a^i - p_k]^+) + f_T([-a^i - w^i]^+), \text{ if } a^i > 0. & (iii) \end{cases}$$

If $k = n$, then go to Step 3. Otherwise set $k := k + 1$ and repeat Step 2.

Step 3. (*Optimal solution*) Calculate the objective function value for the optimal solution σ^*:

$$Z(\sigma^*) = \min_{\bar{a}, \bar{w}, \bar{b}} \Big\{ F_n(\bar{a}, \bar{w}, \bar{b}) + f_W(w^1) + f_D(\max_i a^i) : \\ D_{min} \le w^1 = w^2 = \ldots = w^m \le D_{max} \Big\} \; .$$

and establish the corresponding schedule of jobs by using the backtracking method. Earliest and latest due dates can be calculated as follows: $e = \max_i a^i$; $d = e + w^1$.

Since in each iteration of Step 2 $-p_n \le a^i \le \lfloor \frac{\tau_n}{m} \rfloor - D_{min} + p_n$; $-p_n \le b^i \le \lfloor \frac{\tau_n}{m} \rfloor - D_{min} + p_n$; $0 \le w^i \le \min D_{max}$ (for each $i = 1, \ldots, m$) and $k = 1, \ldots, n$, the time requirement of algorithm **A** is equal to:

$$O\left(n \left(\left\lfloor \frac{\tau_n}{m} \right\rfloor - D_{min} + p_n \right)^{2m} \left(\min \left(\left\lfloor \frac{\tau_n}{m} \right\rfloor, D_{max} \right) \right)^m \right) \; .$$

See that if $f_E(x) \equiv 0$ and $f_D(x) \le f_T(x)$ for each $x \in [0, \sum p_j]$ then all the jobs complete on or before the due window in an optimal solution. So, for $D_{min} = D_{max} = 0$, $f_E(x) \equiv 0$, $f_D(x) \equiv x$ and $f_T(x) \equiv x$ problem **P** reduces to

the classical parallel processor scheduling problem with the makespan criterion $(P||C_{max})$. Since problem $P||C_{max}$ is NP-hard in the strong sense, problem $P||C_{max}$ with a fixed number of machines $m \geq 2$ is NP-hard in the ordinary sense, see Garey and Johnson [9], and problem **P** can be optimally solved in $O(n(\sum p_j)^{3m})$ time, we have

Corollary 1. *Problem* **P** *is NP-hard in the strong sense. It is NP-hard in the ordinary sense for any fixed* $m \geq 2$.

5 Conclusions

In this paper we investigated a parallel processor scheduling problem with a common due window assignment. If a job does not complete within the due window, it incurs a penalty which is described by an arbitrary nonlinear function which is identical for all the jobs. We proved some properties of an optimal solution of the problem which were used to construct an optimal algorithm based on dynamic programming method. We also established the time complexity of the problem.

References

1. Gordon, V., Proth, J.M., Chu, C.: A survey of the state-of-the-art of common due date assignment and scheduling research. European Journal of Operational Research **139** (2002) 1–25
2. Azizoglu, M., Webster, S.: Scheduling about an unrestricted common due window with arbitrary earliness/tardiness penalty rates. IIE Transactions **29** (1997) 1001–1006
3. Kramer, F.J., Lee, C.Y.: Due window scheduling for parallel machine. Mathematical and Computer Modeling **20** (1994) 69–89
4. Liman, S.D., Panwalkar, S.S., Thongmee, S.: Determination of common due window scheduling problem. European Journal of Operational Research **93** (1996) 68–74
5. Liman, S.D., Panwalkar, S.S., Thongmee, S.: Common due window size and location determination in a single machine scheduling problem. Journal of the Operational Research Society **49** (1998) 1007–1010
6. Mosheiov, G.: A common due-date assignment problem on parallel identical machines. Computers & Operations Research **28** (2001) 719–732
7. Yeung, W.K., Oguz, C., Cheng, T.C.E.: Single-machine scheduling with a common due window. Computers & Operations Research **28** (2001) 157–175
8. Kahlbacher, H.G.: Scheduling with monotonous earliness and tardiness penalties. European Journal of Operational Research **64** (1991) 258–277
9. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-completness. W.H. Freeman and Co. (1979)

Distributed Architecture System for Computer Performance Testing

Ezequiel Herruzo[1], Andrés J. Mesones[1], José I. Benavides[1], Oscar Plata[2], and Emilo L. Zapata[2]

[1] Dept. Electronics, University of Córdoba, Spain
eze@uco.es
[2] Dept. of Computer Architecture, University of Málaga, Spain
oscar@ac.uma.es

Abstract. This article presents a system which is based on a distributed network architecture. The system defines a "double" Client-Server structure which permits to incorporate new client systems in real-time using the Internet. We have used this distributed architecture system to develop a tool to measure the computer performance characteristics of several computer architecture systems. The computer performance is tested by a free library called PAPI (Performance API), which allows us to access to internal status registers of several CPU families in several Operating Systems. As this testing has to be done in real CPUs, we have to create a new system to provide this functionality. The structure proposed has only one entry point to the whole system, the Master Server, and several different Architecture Client Servers. We present the network system description and the usage of PAPI for performance testing.

1 Introduction

There are several questions a programmer should consider when developing a new code: *"Am I making use of all the functionality the hardware is giving to me?" "Am I using the proper data structure?"* Most programmers do not consider these questions; they just program the code in the desired programming language, most times, not knowing what the compiler and/or language libraries are really doing with their code. The traditional recipe for this was to make a "profile" of the code, detecting the slices of the code for which most time was spent. This strategy assumes that the only problem is in the program code. However, the problem could be not in the code, but in the data structure: Cache misses could produce a considerable delay in the execution time of the programmed code. In general, the greater amount of events a code produces, the longer time a CPU must devote to handle them. Most times a proper redistribution of the data is translated into higher overall performance. Thus, it is highly desirable, for a given piece of code, to be able to know the real events produced. This figure is very difficult to estimate, but most new processors have introduced several internal registers for these purposes. So tuning the code could be resumed as: running the code in the desired processors, looking at those registers, finding out the reasons of these figures and trying to solve the possible shortcomings.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 140–147, 2006.

Our development has to gain the following goals:

- The system should work with a large amount of different computer systems, with different Operating Systems and/or CPUs.
- User should not need to "physically" move to every target computer.
- Availability of the target computers should be known by the user in advance before testing the code in them.
- User should be able to automatically load the code to be tested, compile it and collect the results.
- Comparisons among the different computer systems should be easy to be done.
- Only authorized users should be able to test the code in target computers.

As it is widely known, the Internet allows us the access to a large variety of different computer architectures. Users have got used to web browsers and the HTTP protocol. Many languages have enhanced the functionality of web servers. By using all this, a distributed network (such as the Internet) system has been developed to provide access to hardware event counters in different computer architectures through the Internet.

2 Background

2.1 Microprocessor Internal Structure

To understand the aim of our proposed system it is necessary to understand, at least briefly, how a microprocessor works. Electronics enhancement throughout the years meant lower cost of miniaturization, higher integration of transistors, larger sizes of silica to work with and, thus, higher working frequencies. This implied that many more units were included within the microprocessors, cache memories became usual inside the CPUs, etc. Performance of the CPU became a major issue. A different data distribution and an instruction arrangement could mean a great benefit in overall performance. For example, a cache miss within each iteration of a loop could slow down a code severely, as each time a cache miss occurs, a whole cache line must be replaced from memory. But, how many times does a cache miss exist for a working code in a particular CPU? Since this question could not be answered easily "off–line", designers of CPUs from the latest generations included some internal registers in which some events were counted. These registers were not directly accessible by the user.

Basically, the CPUs count the amount of times a particular event happens, this is the reason this internal registers are named internal *event counter* registers.

2.2 PAPI

We have used largely PAPI for testing the performance of different computer architecture systems. PAPI (Performance Application Programming Interface) is a free library composed of a set of 40 low-level and 6 high-level functions

which allows the user to make hardware performance tests by accessing the event counter internal registers which new processors include. PAPI has versions for several CPU families and subtypes within those families and several Operating Systems.

We use PAPI to test and measure all the available hardware counters of the processors, i.e. instruction counters (MFLOPS, MIPS, Instructions issued per cycle, etc), Cache access (L1 Cache Hits, L1 Cache Misses, L2 Cache Hits, etc), running characteristics (Total number of cycles), etc.

2.3 Related Works

Several articles [1, 6] showed that PAPI worked properly under Linux environment for accessing hardware event counters. Taking these articles as the basis, several authors have proposed tools which use PAPI to obtain information about computer performance.

IDB [10] for Performance Analysis of Parallel Scientific Applications. This is one of the first systems for the testing performance that uses PAPI. This system dissociates the performance analysis from the underlying architecture by means of the analysis of the control flow chart of the program.

SvPablo. This tool[9] is composed of a graphical code navigator and a performance visual displayer, developed by Illinois University. It combines PAPI with a dynamic performance instrumentation software library called "*Pablo Toolkit*". This tool provides the hardware event counters of the MIPS R10000 CPU. This is the main lack, as it has not been ported to any other architecture.

DynaProf. It is a performance analysis tool[2] designed for inserting performance measurement instrumentation (code sentences) directly into running applications. It is not necessary to get the source code of the program and compiling it again, thus *DynaProf* operates on a executable code and does not rely on the compilation process.

HPCToolkit. More than a standalone tool, *HPCToolkit*[4] is a set of tools working together under the same graphical interface. Its most interesting characteristic is that this environment generates the output in XML format, allowing a graphical visualization of the results in a web navigator. However, *HPCToolkit* does not work in network environments, the output is not sent to a remote computer.

All of the previous tools are entitled to work under network–isolated computers, and thus users must work directly in each of the computers under performance analysis. *SvPablo* works only under certain computer architecture (MIPS R10000) which makes the availability of this tool to be rather short and economically much more expensive. *HPCToolkit* is the tool closest to our conception, because the results are shown in a web page, however, these pages are not sent through the Internet, and the computational cost is much higher than our approach.

3 Distributed Architecture Structure

The structure we propose to gain the goals of the system (defined in section 1) is called DAS4CPT (Distributed Architecture System for Computer Performance Testing). DAS4CPT is somewhat alike a proxy system. DAS4CPT is divided into two different subsystems: the Master Server Subsystem and the Architecture Client Servers Subsystem.

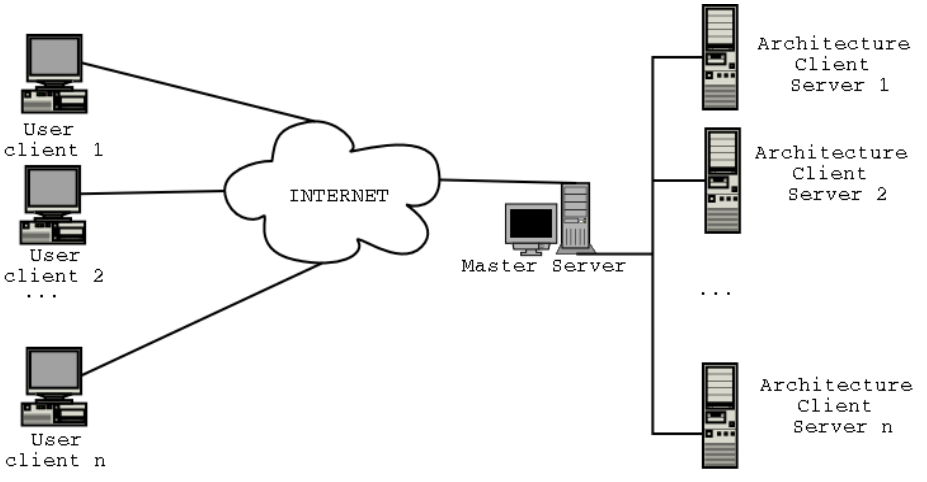


Fig. 1. DAS4CPT Network Structure

3.1 Master Server

The Master Server subsystem is composed of one computer which is in charge of controlling the users, checking the availability of the target computers, sending the information (source code and compiler flags) to the target computers, collecting the results of the test codes from every target computer, showing the comparative results to the users of all of their tests, etc. In a few words, it is the entry point to the system.

As one of the requirements was to be able to gain access to several computer architectures without the need of physically move to each of them, on the other hand, there should be some mechanisms to authorize users. A web server allows users to connect from anywhere and an authorization policy is easily implemented by a login procedure. So we decided that the Master Server is based on a Web server (we chose Apache) built up with PHP4 and MySQL database. The database would store information about users (login, password, authorization, etc.), about Architecture Client Servers (computer architecture, operating system, availability, etc.), about the tests (source code, compiler flags, authorization to be executed, results statistics for each test/user/architecture/PAPI Event).

3.2 Client Servers

Target computers are called Architecture Client Servers, because every one of them offers a service to the Master Server (and indirectly to the final users) which is its own Computer and System Architecture. These Architecture Client Servers are responsible for receiving the source code of the test, compiling it, executing it and sending the PAPI results back to the server.

First thought was to install a web server on each machine, but this approach would not be efficient, because users should know in advance which target computers were available and their URLs, and probably to install a lot of web servers is not admissible for the policy of some companies and universities because of security issues. In any case, it is not strictly necessary to install a web server. Basically, the Architecture Client Servers have to listen to requests from the Master Server, to receive test source code and compiler flags, to compile the test, to execute it and to return the results to the Master Server. Architecture Client Servers have not any check about users, that is responsibility of the Master Server. For this task, the Architecture Client Server has been developed using Java in order to make it as portable as possible. This Java program (server) runs a thread for every requested connection, this thread is responsible for receiving the source code, calling a C compiler (which should be previously installed in the target computer, we recommend DJGPP or any other GNU compiler), compiling the source code, executing the program and sending back the results to the Master Server. The Architecture Client Server has been designed to be as low load as possible. This design is very effective and versatile, because all the policies rely on the Master Server, so any change to the policies would be translated in the system by changing the Master Server, leaving the Architecture Client Server unchanged.

3.3 Network Structure

As it has been previously noted, DAS4CPT is based on a "Double" Client–Server network structure. "Double" means that both the Master Server and Architecture Client Servers behave as clients and servers. The information interchange procedure between the Master Server and an Architecture Client Server for a testing is the following:

1. The Master Server rearranges the user source code, in which the user has indicated which slice of code is to be tested, including all the necessary commands in the code to allow the count of the PAPI Events produced only in that slice of code.
2. The Master Server reserves an identifier for the test, in order to be able to send the result back to the client computer which has requested that test.
3. The Master Server sends the information to the Architecture Client Server with the following structure: [id. + flags + source code + END]
4. The information is received by the Architecture Client Server, which saves the source code of the test in a file named with the received identifier. The Architecture Client Server throws a thread for each connection accepted which will be in charge of handling the connection.

5. The Architecture Client Server compiles the source code. In case of error, the Architecture Client Server sends the compiler error message back to the Master Server, or, if successful, the test is executed.
6. The Architecture Client Server returns the result of the test to the Master Server.

3.4 Security Issues

As there are not sensitive information between the Architecture Client Server and the Master Server, we have not used SSL or any other secure method for the connection.

In order to avoid malicious test codes, some special users must authorize the tests that the users have programmed. Tests that have not been authorized are not eligible for been executed in the Architecture Client Servers. This policy can be overridden by the Administrator, in that case, all the tests are eligible by default. Architecture Client Servers accept connections coming only from the Master Server.

4 PAPI as Core for Testing

Hardware counters access is needed for several kinds of CPUs and/or Operating Systems, in which some CPUs have just a few counters, some Operating Systems do not allow the access to these counters within an user profile, etc. PAPI is a system which steps over all differences by providing a unified interface.

Although "native" CPU events are supported by PAPI, our system will only handle PAPI predefined (or standard) events, because these PAPI events cover a quantity of hardware events large enough to be well suited with the generic purposes of our system. PAPI events will allow us to take precise measures about many common items such as cache requests (L1/L2/L3 cache accesses, cache hits, cache misses, etc), conditional branching, branch prediction, TLB operations, data access, floating point operations, instructions counting (instruction per second, total number of cycles, instructions issued, etc).

5 Using the System to Performance Testing

After introducing the system structure designed and knowing some more about PAPI, we will show the implementation of the Master Server. This is how an external user would work our system. First of all, the user must log in. After a successful login, the user would be allowed to read his/her tests, write new ones or modify those previously written tests, to execute them on the Architecture Client Servers running at that very moment, to observe the statistics of all of the tests ran earlier, etc.

We are going to introduce the way a user interacts with DAS4CPT in order to develop a test and execute it in an Architecture Client Server.

1. The user in a Client User sends the test information to the Master Server; the information that should be provided is the *description* of the test, where the user could write down some information about the test; the *Previous Source Code* is the opening of the "main" function and any other previous codes needed for the code under testing, such as data initialization, dynamic memory allocation, etc.; the *Source Code under Test* is the slice of code for which the hardware events will be taken into account and counted; the *Later Source Code* is the source code remaining right after the source code under test until the closing brace "}" of the "main" function (The closing of the "main" function should not be done within the slice of the source code under test).
2. The Client User makes a request to the Master Server to use one of the tests already saved in the Master Server for that user, in order to execute it. In that case, the Master Server replies with a list of Architecture Client Servers active in that very moment.
3. The User Client picks one of the Architecture Client Servers from the list.
4. The Master Server sends to the user a list of PAPI events available depending on the selected Architecture Client Server, because not all the PAPI events are available on all the Computer Architectures.
5. Client User selects one of the PAPI Events available for the Architecture Client Server previously selected.
6. The Master Server sends a packet of information to the selected Architecture Client Server, in the way described in section 3.3.
7. Once the Master Server receives the result of the execution of the test, this result is saved in the database and it is sent to the Client User. Additionally, the Client User could decide to receive the results by email.

The system can inform each Client Server about: Operating System running, characteristics of the CPU: levels of cache, type of cache, total RAM memory, CPU frequency, etc. We have executed approximately 2.900 tests using DAS4CPT for 16 hours, with an average execution of 185,5 tests per hour, 97,5% of these tests have been successful. Readers have at their disposal the system in the URL: udaos02.uco.es/DAS4CPT. A guest user (login=guest; password=guest) is provided to make use of it.

6 Conclusions

The network architecture presented in the previous sections shows that it is a good solution for any distributed system in which geographical dispersion of resources prevents the user from making use of them. DAS4CPT is an application of the system described for a specific task: testing source code in real Computer Architecture systems. DAS4CPT has been developed by creating a Master Server which has been used by a large amount of university students. Although a previous task must be done (the Architecture Client Server program must be installed in the target computers), the dynamic addition and elimination of Architecture Client Servers from the DAS4CPT system has been a very powerful tool.

References

1. J. Dongarra, K. London, S. Moore, P. Mucci and D. Terpstra, *Using PAPI for Hardware Perfomance Monitoring on Linux Systems*, In Proceedings of the Conference on Linux Clusters: The HPC Revolution. 25–27 June 2001, (Urbana, Illinois, USA)
2. P.J. Mucci, *http://www.cs.utk.edu/∼mucci/dynaprof/* DynaProf *Last modification: November–2003.*
3. John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann; 3rd edition, 2002. ISBN: 1-55860-596-7
4. HPCToolkit project team (Rice University), *http://hipersoft.cs.rice.edu/hpctoolkit/* HPCToolkit Homepage *Last modification: December–2003.*
5. ICL PAPI Team and contributors, *http://icl.cs.utk.edu/projects/papi* Official PAPI Website.*Last modification: May–2004.*
6. K. London, S. Moore, P. Mucci, K. Seymour and R. Luczak, *The PAPI Cross–Platform Interface to Hardware Performance Counters* In Proceedings of the Department of Defense Users' Group Conference. 18–21 July 2001, (Biloxi, Mississippi, USA)
7. Andrés J. Mesones et al., *Medición de eventos del procesador en las prácticas de diseño de procesadores*. In Proceedings of the XIV Jornadas de Paralelismo. 15–17 September 2003, (Leganés, Spain). pp. 277-280. ISBN: 80-89315-34-5.
8. William Stallings, *Computer Organization and Architecture*, Prentice Hall; 6th edition, 2002. ISBN: 0-13035-119-9
9. SvPablo project team (University of Illinois at Urbana–Champaign), *http://www–pablo.cs.uiuc.edu/Project/SVPablo/SvPabloOverview.htm* SvPablo *Last mofication: October–2003.*
10. J. Nesheiwat and B.K. Szymanski,*Instrumentation Database System for Performance Analysis of Parallel Scientific Applications*, Parallel Computing, 28, no. 10, pp. 1409-1449, 2002

Data Access Time Estimation for the CASTOR HSM System

Marcin Kuta[1], Darin Nikolow[1], Renata Słota[1], and Jacek Kitowski[1,2]

[1] Institute of Computer Science, AGH-UST, al.Mickiewicza 30, Cracow, Poland
[2] Academic Computer Centre CYFRONET AGH, ul.Nawojki 11, Cracow, Poland
{mkuta, darin, rena, kito}@agh.edu.pl

Abstract. The paper presents a system for estimating the access time for data stored on the CASTOR Hierarchical Storage Management (HSM) system developed at CERN. The estimation is based on the gray-box approach. The system consists of two modules: Monitor and Simulator. Information about the current state of the HSM system is obtained by the Monitor via CASTOR API functions. The second module is based on event driven simulation of the HSM system. Special attention to the queuing algorithm is paid. Implementation details and tests results are presented.

1 Introduction

The Grid technology allows to perform large scale computations by using distributed resources provided by independent sites. Data Grids are focused on the management of data rather than on the computational issues [1]. The Data Grids often integrates heterogeneous storage systems (including HSM systems) with various performance characteristics. HSM systems are commonly used to deal with huge amount of data for economic reasons. Since the data are distributed and can be accessed from different locations, the problem of optimal access to these data arises. One method to cope with this problem is the replication method. When a data file is replicated the system has to choose which replica to use. In order to make this decision the information about the access times for each replica has to be available. The storage systems themselves do not provide such information. That is why a system estimating the data access time need to be developed.

In our previous work an estimation system for the DiskXtender HSM system has been implemented [2]. Rodney Van Meter in [3] proposes Storage Latency Estimation Descriptors (SLEDs) as a method of supplying to the client a predictive information about the I/O performance of the underlying storage systems. Shen et. al in [4] present a multi-storage architecture and a performance prediction method to increase I/O efficiency of scientific applications using SRB [5]. Delphoi service, part of the GridLab project uses queue waiting time estimation and estimation of data transfer time for optimal jobs scheduling on machines [7].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 148–155, 2006.

Tape devices, which are commonly used in HSM systems, have access times varying from less than a second to more than 100 s. Modeling the access time of modern tape drives is not trivial. Hillyer and Silberschatz [6] present in their work an accurate and detailed model for estimating the locate time for the Quantum DLT4000 tape drive. Miller and Johnson [8] propose tape seek time model based on a piece-wise linear regression approximation. Sandstå and Midtstraum [9] present a low cost access time model for serpentine tape drives. They provide analytic cost function for each possible seek scenario.

CASTOR (the CERN Advanced STORage manager), is a Hierarchical Storage System (HSM) developed in CERN [10]. It is used to store huge amounts of data from HEP experiments as well as to store general purpose user files. CASTOR transparently migrates the data between the disk cache and the tertiary storage. The access to data in CASTOR is through the use of rfio (remote file input/output) protocol.

The rest of the paper is organized as follows. The next section describes general architecture of the CASTOR HSM system. The third section presents data access time estimation algorithm and shows implementation details. Experimental results for the CASTOR HSM system are presented in the fourth section. Finally, the conclusions are presented in the last section.

2 CASTOR HSM System

2.1 General CASTOR Architecture

CASTOR is an open source hierarchical storage management system designed for (but not constrained to) storing HEP users' and experiments' files. It manages both secondary (disk) and tertiary (tape) storage.

CASTOR is a modular software system. This allows changing components without affecting the whole system. The system also allows for very distributed configurations.

All tape access is managed by the CASTOR stager. The client does not normally know about the tape location of the files being accessed. The stager interfaces with several modules:

- Name server - provides the CASTOR namespace, which appears as a normal UNIX filesystem directory hierarchy.
- Volume Manager (VMGR) - gives the status of tapes and select a tape for migration if the client created a new file or updated an existing one.
- Volume and Drive Queue Manager (VDQM) - provides a FIFO queue for accessing the tape drives.
- The CASTOR tape mover, Remote Tape COPY (RTCOPY) - is a multithreaded application with large memory buffers, for performing the copy between tape and disk.
- The RFIO server (rfiod) - manages the disk pools.

2.2 Scheduling Policy

CASTOR queue manager schedules file requests in order to:

- minimize number of tape mounts
- avoid starvation in the case when new requests for already mounted tape are continuously incoming. This could block requests which are already in the queue but can not be served because all drives are occupied.

The queue manager compares the submission date of the next request for the currently mounted tape with the submission date of the first not served request in queue. If the difference between the submission dates exceeds certain value (`VDQM_MAXTIMEDIFF` parameter), the current tape is unmounted even if there are requests for that tape which could be served. Bigger values of `VDQM_MAXTIMEDIFF` parameter mean that the system wastes less time for mounts and dismounts, lower values mean that scheduling is done in more fair manner.

Due to this priority scheduling algorithm in many cases even perfect estimation cannot give exact results because some requests can be delayed due to the incoming of new requests after the time estimation is done.

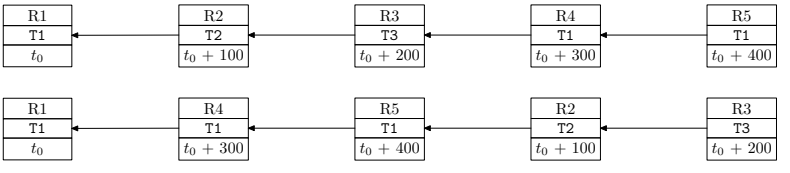


Fig. 1. Request queue and order it is scheduled by CASTOR

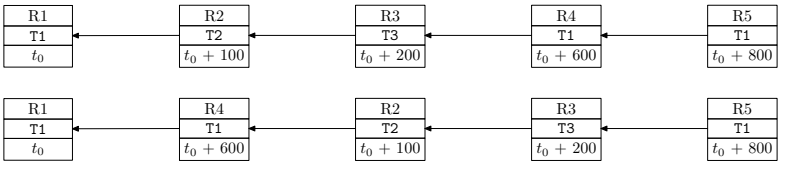


Fig. 2. Another request queue and order it is scheduled by CASTOR

Fig 1 presents simple request queue to tape library with one available drive. To serve request 1 tape `T1` is mounted. Then, to minimize number of mounts and dismounts, requests 4 and 5 to tape `T1` are served before requests 2 and 3.

Fig 2 presents similar request queue but with slightly different submission time to system. After serving request 1 tape `T1` is mounted and requests 4 and 5 could be served next. In fact only request 4 is served and tape `T1` dismounted. This is because difference between request's 5 arrival time and arrival time of request 2 (first not served request in queue) exceeds `VDQM_MAXTIMEDIFF` value (typically 600 secs).

3 Implementation of Data Access Time Estimation System for CASTOR

The implementation of the estimation system is based on the general gray-box approach [11]. That means the algorithm of Simulator is based on the knowledge gathered by observing the HSM system behavior, rather than by studying the source code. Such approach seems cheaper than modifying source code.

Anyway, the estimation algorithm depends to a certain extend, on the given HSM system and should reflect its peculiarities.

Typically during serving a request an HSM system can be in one of the following stages:

1. Waiting for resources
2. Rewinding the tape to the beginning of the tape
3. Unloading the tape
4. Moving tape from drive to slot
5. Moving tape from slot to drive
6. Loading the tape to become on-line
7. Positioning the tape to the first block of file being accessed
8. Transferring data from tertiary storage to disk cache
9. Transfer data from disk cache to client

The duration of stages 3, 4, 5, 6 is modeled by constant. The duration of stages 2, 7 depends on the current and target position and on the tape model. The duration of stages 8, 9 depends on the file size, the transfer rate and the system load (number of simultaneous requests). Stages 8, 9 shows that before delivering data to the user the whole data is introduced into the HSM disk cache what contributes to a higher latency time. This drawback is common to other HSM systems (eg. DiskXtender). Duration of stage 1 is not given by analytic formula and its value comes from the simulation.

3.1 Implementation Details

The system consists of two modules: Monitor and Simulator.

The Monitor module queries the CASTOR HSM system about its configuration and current state. The following data structures are kept in the model:

- list of automated media libraries,
- list of the drives, theirs state and characteristics,
- list of the tapes and theirs state,
- request queue,
- list of copies and fragments associated which each file in request queue,
- event queue.

The following CASTOR API functions are used to obtain information about the CASTOR HSM system current internal state:

- `Cns_stat` - gets information about a file from the name server.
- `Cns_getsegattrs` - gets the file segment attributes (size, block address, compression ratio, copy number).

- `stageqry` - this function is used to obtain the request queue, the state of each file, its submission time and the current size on disk cache. Each file is identified by logical name in the name server.
- `stage_qry_Hsm` - queries stager catalogue about an HSM file, it is used to check if the file is cached.
- `vmgr_listlibrary` - this function is used to obtain the list of all tape libraries managed by CASTOR.
- `vmgr_listtape` - this function allows to obtain the list of all tapes known by CASTOR and their state (DISABLED, EXPORTED). Tapes are identified by volume visual identifier.
- `vmgr_listdgnmap` - provides mapping between cartridge model, device group name and tape library.
- `Ctape_status` - gets the status of all tape drives. (e.g. idle, assigned).
- `vdqm_UnitStatus` - checks the current tape drive state (e.g. mounted, read, released).

The Simulator module uses event driven approach. Appropriate events are added to event queue depending on the current state of the model. After processing an event a global clock value is increased with the event estimation time.

The Monitor and Simulator modules communicate by socket mechanism. Data marshalling is realized with The Externalisation Template Library, designed for converting C++ data structures to machine independent representation [12].

All necessary for simulation parameters describing HSM components are read from estimator's configuration file. The loading and mounting time, unloading and unmounting time constants were calculated experimentally. The tape transfer rate was calculated by simple script analysing CASTOR logs.

The positioning time is computed using the low cost model introduced by Sandstå in ([9]). Unfortunately the model requires the block address of the beginning of each track of all used tapes to be calculated experimentally. As it is time consuming process the necessary data may not be available. Estimator enables to optionally model positioning time by constant.

4 Experimental Results

4.1 Testbed Configuration

The experiments were performed at the ACK Cyfronet AGH-UST site in Cracow [13] using the following equipment:

- ATL 7100 - automated tape library with 4 DLT7000 drives connected to an HP9000 K class server with four processors running HPUX 11.00
- ATL 2640 - automated tape library with 3 DLT7000 drives connected to an HP9000 K class server with four processors running HPUX 11.00

This equipment is primary devoted for the DiskXtender HSM system being in production. In order to perform the access time estimation tests for CASTOR

one drive from each tape library was configured out from DiskXtender and assigned to CASTOR. Since the robot arms devices are not exclusively used by DiskXtender software it is possible to run the both HSM systems at the same time sharing the same grippers.

The castor software installed is ver. 1.7.1.

4.2 Testing Procedure

In order to perform the tests 200 files having sizes between 10MB and 1GB have been stored on the CASTOR HSM system using four tapes. The total size of files was over 20GB and disk cache size was 2GB. During the tests file read requests with Zipf-like distribution pattern have been generated.

Three types of tests were done:

1. Single request under idle system - only one request is being served at a moment.
2. Multiple requests of files from tapes - only files residing on tapes are requested. Multiple requests can be served at a moment. The disk cache is cleared after each request.
3. Multiple requests of files from tapes and disk cache (the most realistic one)

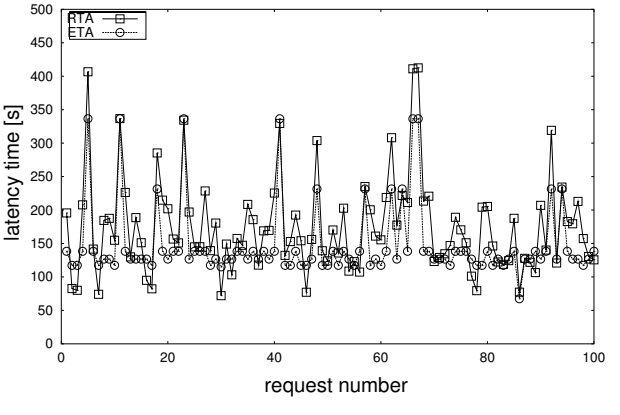
For each request in the tests the Estimated Time of Arrival (ETA) and Real (measured) Time of Arrival (RTA) are logged.

4.3 Test Results

The test results are presented in Figures 3, 4, 5. The following notation is used within the next part of the paper:

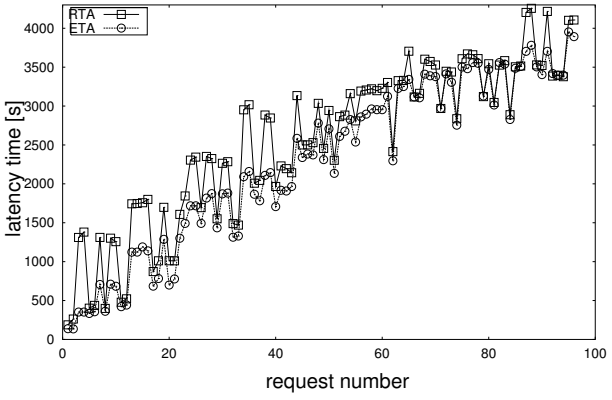
$$\Delta = RTA - ETA,\ \delta = \frac{RTA-ETA}{RTA},\ \Delta_{mean} = \frac{\sum_{i=1}^{n_{req}} |\Delta_i|}{n_{req}},\ \delta_{mean} = \frac{\sum_{i=1}^{n_{req}} |\delta_i|}{n_{req}},$$

We can see that the estimations in the single request test (Fig.3) are the least accurate. This is mainly caused by the inaccurate model of positioning time for



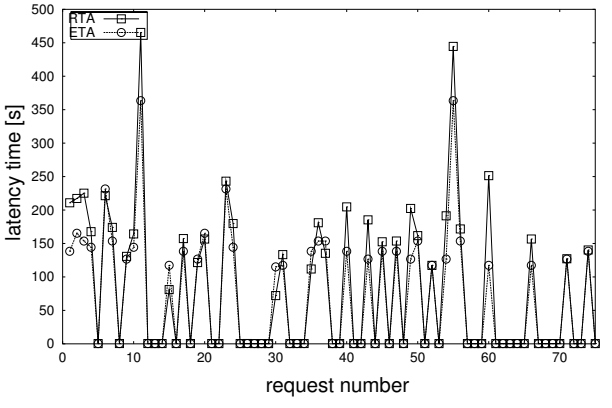
n_{req}	100
ETA_{mean}	147,55s
RTA_{mean}	175,43s
$\|\Delta\|_{mean}$	37,76s
$\|\delta\|_{mean}$	21,83%

Fig. 3. Single requests test



n_{req}	100
t_{int}	60s
ETA_{mean}	2275,5s
RTA_{mean}	2541,8s
$\|\Delta\|_{mean}$	268,2s
$\|\delta\|_{mean}$	14,0%

Fig. 4. Multiple requests test of tape files



n_{req}	75
t_{int}	60s
ETA_{mean}	70,95s
RTA_{mean}	83,04s
$\|\Delta\|_{mean}$	12,00s
$\|\delta\|_{mean}$	10,85%

Fig. 5. Multiple requests test of tape and disk cache files

DLT tapes and by the variations of compression ratios of data files which in turn causes variations of transfer times. Anyway the accuracy is sufficient for using in a replica selection process. The accuracy in the multiple requests from tapes test (Fig.4) is higher which is caused by the averaging of errors in the case when there are more than one requests in the queue. The best accuracy has been achieved in the third most realistic test (Fig.5), where requests of files from the disk cache are also estimated. The reason for this is that the disk access times are easier to predict than the tape access times.

The parameter t_{int} denotes time interval between consecutive requests.

5 Conclusions

In this paper we have presented an access time estimation method for the CASTOR HSM system. The estimation is based on the gray-box approach proposed in our previous work. The test results have shown that the achieved accuracy is

sufficient for using in Grid environments with replicated data sets also for the Castor HSM. This result validates the approach to different HSM systems.

Acknowledgment

AGH-UST grant is acknowledged.

References

1. "DataGrid – Research and Technological Development for an International Data Grid", EU Project IST-2000-25182.
2. Nikolow, D., Słota, R., Kitowski, J. "Data Access Time Estimation for HSM Systems in Grid Environment", in: Proc. of the Cracow Grid Workshop, Cracow, Poland, December 11-14, pp.209-216, ACK Cyfronet-AGH.
3. Meter, R., V., "SLEDs: Storage latency estimation descriptors", In Ben Kobler, editor, in Proc. 6th NASA Goddard Conference on Mass Storage Syst. and Tech. in Coop. with 15th IEEE Symp. on Mass Storage Syst., pp. 249-260, March 1998. Cracow, 2002.
4. Shen, X., Liao, W., Choudhary, A., "Remote I/O Optimization and Evaluation for Tertiary Storage Systems through Storage Resource Broker", in IASTED Applied Informatics, Innsbruck, Austria, February, 2001.
5. Baru, C., Moore, R., Rajasekar, A., Wan, M., "The SDSC Storage Resource Broker", in Proc. CASCON'98 Conference, Toronto, Canada, Dec. 1998.
6. Hillyer, B.K., Silberschatz, A., "On the Modeling and Performance Characteristics of a Serpentine Tape Drive", in Proc. of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Philadelphia, Pennsylvania, USA, May 1996, pp.170-179.
7. Maassen J., Rob V. van Nieuwpoort, Kielmann T., Verstoep K., den Burger M. Middleware Adaptation with the Delphoi Service. In Proceedings of the 2004 Workshop on Adaptive Grid Middleware, September 2004.
8. Miller, E. L. and Johnson, T., "Performance Measurements of Tertiary Storage Devices", in Proc. of the 24th VLDB Conference, New York, USA, August 1998, pp. 50-61.
9. Sandstå, O., Midstraum, R., "Low-Cost Access Time Model for Serpentine Tape Drives", in Proc. of 16th IEEE Symp. on Mass Storage Systems, the 7th NASA Goddard Conf. on Mass Storage Systems and Technologies, San Diego, California, USA, March 1999, pp. 116-127.
10. The CASTOR Project, `http://castor.web.cern.ch/castor`.
11. Nikolow, D., Słota, R., Kitowski, J., "Gray Box Based Data Access Time Estimation for Tertiary Storage in Grid Environment"
12. Pereira, J.O., XTL - The Externalization Template Library, Universidade do Minho, Braga, `http://xtl.sourceforge.net/xtlguide.pdf`
13. Academic Computer Center Cyfronet: `http://www.cyf-kr.edu.pl/`

Towards Distributed Monitoring and Performance Analysis Services in the K-WfGrid Project*

Hong-Linh Truong[1], Bartosz Baliś[2], Marian Bubak[2,3], Jakub Dziwisz[2], Thomas Fahringer[1], and Andreas Hoheisel[4]

[1] Institute for Computer Science, University of Innsbruck, Austria
{truong, tf}@dps.uibk.ac.at
[2] Institute of Computer Science, AGH, Poland
{balis, bubak}@uci.agh.edu.pl,dziwisz@icslab.agh.edu.pl
[3] Academic Computer Centre – CYFRONET, Poland
[4] Fraunhofer Institute for Computer Architecture and Software Technology, Germany
andreas.hoheisel@first.fraunhofer.de

Abstract. The complexity and the dynamics of the Grid environment and of the emering workflow-based applications on the Grid require novel performance monitoring and analysis services in order to capture monitoring data at multiple levels of abstraction, to analyze the data and to correlate metrics among entities. In this paper, we present the design of distributed monitoring and performance analysis services in the K-WfGrid project. We give an overview of the architecture of the performance and monitoring services and discuss useful performance and dependability metrics for workflows in K-WfGrid. We describe basic system components including the monitoring and instrumentation service, the performance analysis service along with data representation and service interface.

1 Introduction

Monitoring and analysis services are an important part of any distributed system and are essential in Grid environments because performance and monitoring information is required not only by the user to get an overview about the infrastructure and the running applications, but also by most Grid services such as brokering services, data access optimization services, and schedulers. However, due to its complexity and dynamics, various entities encompassing infrastructure elements, applications, middleware, and others, need to be monitored and analyzed in order to understand and explain performance behavior in the Grid. The K-WfGrid EU IST Project entitled "Knowledge based Workflow system for Grid Applications" [6] aims at providing knowledge-based supports for workflow construction and execution. The main objective of the K-WfGrid project is to enable the user to semi-automatically compose a workflow of Grid services, execute the composed workflow application, monitor the performance of the Grid infrastructure and workflow applications, analyse the resulting monitoring information, capture the knowledge contained in the information, and reuse the joined knowledge gathered from all participating users in a collaborative way in order to efficiently construct workflows for new Grid applications.

* The work described in this paper is supported by the European Union through the IST-2002-511385 project K-WfGrid.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 156–163, 2006.

In order to support the semi-automatic workflow composition, execution, and knowledge capture, it is required to monitor and analyse various types of performance and monitoring data related not only to resources and networks but also to workflow applications. Moreover, to support the knowledge gathered during the operation of the Grid, performance results must be stored in a knowledge base for further tasks. Our goal is to design and develop distributed monitoring and performance analysis services that address the aforementioned issues. In this paper, we describe the design of the K-WfGrid monitoring and analysis services with special focus on requirements, architecture, sensors, metrics, data representation, and service interfaces.

2 Related Work and Motivation

Current monitoring systems are usually specialized, and focus only on application or infrastructure monitoring [2, 11]. Those systems are tailored to collect and deliver a particular type of data which is reflected in their internal design and interface exposed to clients to obtain the monitoring data. Instead of monitoring applications or infrastructure separately, we proposed a unified approach to the performance monitoring and analysis for the Grid [10]. To integrate monitoring data coming from various sources such as Grid applications, infrastructure, middleware, etc., an integrated and generic framework for measuring and collecting these diverse monitoring data is required. However, until now, little effort has been spent in the development of such frameworks. Rare efforts to provide a generic monitoring infrastructure are R-GMA [4] and Mercury Monitor [7]. R-GMA employs an approach based on relational data model. Mercury is a system which provides monitoring data as metrics and also supports steering. It provides sensors for monitoring of application and resources.

However, both approaches have their limitations. R-GMA is based on Java servlets technology, where data is represented in an XML-based relational model. Those features make the solution rather slow, not suitable for transfers of large amounts of data in a grid-scale system. Mercury, in fact, was built to overcome the limitations of R-GMA with respect to data transfer. Nevertheless, both Mercury and R-GMA have deficiencies that make them unsuitable for a service-oriented knowledge-rich workflow-based grid system. First, the systems are not tailored to work in a service oriented environments. Both client-monitor and monitor-sensor interfaces are based on custom APIs which limits their interoperability. Second, data representation does not take into account semantic information which is indispensable to make the data really meaningful. Third, we argue that monitoring and performance analysis of workflow applications requires a specific support which is not addressed in the mentioned systems. This support includes, among others, a high-level abstract representation of workflow applications presented to end-user, and a standardized instrumentation service; using the abstract representation, the user can pick the workflow regions, workflow activities, or code regions to be instrumented, while the instrumentation service makes it possible to issue the instrumentation requests in a standardized manner.

Motivated by the limitation of existing tools, our goal is to develop a distributed and generic monitoring and performance analysis framework for workflow-based Grid applications. Unique features of our approach are the following:

- monitoring sensors layer is fully decoupled from the monitoring system itself; standard libraries, interfaces and procedures are developed to deploy both stand-alone and application-embedded (instrumentation) sensors,
- monitoring sensors are configurable, for example, they may be activated or deactivated at any time; sensors can be event-driven or demand-driven and with/without rule-based monitoring capabilities [9],
- monitoring data and performance results are delivered to clients on the fly,
- monitoring data types are defined with ontological description to support knowledge extraction,
- monitoring and performance analysis services are decentralized, based on a peer-to-peer model, to support scalability, availability and fault-tolerance,
- monitoring and analysis services are available as Grid/Web services to enable interoperability while the actual communication with the services is based on a low-level, but fast, mechanism to ensure efficiency.

3 Overall Architecture

To cope with the dynamic nature of the Grid, the monitoring and analysis services have to operate in a distributed and self-organizing manner. Therefore, the monitoring and analysis services will utilize a peer-to-peer architecture model. As the key issues of the Grid are *integration* and *interoperability*, the Grid services for monitoring and performance analysis of Grid infrastructures and workfows must expose a well-defined interface to other services in order to access them. Moreover, K-WfGrid is based on a service-oriented architecture (SOA). As a result, the monitoring and analysis services must be built based on a Grid service-oriented model. Performance data has to be shared among diverse services and multiple types of data have to be collected and delivered by the monitoring and analysis service. Therefore, it requires a common XML-based representation for instrumentation and monitoring requests, and XML representations for performance and event data that can be used by other services, user interfaces and Grid applications to invoke and control the monitoring and performance analysis. As performance results are stored in a knowledge base, a novel ontology describing performance data of Grid workflows is required.

Fig. 1 presents the architecture of Grid performance monitoring and analysis services. The architecture includes two main services: Generic Monitoring and Instrumentation Infrastructure (GEMINI) and Grid Performance Analysis Service (PAS). All of them will be OGSA-based services executed on multiple Grid sites. They support the instrumentation, monitoring and performance analysis of Grid workflow-based applications and infrastructures. To implement them we rely on existing implementations of the Web Service Resource Framework (WSRF)[13], e.g. Globus Toolkit (GT) [3].

The GEMINI is responsible for conducting the instrumentation, collecting performance data from applications and resources and providing that data to the PAS or other external services which require performance and monitoring data. GEMINI includes an Instrumentation Service (supporting dynamically enabled instrumentation of Grid workflow applications) and a Monitoring Service (collecting and providing performance measurements). The PAS controls the instrumentation of Grid workflow applications and analyzes the performance of applications and infrastructures based on

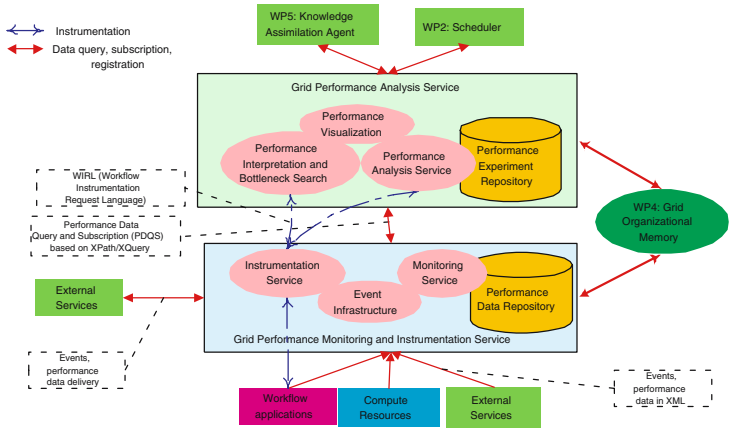


Fig. 1. Architecture of the Monitoring and Performance Analysis Framework

performance and monitoring data provided by the GEMINI. Moreover, the PAS supports the performance interpretation and bottleneck search for workflows. Both services – GEMINI and PAS – publish the information about themselves and about the types of performance data they provide into the Grid Organizational Memory (GOM) [5] which is an OWL-based knowledge base and service registry currently beeing developed within the K-Wf Grid project. In addition, an event infrastructure is provided by the monitoring infrastructure.

4 Metrics

The performance monitoring and analysis services will capture and provide several performance metrics that characterize workflow-based Grid applications. Metrics are well classified and are associated with multiple levels of abstraction such as code region, invoked application, activity, workflow construct, etc. To help the client query and subscribe metrics provided by the monitoring at runtime, information about every metric monitored is described in OWL (Web Ontology Language) [12]. The performance data of workflows is described in an OWL-based ontology by using WfPerfOnto [8].

Performance metrics are built from performance measurements and events obtained. Examples of application performance metrics are elapsed time (end-to-end response time), CPU time spent in user/system mode, communication time, etc. [8]. Events containing execution status of workflows include workflow instantiating workflow / instantiation finished (success / error), workflow execution state changed (e.g., initiated,active, terminated, suspended, completed), etc.

Infrastructure metrics include both static and dynamic information, e.g. machine name, IP address, operating system, CPU type, maximum memory/disk size, memory/disk/CPU usage, availability of a machine/service, network path bandwidth/latency/availability, etc. Given a large number of infrastructure monitoring tools, (see [2]), we do not intend to develop a new infrastructure monitoring. Instead, we focus on the

integration of existing infrastructure monitoring tools into our framework, making tool-specific metrics available through well-defined representations and interfaces.

5 Generic Monitoring and Instrumentation Infrastructure

We have designed and developed a generic monitoring and instrumentation infrastructure – GEMINI. Basically, GEMINI is composed of two main layers: (1) the network of Monitors which expose external interfaces for clients, and whose main task is to act as monitoring data broker, i.e. to manage sensors and deliver clients' requests to sensors and monitoring data back from sensors to clients; (2) Sensor Infrastructure, i.e. a number of sensors connected to Monitors which extract monitoring data and deliver it to Monitors. Though the current prototype of GEMINI features only a set of separate Monitors with a number of sensors connected to each one, in the final version we plan a fully decentralized peer-to-peer system such as the one shown in Fig. 2. In this figure, there is a distributed and decentralized network of Monitors, organized hierarchically into an upper-layer Domain Monitors (D-Monitors) and lower-layer Monitors. This organization into a super-peer topology increases the scalability of the system. A relatively low number of D-Monitors manage underlying Monitors. Thus the information needed for system management is shared only between D-Monitors.

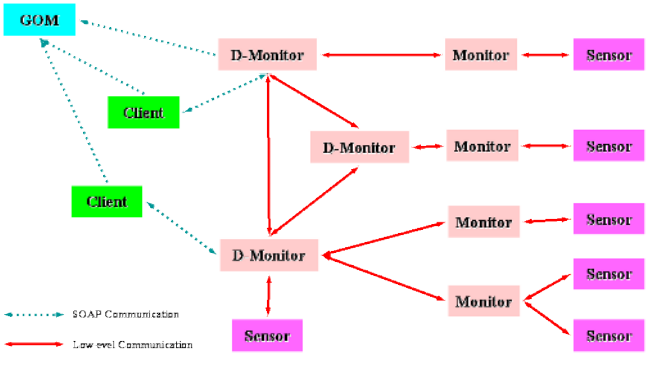


Fig. 2. Architecture of GEMINI monitoring framework

Each D-Monitor exposes two web-service interfaces: the Monitoring Interface and the Instrumentation Interface. Clients use those interfaces for requesting monitoring data and also for issuing application instrumentation requests. The actual data transfer from sensor(s) to the client (not shown) is direct and based on a low-level communication mechanism for efficiency reasons.

We provide a set of Generic Sensors which can be used to deploy both stand-alone sensors, or application-embedded ones. Generic Sensors can be easily extended with plug-ins to produce any type of monitoring data. The sensor model in K-WfGrid is an extension of a previous sensor model for Grid monitoring and data integration [10].

Several types of sensors will be supported, such as event-driven sensors (sensors deliver monitoring data upon an event), demand-driven sensors (sensors provide monitoring data upon a request), and sensors which support rule-based monitoring (sensors use rules to control their actions). Developers can very easily extend the monitoring framework just by providing new types of sensors for special purposes.

In case of workflow applications, we instrument and monitor them at several layers:

- *Workflow layer*. In this case we are interested in information on the level of entire worklow, for example its execution status, which activity is currently running, etc. We obtain this information directly from the Grid Workflow Execution Service which is instrumented for this purpose to generate appropriate events to GEMINI.
- *Activity layer*. This layer addresses individual activities of the workflow, including code regions inside activities. Instrumented services generate appropriate information to GEMINI to enable monitoring and analysis at this level.
- *Legacy code layer*. Sometimes activities invoke legacy code, even, for example, stand-alone MPI applications. We support this 'multi-lingual' scenario by adapting "legacy" monitoring system OCM-G to work as a sensor for GEMINI [1]. This is relatively easy thanks to the standard sensor layer of GEMINI.

6 Distributed Performance Analysis

Although most monitoring tools in the Grid operate in a distributed fashion, most performance analysis tools obtain monitoring data from distributed sources but analyze this data at a centralized location. Our approach is different: we design a distributed performance analysis service (DIPAS). The DIPAS includes a set of distributed Grid services which collect performance and monitoring data from the monitoring service and collaborate in doing the analysis in a distributed fashion.

Fig. 3 presents the distributed analysis framework which consists of a set of distributed Grid analysis agents. Agents are organized in groups and communicate following a peer to peer model. Agents communicate with each other by exchanging standard messages whose ontology is described by the *WfPerfOnto* ontology [8]. The clients of DIPAS will utilize the performance analysis service by invoking service operations provided by agents whose information will be published into the GOM. Thus any client that wants to request for performance analysis information can discover the analysis service by accessing the information published in the GOM. A client that wants to send an analysis request first locates an agent by searching the GOM.

Agents obtain monitoring data from the MIS and they can control the instrumentation of the workflows. Once an agent has done some analyses it stores the performance results into a performance experiment repository.

The performance of workflows will be analyzed during runtime at various levels of detail including code region, activity, and workflow construct and workflow. Different workflow graphs and performance results are collected and stored into a performance experiment repository. Based on the performance experiment repository we can conduct a multi-experiment analysis. Moreover, the performance analysis has to analyze the performance of concrete workflows and to map the performance results of concrete workflows to abstract workflows.

7 Service Interface and Data Representation

Besides exposing Web services operations, the monitoring and analysis services have to provide well-specified representations for the data they provide and for the request used to access and retrieve the data. In K-WfGrid, we have to address (i) how instrumentation requests are specified, (ii) how monitoring data and events are described, and (iii) how requests for monitoring data are defined.

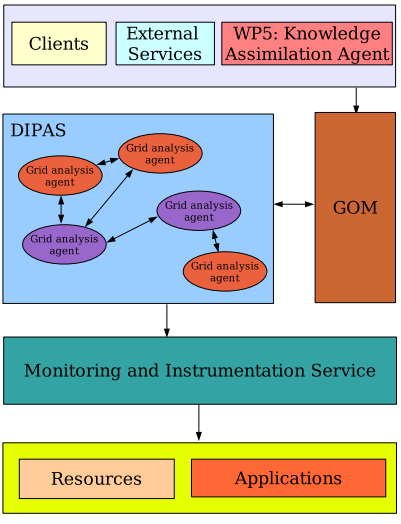


Fig. 3. Distributed analysis framework

First, we will use the workflow instrumentation request language (WIRL) as the language between the instrumentation requester (e.g. the Performance Analysis Service) and the instrumentation service. WIRL is an XML-based request and response protocol developed at the Univeristy of Innsbruck. A WIRL request consists of experiment information and instrumentation tasks. Experiment information (e.g., activity identifier, application name, computational node, etc.) identifies applications to be instrumented. Instrumentation tasks specify instrumentation operations, such as a request for all instrumented functions within an application, to enable or disable an instrumented code. An instrumentation task may contain information about code regions and metrics of interest. A WIRL response contains the name of a request, the status of the request (e.g. OK, FAIL), and detailed result information.

Second, performance measurements and monitoring data of applications and infrastructures are represented in XML. Each type of performance and monitoring data is provided by a sensor type. A message containing performance data of a monitored resource (e.g. machine, network path, code region) consists of information about the resource identifier, sensor identifier, experiment identifier, and the performance measurements. The sensor identifier, resource identifier, and experiment identifier are generic information (meta-data) that describes the monitoring data. the part expressing performance measurements is dependent on each sensor type.

Information about the supported and available monitoring data as well as the monitoring and analysis services has to be published into the GOM so that clients can access and retrieve interesting monitoring data.

Third, performance monitoring and analysis services support data query and subscription, as well as notification. Requests for data query and subscription will be expressed in a pre-defined XML schema named PDQS (Performance Data Query and Subscription). PDQS requests will be used in service interfaces for data query and subscription. PDQS requests are constructed based on OWL descriptions of the monitoring data published in the GOM. Basically, the data subscription and query requests include the subscription time (specifies the duration during which the subscription is valid), the

sensor and resource identifier (determine types of monitoring data), and the data filters (used to filter the content of performance data).

8 Summary

In this paper, we have presented the design of a novel distributed monitoring and analysis framework, which is currently being developed as a part of the K-WfGrid project. To cope with the dynamics and complexity of the Grid and the workflow-based applications executed on the Grid, the monitoring and performance analysis services are designed to work in a distributed manner – both in terms of architecture and functionality, following a peer to peer communication model, and to operate at multiple levels of abstraction, such as code region, activity and workflow. To address the integration and interoperability in the Grid, the monitoring and performance analysis services offer well-defined Web Service interfaces and data representations based on XML and OWL to other services and clients in order to discover the performance monitoring and analysis services and to utilize them. We are currently implementing the first prototype of our this framework. The first running prototype is expected to be available in autumn of 2005 under an open source licence.

References

1. B. Balis, M. Bubak, W. Funika, R. Wismueller, M. Radecki, T. Szepieniec, T. Arodz, M. Kurdziel: Grid Environment for On-line Application Monitoring and Performance Analysis, Scientific Pogrammning, vol. 12, no. 4, 2004, pp. 239-251.
2. M. Gerndt, R. Wismueller, Z. Balaton, G. Gombas, P. Kacsuk, Z. Nemeth, N. Podhorecki, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, T. Margalef, Performance Tools for the Grid: State of the Art and Future, APART White Paper, Shaker Verlag, 2004.
3. The Globus Project homepage: http://www.globus.org
4. R-GMA homepage: http://www.r-gma.org
5. K. Krawczyk, R. Slota, M. Majewska, B. Kryza, J. Kitowski: Grid Organization Memory for Knowledge Management for Grid Environment, In Proc. Cracow Grid Workshop, CGW'04, December 12-15, 2004, Cracow, Poland.
6. K-WfGrid project homepage: http://www.kwfgrid.net
7. N. Podhorszki, Z. Balaton, G. Gombas. Monitoring Message-Passing Parallel Applications in the Grid with GRM and Mercury Monitor. In: Grid Computing. Proc. 2nd European Across Grids Conference, Nicosia, Cyprus, January 2004, Springer.
8. H.-L. Truong, T. Fahringer, F. Nerieri, S. Dustdar: Performance Metrics and Ontology for Describing Performance Data of Grid Workflows, IEEE International Symposium on Cluster Computing and Grid 2005 (CCGrid2005), IEEE Computer Society Press, Cardiff, UK, 2005.
9. H.-L. Truong, T. Fahringer: Self-Managing Sensor-based Middleware for Grid Monitoring and Performance Data Integration, 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), IEEE Computer Society Press, Denver, Colorado, USA, 2005.
10. H.-L. Truong, T. Fahringer: SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid, Scientific Programming, 12(4):225-237, IOS Press, 2004.
11. S. Zanikolas, R. Sakellariou: A Taxonomy of Grid Monitoring Systems, Future Generation Computing Systems,vol. 21, no. 1, pp 163-188, January, 2005.
12. OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref
13. I. Foster el al. Modeling Stateful Resources with Web Services. Specification, Globus Alliance, Argonne National Laboratory, IBM, USC ISI, Hewlett-Packard, January 2004.

A New Diagonal Blocking Format and Model of Cache Behavior for Sparse Matrices

Pavel Tvrdík and Ivan Šimeček

Department of Computer Science and Engineering,
Czech Technical University, Prague
{tvrdik, xsimecek}@fel.cvut.cz

Abstract. Algorithms for the sparse matrix-vector multiplication (shortly $SpM{\times}V$) are important building blocks in solvers of sparse systems of linear equations. Due to matrix sparsity, the memory access patterns are irregular and the utilization of a cache suffers from low spatial and temporal locality. To reduce this effect, the *diagonal register blocking* format was designed. This paper introduces a new combined format, called *CARB*, for storing sparse matrices that extends possibilities of the diagonal register blocking format.

We have also developed a probabilistic model for estimating the numbers of cache misses during the $SpM{\times}V$ in the CARB format. Using HW cache monitoring tools, we compare the predicted numbers of cache misses with real numbers on Intel x86 architecture with L1 and L2 caches. The average accuracy of our analytical model is around 95% in case of L2 cache and 88% in case of L1 cache.

1 Introduction

There are several formats for storing sparse matrices. They have been designed mainly for the $SpM{\times}V$. The $SpM{\times}V$ for the most common format, the *compressed sparse rows* (shortly CSR) format, suffers from low performance due to the indirect addressing. Many studies were published about increasing the efficiency of the $SpM{\times}V$ [4, 1].

There are some formats, such as *register blocking*, that eliminate indirect addressing during the $SpM{\times}V$. Then, vector instructions can be used. These formats are suitable only for matrices with a known structure of nonzero elements.

The overhead of a reorganization of a matrix from one format to another one is often of the order of tens of executions of a $SpM{\times}V$. So, such a reorganization pays off only if the same matrix $\mathcal{A}$ is multiplied with multiple different vectors, e.g., in iterative linear solvers.

2 Terminology and Notation

2.1 The Cache Model

The cache model we consider corresponds to the structure of L1 and L2 caches in the Intel x86 architecture. An *s-way set-associative* cache consists of h *sets* and one set consists of s independent *blocks* (called *lines* in the Intel terminology).

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 164–171, 2006.

Let C_S denote the size of the data part of a cache in bytes and B_S denote the cache block size in bytes. Then $C_S = s \cdot B_S \cdot h$. Let S_D denote the size of type `double` and S_I the size of type `integer`.

We consider both types of cache misses: (1) *Compulsory* misses (sometimes called *intrinsic* or *cold*) that occur if the required memory block is not in the cache since it is accessed for the first time, and (2) *thrashing* misses (also called *cross-interference*, *conflict*, or *capacity* misses) that occur if the required memory block is not in the cache even though it was previously loaded, since it has been replaced prematurely from the cache due to the capacity reasons.

2.2 Common Sparse Matrix Formats

In the following text, we assume that $\mathcal{A}$ is a real sparse matrix of order n. Let n_Z be the total number of nonzero elements in $\mathcal{A}$.

The compressed sparse row (CSR) format. A matrix $\mathcal{A}$ stored in the CSR format is represented by 3 linear arrays A, adr, and ci. Array $A[1, \ldots, n_Z]$ stores the nonzero elements of $\mathcal{A}$, array $adr[1, \ldots, n]$ contains indexes of initial nonzero elements of rows of $\mathcal{A}$, and array $ci[1, \ldots, n_Z]$ contains column indexes of nonzero elements of $\mathcal{A}$. Hence, the first nonzero element of row j is stored at index $adr[j]$ in array A. This is the most common format for storing sparse matrices.

The register blocking formats. These formats are designed to handle randomly occurring dense blocks in a sparse matrix. Let N_B denote the number of dense blocks and L denote the total number of elements in all blocks. Clearly, $L \geq n_Z$. There are 2 variants:

- *Rectangular register blocking* (shortly *RRB*): Elements of $\mathcal{A}$ are grouped into dense rectangular blocks of the same size $r_x \cdot r_y$. Each block i is described by a pair $[x_i^{\text{start}}, y_i^{\text{start}}]$ (the position of its left upper corner) and $r_x \cdot r_y$ numbers (the values of all its elements, including zeroes). This format has been deeply studied [4] (including fast heuristics for finding a suboptimal sizes r_x and r_y of rectangular blocks) and it is not discussed further here.
- *Diagonal register blocking* (shortly *DRB*): Nonzero elements of $\mathcal{A}$ are grouped into **dense** diagonal blocks whose sizes can differ. Each block i is described by a pair $[x_i^{\text{start}}, y_i^{\text{start}}]$ (the position of its begin) and y_i^{end} (the y-position of its end) and $y_i^{\text{end}} - y_i^{\text{start}} + 1$ numbers (the values of all its elements). The main idea of the DRB format is illustrated on Figure 1(b). It can be implemented in 2 ways:
  - a pair of arrays $block\_elems[1, \ldots, L]$ (elements of diagonal blocks) and $block\_aux[1, \ldots, N_B]$ (items $x_i^{\text{start}}, y_i^{\text{start}}, y_i^{\text{end}}$, and pointers to the first elements of blocks into the array $block\_elems[1, \ldots, L]$) (see Figure 1(c));
  - a linked list of diagonal blocks (see Figure 1(d)).

In the DRB format, nonzero elements that have no diagonally adjacent nonzero elements are called *isolated* elements.

Storing a matrix as a set of small dense blocks is the most common technique for improving performance of the $SpM \times V$.

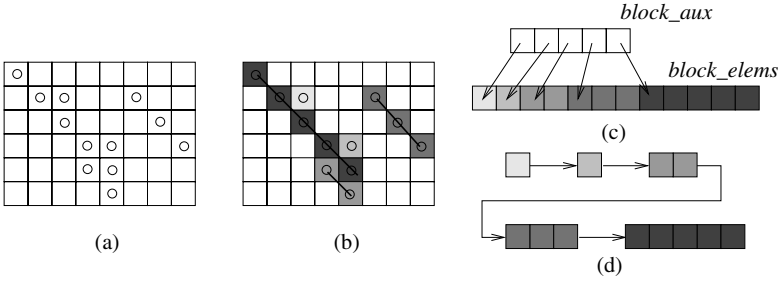


Fig. 1. The idea of the DRB format: (a) Nonzero elements of a sparse matrix $\mathcal{A}$. (b) A diagonal blocking of the nonzero elements. (c) Blocks are stored in arrays *block_aux* and *block_elems*. (d) Blocks are stored as linked dense arrays.

3 New Formats for Sparse Matrices

3.1 The Combined Diagonal Register Blocking Format

In the classical DRB format, all nonzero elements must be in diagonal blocks and isolated elements are stored as blocks of size 1. In many cases, the amount of memory for storing all these isolated-element blocks can be reduced. To combine advantages of formats CSR and DRB, we have designed a new combined format, called *C-DRB*. The goal is to decrease the memory complexity and also to improve the data locality of the $SpM{\times}V$. Simply speaking, in the C-DRB format, isolated elements are stored in the CSR format, while the blocks in the DRB format.

Let n'_{Z} denote the number of *isolated* elements in matrix $\mathcal{A}$ and let L' denote the total number of elements of all diagonal blocks. Clearly, $L' = L - n'_{\mathrm{Z}}$. Let N'_{B} denote the number of diagonal blocks of size ≥ 2.

Matrix $\mathcal{A}$ in the C-DRB format is represented by

- array $block\_elems[1,\ldots,L']$ and array $block\_aux[1,\ldots,N'_{\mathrm{B}}]$ that contain all non-isolated nonzero elements of blocks of size ≥ 2 in the DRB format.
- arrays $A[1,\ldots,n'_{\mathrm{Z}}], ci[1,\ldots,n'_{\mathrm{Z}}]$, and $adr[1,\ldots,n]$ that contain all isolated elements in the CSR format.

It is easy to see that to store all isolated elements, the DRB format requires $n'_{\mathrm{Z}}(4S_{\mathrm{I}}+S_{\mathrm{D}})$ bytes, whereas the C-DRB format requires $nS_{\mathrm{I}}+n'_{\mathrm{Z}}(S_{\mathrm{I}}+S_{\mathrm{D}})$ bytes. Hence, for matrices that satisfy $3n'_{\mathrm{Z}} > n$, i.e., the average density is greater than one isolated element per 3 rows, the C-DRB format is more space efficient.

3.2 Heuristic-Based Diagonal Register Blocking Format

Even though the C-DRB format combines the best features of the original two common formats, it may still suffer from one potential problem. Although it is easy to find all diagonal **fully dense** blocks, i.e., blocks that are **fully filled** with nonzero elements, the number of these blocks may grow significantly with

the order of matrix $\mathcal{A}$ and in the same way, the size of the required auxiliary data structure (array $block\_aux[1, \ldots, N'_{\mathrm{B}}]$) may grow. In general, this may deteriorate the space efficiency and the performance of the $SpM{\times}V$. The solution is obvious: to allow diagonal **partially full** blocks. The maximum "sparsity" of diagonal blocks can be defined by a *block heuristic*. Such a heuristic decides which fully dense blocks or isolated elements are concatenated into sparser diagonal blocks, called **H-dense** blocks. Various (even architecture dependent) block heuristics can be designed. We have designed a heuristic to minimize the space complexity. Elements are stored in a diagonal block only if the overhead of diagonal storage pays off. Let us consider a diagonal block of size u containing v nonzero elements. These elements consume $v(S_{\mathrm{D}} + S_{\mathrm{I}})$ bytes if they are stored in the CSR format or $4S_{\mathrm{I}} + uS_{\mathrm{D}}$ if they are stored as a diagonal block. So, our heuristic evaluates the condition $4S_{\mathrm{I}} + uS_{\mathrm{D}} \leq v(S_{\mathrm{D}} + S_{\mathrm{I}})$. This format is called *H-DRB*.

3.3 Cache Adaptive Formats

The C-DRB and H-DRB formats, in the same way as their predecessor, the DRB format, may contain very long diagonal blocks. However, depending on the particular cache mapping function, it may happen that long diagonal blocks produce thrashing misses with themselves. If we consider H-dense blocks, the problem is even worse. The solution is obvious: to put bounds on the length of diagonal blocks with respect to the cache memory parameters.

These observations lead us to designing modified formats that adapt to the cache memory parameters. We call them **cache-adaptive formats**. Matrix $\mathcal{A}$ is divided into disjoint horizontal belts whose height is denoted by l. The size of the parameter l must be carefully chosen to guarantee that all blocks in one belt fit into the cache. Its maximal value denoted by l_{MAX} can be estimated as follows. During multiplication of one diagonal block, l_{MAX} elements of array x, y, and $block\_elems$ are used. For optimal reusing, the cache must hold all these data, so $C_{\mathrm{S}} \geq 3S_{\mathrm{D}}l_{\mathrm{MAX}}$ which implies $l_{\mathrm{MAX}} \leq C_{\mathrm{S}}/(3S_{\mathrm{D}})$.

We have designed a cache-adaptive heuristic-based register blocking format (shortly *CARB*). Diagonal blocks are constructed only within individual belts and sorted within one belt by the positions of their upper corners.

4 Probabilistic Analysis of the Cache Behavior

Our model is based on the following simplified assumptions (similar as in [2]). We assume that

1. There is no correlation among mappings of used arrays into cache blocks.
2. The whole cache size is used for data of the $SpM{\times}V$.
3. Each execution of the $SpM{\times}V$ starts with empty caches.
4. All diagonal blocks are of the same length l. It should be valid only for thrashing misses evaluation.

The total number of cache misses is the sum of compulsory misses, thrashing misses caused by the multiplication of diagonal blocks, and thrashing misses caused by multiplication of elements stored in the CSR format (see [2]),

$$N_{\mathrm{CM}} = N_{\mathrm{CM}}^{\mathrm{C}} + N_{\mathrm{CM}}^{\mathrm{T}} + N_{\mathrm{CM}}^{\mathrm{T}}(\mathrm{CSR}).$$

4.1 Compulsory Misses

The DRB format
The total size of arrays x and y in bytes is $N_{x+y} = 2nS_{\mathrm{D}}$, the size of array *block_aux* is $N_{block\_aux} = 4N_{\mathrm{B}}S_{\mathrm{I}}$, the size of array *block_elems* is $N_{block\_elems} = LS_{\mathrm{D}}$. The number of compulsory misses can be approximated by this sum divided by cache block size B_{S}. So,

$$N_{\mathrm{CM}}^{\mathrm{C}} = \frac{2nS_{\mathrm{D}} + 4N_{\mathrm{B}}S_{\mathrm{I}} + LS_{\mathrm{D}}}{B_{\mathrm{S}}}.$$

The C-DRB format and the H-DRB format
The total size of arrays x and y is $N_{x+y} = 2nS_{\mathrm{D}}$. The size of memory for storing auxiliary information for blocks is $N_{aux} = 4N'_{\mathrm{B}}S_{\mathrm{I}}$ and the lower bound for elements in blocks is $N_{block} = L'S_{\mathrm{D}}$ and the total size of data for CSR is $N_{CSR} = S_{\mathrm{I}}n + (S_{\mathrm{D}} + S_{\mathrm{I}})n'_{\mathrm{Z}}$. The number of compulsory misses can be approximated by this sum divided by cache block size B_{S}. So,

$$N_{\mathrm{CM}}^{\mathrm{C}} = \frac{2nS_{\mathrm{D}} + 4N'_{\mathrm{B}}S_{\mathrm{I}} + L'S_{\mathrm{D}} + S_{\mathrm{I}}n + (S_{\mathrm{D}} + S_{\mathrm{I}})n'_{\mathrm{Z}}}{B_{\mathrm{S}}}.$$

4.2 Thrashing Misses

The DRB format, the C-DRB format, and the H-DRB format
During one execution of the $SpM{\times}V$, all arrays are read only once except arrays x and y. If $l < l_{\mathrm{MAX}}$, then almost no thrashing misses occur, so $N_{\mathrm{CM}}^{\mathrm{T}} = 0$.

If $l > l_{\mathrm{MAX}}$, read operations for arrays x and y cause thrashing misses. For every block, elements of arrays x and y must be reloaded. Hence, the number of thrashing misses can be approximated by

$$N_{\mathrm{CM}}^{\mathrm{T}}(\mathrm{DRB}) \simeq \frac{2S_{\mathrm{D}}N_{\mathrm{B}}l}{B_{\mathrm{S}}} \simeq \frac{2S_{\mathrm{D}}L}{B_{\mathrm{S}}},\ N_{\mathrm{CM}}^{\mathrm{T}}(\text{C-DRB, H-DRB}) \simeq \frac{2S_{\mathrm{D}}N'_{\mathrm{B}}l}{B_{\mathrm{S}}} \simeq \frac{2S_{\mathrm{D}}L'}{B_{\mathrm{S}}}.$$

The CARB format
Under our assumptions, no thrashing misses can occur, so $N_{\mathrm{CM}}^{\mathrm{T}} = 0$.

5 Evaluation of the Results

All results were measured at Pentium Celeron 2.4 GHz, 512 MB@ 266 MHz, running OS Windows XP with the following cache parameters:

The L1 cache is a data cache with $B_{\mathrm{S}} = 64$, $C_{\mathrm{S}} = 8K$, $s = 4$, $h = 32$, and LRU replacement strategy. The L2 cache is unified with $B_{\mathrm{S}} = 64$, $C_{\mathrm{S}} = 128K$, $s = 2$,

$h = 1024$, and LRU strategy. Furthermore, $S_{\mathrm{D}} = 8B$, $S_{\mathrm{F}} = 4B$, $S_{\mathrm{I}} = 4B$, For the CARB format (see Section 3.3), we have used the parameter $l_{\mathrm{MAX}} = 256$.
SW: Microsoft Visual C++ 6.0 Enterprise edition
Intel compiler version 7.1 with switches:
`-O3 -Ow -Qpc64 -G6 -QxK -Qipo -Qsfalign16 -Zp16`
All cache events were monitored by Intel Vtune performance analyzer 7.0 and verified by the Cache Analyzer [3].

5.1 Test Data

We have written a program GEN that generates symmetric, positive definite matrices produced by discretization of two elliptic partial differential equations with Dirichlet boundary condition on rectangular grids. For testing purpose, we have used 4 matrices

1. Matrix A is a random banded sparse matrix
2. Matrix B is filled by randomly located short ($l < 10$) diagonal blocks
3. Matrix C is given by discretization on a 100×100 rectangular grid
4. Matrix D is given by discretization on a 10×1000 rectangular grid

We have used 52 real matrices from various technical areas from MatrixMarket and Harwell sparse matrix test collection.

5.2 Time and Memory Complexity of the $SpM{\times}V$

Table 1 illustrates that the $SpM{\times}V$ using the CARB format causes in all cases except one less compulsory and thrashing misses than the DRB and C-DRB formats. The different way of storing elements caused

1. less number of compulsory misses, because isolated points need less memory.
2. less number of thrashing misses, because in the DRB and C-DRB formats, blocks of large sizes cause thrashing misses with themselves and other blocks during the $SpM{\times}V$.

Table 1. Summary of measured cache misses

Type of cache misses	format	matrix A	matrix B	matrix C	matrix D
Compulsory	CSR	22,6	62,3	55,1	52
(in K)	DRB	36,3	57,5	57,6	54,5
	C-DRB	23,8	57,6	44,1	41,5
	CARB	22,7	56,6	44,2	41,7
Thrashing	CSR	0	48,2	0,1	4,5
in L2 cache memory	DRB	2,7	49,5	17,8	25,8
(in K)	C-DRB	4,7	50,4	20,2	22,8
	CARB	0	47	0,2	4,5
Thrashing	CSR	0,1	85,3	4,9	4,5
in L1 cache memory	DRB	2,9	87,5	36,4	47,1
(in K)	C-DRB	4,7	87,4	31,9	44,5
	CARB	0	86,9	11,4	11,8

Table 2. The execution time of $SpM{\times}V$ in ms for type `double` (for type `float`)

Format	matrix A	matrix B	matrix C	matrix D
CSR	1,49 (0,92)	8,47 (4,15)	3,19 (2,08)	3,3 (2,12)
DRB	2,57 (2,04)	11,4 (4,63)	5,26 (3,33)	5,46 (3,2)
C-DRB	1,95 (1,24)	11,65 (4,56)	4,52 (2,39)	4,47 (2,2)
CARB	1,46 (0,94)	9,5 (4,08)	2,6 (1,54)	2,73 (1,51)

Table 2 illustrates that the DRB and C-DRB formats suffer from greater amount of auxiliary structures, which degrades performance. Clearly, matrices A and B are not suitable for diagonal blocking, since they consist mostly of isolated elements and very short diagonal blocks. For these matrices, the CARB format is slightly slower than CSR. For matrices C and D, the $SpM{\times}V$ using the CARB format is about 20% for type `float` and 35% for type `float` faster than using the CSR format.

Significant speedups (more than 10%) were achieved for 42% of real matrices, whereas significant slowdowns (more than 10%) were achieved for 10% of them.

5.3 Evaluation of the Probabilistic Model

Let $R_2 = \frac{N_{\mathrm{CM}}(L2)}{RN_{\mathrm{CM}}(L2)}$ and $R_1 = \frac{N_{\mathrm{CM}}(L1)}{RN_{\mathrm{CM}}(L1)}$ denote the ratios of the estimated and real numbers of misses for L1 and L2 caches, respectively. They represent the accuracy of the probabilistic model.

Table 3. The accuracy of the analytical models for the different formats

Ratio	format	matrix A	matrix B	matrix C	matrix D
R_2	CSR	100,6	101,6	99,8	92,4
(in %)	DRB	93,2	100,5	76,4	67,9
	CDRB	83,1	99,6	68,6	64,6
	CARB	99,9	102,9	99,5	90,2
R_1	CSR	106,3	76,1	98,3	98,9
(in %)	DRB	92,7	74,1	61,3	53,6
	CDRB	99,0	74,2	58,0	48,2
	CARB	99,9	74,3	79,5	77,9

Table 3 illustrates that the real number of cache misses was predicted with average accuracy 95% in case of the L2 cache and 88% in case of the L1 cache for the CARB format. The accuracy of an analytical model is influenced by the following assumptions.

1. We consider only effect of some read operations, mainly prologues and epilogues of cycles. This omission decreases the model accuracy especially for the L1 cache due to its less capacity.
2. We assume that all diagonal blocks are of the same length. This assumption is valid only for matrix A.

3. We assume that both L1 and L2 caches are data caches. In the Intel architecture, this assumption holds only for the L1 cache, whereas the L2 cache is unified and it is used also for storing codes of the $SpM \times V$ and system tasks. This fact is not taken into account in our formulas, but the error is small due to small code sizes.

6 Conclusions

The contribution of this paper is twofold.

- We have designed new formats for storing sparse matrices that combine advantages of the CSR format and the DRB format. One of them is adaptive to the parameters of the cache.
- We have derived an analytical probabilistic model for estimating the numbers of cache misses during a $SpM \times V$ using these formats. We have concentrated on the Intel architecture with L1 and L2 caches. Using HW cache monitoring tools, we have verified that the accuracy of our analytical model is around 90%. The errors in estimations are due to simplifying assumptions in our model. Both the model and the measurements prove that our new formats reduce the number of cache misses by around 12% in case of L2 cache and 5% in case of L1 cache.

Acknowledgements

This research has been supported by grant GA AV ČR IBS 3086102, by IGA CTU FEE under CTU0409313, and by MŠMT under research program #J04/98:212300014.

References

1. J. Mellor-Crummey and J. Garvin. Optimizing sparse matrix vector product computations using unroll and jam. *International Journal of High Performance Computing Applications*, 18(2):225–236, 2004.
2. P. Tvrdík and I. Šimeček. Analytical modeling of sparse linear code. *PPAM*, 12(4):617–629, 2003.
3. P. Tvrdík and I. Šimeček. Software cache analyzer. In *Proceedings of CTU Workshop*, volume 9, pages 180–181, Prague, Czech Republic, Mar. 2005.
4. R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee. Performance optimizations and bounds for sparse matrix-vector multiply. In *Proceedings of Supercomputing 2002*, Baltimore, MD, USA, November 2002.

GridSpace – Semantic Programming Environment for the Grid

Tomasz Gubała[2,3] and Marian Bubak[1,2]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] Section Computational Science, UvA, Kruislaan 403, 1098 SJ Amsterdam
bubak@agh.edu.pl, gubala@science.uva.nl

Abstract. The Grid computing plaform which has emerged recently, while robust and feature-rich, is still considerably complex to program for. Not only the migration of legacy applications is difficult – even the development of a new, complex application from the beginning could be quite a challenge using today's Grid middleware. This work proposes a new approach to Grid programming. The presented solution called GridSpace, based on the component programming methodology and Semantic Grid initiative achievements, employs decomposition, dynamic organization and semantic comparison techniques in order to provide a new, abstract layer for programmers of Grid applications.

Keywords: grid programming, semantic discovery, dynamic workflow.

1 The Grid Programming Problem

Thus far, the approach to Grid systems was mainly focused on the middleware layer. It provides a solution to the main problem: sharing various resources globally over multiple administrative domains. The effort applied has provided us with some considerable solutions, relatively mature and at near-production quality. [7, 20]. While all these initiatives make the Grid computing paradigm available for us, they nevertheless remain complex. Programming for these frameworks poses significant problems for Grid application developers who want to use the Grid in their daily work. Herewith we want to address the problem and propose a set of tools which may significantly simplify the act of programming Grid applications.

There are some reasons why traditional programming approaches of today will not serve Grid programmers well:

- the Grid is a constantly changing environment,
- resource users do not instantiate resources on their own,
- Grids contain resources from distinct administrative domains.

These factors cause Grid applications to be more loosely coupled that the applications we use at the moment. A developer cannot rely on a single computational resource to be there forever; he needs to be much more flexible. What is more,

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 172–179, 2006.

he is usually forced to use the resources provided and managed by others, whose policies cannot be influenced. This requires higher adaptivity. The requirements of flexibility and adaptivity can only be fulfilled in a highly dynamic system. This document describes a proposal of a platform which provides the means to develop highly dynamic Grid applications.

2 Related Work

The concept of the GridSpace from the programming paradigm perspective is based on the idea of the component programming model. There are projects which try to introduce this technique to the Grid developers community. The Common Component Architecture combines the IDL-based distributed framework concept with certain requirements of scientific applications [1]. Some frameworks dedicated to Grid computing are compliant with the standard, such as the XCAT Framework [11] and the MOCCA computing platform [17]. Another model derived from component techniques is the Fractal model [4]. It is focused on the idea of hierarchical, recursive composition of reusable software modules to properly handle the complexity of Grid applications.

Another approach is to device a ready solution instead of proposing a new standard. One of the solutions to the Grid application decomposition problem involves active objects which may be composed, deployed and published in Internet-wide Grids [3]. Another solution with a similar level of decomposition granularity is the H2O framework which proposes specially-designed pluglets as a means to wrap, deploy and reuse software as Grid applications [13]. Some other descriptions of research in this field may be found in [8].

A good overview of the effort that the Grid programming problem got may be found in [15]. This challenge has caused the emergence of a special research group of the Global Grid Forum dedicated to advanced programming models. One of the achievements of the group was a thorough overview of the techniques of developing software for distributed environments [16]. One of the recommendations of the group is a new GridRPC standard [18] which uses the remote procedure call paradigm to deliver a clear and highly useful programming model for Grids. The document lists a set of function names along with their semantics which should be implemented by every GridRPC-compliant framework. Up to now the group has pointed to implementations of the technology in two distinct systems: NetSolve [21] and Ninf [19].

Other development initiative is the GridSuperscalar programming paradigm [2]. The idea of GridSuperscalar is based on the concept of implicit parallelism of assembler code which may be extracted and exploited by an intelligent processor during program execution. In that paradigm, Grid developers program their applications in a sequential way while a specialized parser and scheduler are able to identify basic blocks of code and dependencies among them. This knowledge then forms a basis for out-of-order parallel scheduling of different tasks in a Grid.

A very strong research initiative related to the Grid programming concept has emerged from the area of workflow-based applications. Started with refining

IT world solutions (GSFL [12]) several proposals were made for scientific applications, some of which were successfully applied [5, 6, 10]. The projects usually use workflows for distribution of computations across a Grid, from conceptually-simple parameter study problems, through more complex directed acyclic graph approaches, to Turing-complete solutions such as the ones based on Petri Nets [22]. Even though the research conducted so far is impressive, some challenges remain to be solved in that area through subsequent scientific workflow-focused initiatives [9].

3 The Overview of the GridSpace Environemnt

The core concept of the solution is to look at the Grid computing from the perspective of an application developer and the working environment he requires – appropriate development tools: languages, interpreters, debuggers, monitoring etc. It is important that these tools introduce a special layer between the programming infrastructure and the resources on the Grid. Grid middleware offers such an interface on a technical level – it enables access to various distinct Grid resources using similar methods and protocols. However, the application programmer needs yet another layer to address the requirements of the dynamic Grid: a semantic-rich abstraction of Grid resources made available for him. The traditional notions of APIs and libraries do not work appropriately in the Grid environment – static binding between routine calls and routine implementations is not sufficient when the resource pool changes too frequently. The /emphstatic binding here means that even if the call client is dynamically linked to the server, it still completely depends on that certain server to be there. A new layer which provides runtime translation of requirements into particular routine call signatures is needed.

The GridSpace system is meant to:

- separate the application developer from the ever-changing Grid resources,
- provide unified access to resources by means of semantic abstractions,
- seamlessly introduce dynamism into the newly-created applications,
- support an evolving and well-organized library of applications,
- allow easy reuse of previously-constructed solutions.

The system we propose is a set of connected modules supporting the Grid application developer: a problem solving programming language to design solutions, an interpreter to execute it, a runtime layer to run the written applications and a distributed library to maintain, share and reuse solutions within a community.

Fig. 1 depicts various users and how they interact with the environment. GridSpace provides a common space for multiple developers to design and interchange their concepts as programs. Data providers add metadata descriptions to share resources with other users. In a similar way, the providers of computational elements may add their descriptions to publish resources and make them available for developers. The developers of Grid applications, using the GridSpace

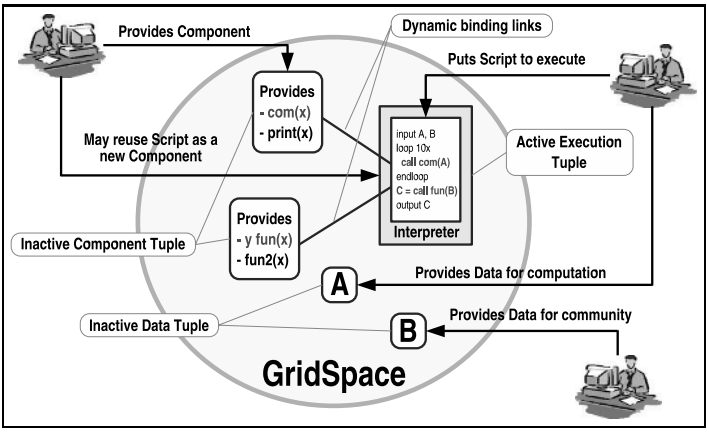


Fig. 1. GridSpace works as a vital part of PSE for multiple users

as a problem-solving environment, may apply these resources to solve certain issues. The modularity and generality of our concept make it usable for many different fields of research. In subsequent sections we describe how our tools aid the developer in the difficult task of Grid programming.

4 GridSpace Scripting Language Supporting Problem Solving

The scripting language for development of dynamic Grid applications is the topmost part of our system. It combines simple grammar, web addressing techniques and well-known flow control constructs to direct support design of distributed applications for the Grid. This language is meant as glue connecting various Grid resources together in a single application – it does not support computations directly. All the processing of the logic of an application is done within the elements which make up the application. The most distinct features are:

- using metadata descriptions instead of real data structures,
- metadata using ontologies to describe both data and computation,
- simple variables (for control expressions) based on the XML Schema,
- addressing (pointers) based exclusively on URLs,
- real processing embedded in routine calls with *ad-hoc* dynamic binding provided by the interpreter,
- direct support for parallel and distributed execution.

In order to understand the added value of our proposition it is important to perceive every piece of a new Grid application as an abstract entity defined only by its structural and semantic description. Such a description-based approach allows for highly flexible programming as the physical counterpart of a certain data or computational resource is bound to the description exclusively

at runtime and may change between subsequent application executions. This is the role of a sufficiently-sophisticated language interpreter and the runtime system.

5 Design of GridSpace Execution Environment

Technically, the interpreter residing inside our system is a simple stack machine which parses and executes scripts. The interesting function of this tool is the method of accessing every piece of data and computational resource. It translates all the higher-level, abstract semantic descriptions into physical realizations of these entities in the Grid. To this end, it uses an internal ontology inference infrastructure to reason over the provided information. As the main source of information about the available resources it uses the runtime engine residing below the programming layer and constituting an interface to the lower-level Grid middleware.

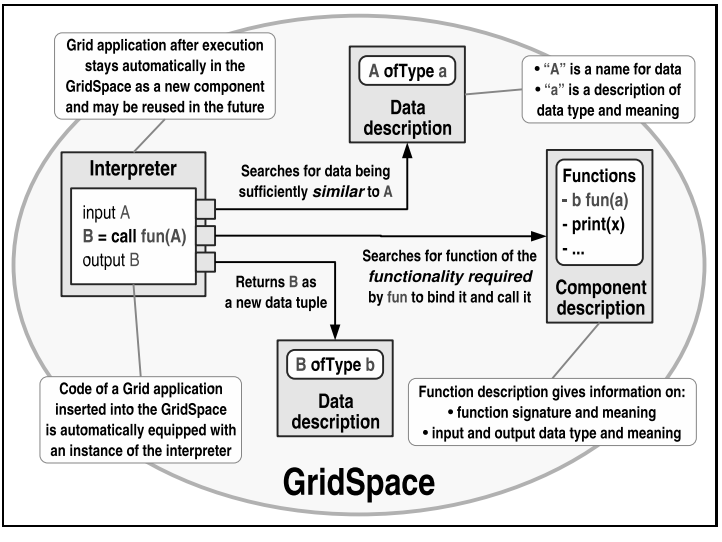


Fig. 2. Details of the dynamic semantic binding process

The most important feature of the interpreter is the notion of dynamic routine call binding. The application describes what kind of functionality it needs at a certain stage of computation and the engine tries to find an implementation of such functionality provided by the environment. This process depends heavily on ontological descriptions of data and computation semantics as it has been already proven that a simple signature-based syntactic description is not sufficient for that kind of dynamic discovery and matching process. The process is presented in Fig. 2.

6 Runtime Subsystem for GridSpace

Every programming language needs a runtime engine to conduct any processing. Our solution is based on reuse of the well-known and widely-used concept of tuple space, applied as higher level Grid middleware. The most important tuple types are the data tuple, the component tuple and the application execution tuple. All of these reside in the GridSpace logically; physically they are stored in a distributed XML registry. Each tuple consists of one description document which gives all the details of its structure, syntax and semantics. No real data or components are stored in the space – they are just referred to by URLs in descriptions. Please consult Fig. 1 to see how different tuples may appear in the GridSpace tuple space.

Data tuples describe the structure and meaning of single pieces of data, either produced during application execution or inserted into the space by a user. The description points to a certain place which contain the data. Similarly, the component tuple contains a thorough description of component functionality and capabilities – by using this information the interpreter is able to perform dynamic routine call binding.

Both previous tuples are inactive in nature. Once they are put into the space, they reside there. The last type of tuple, plan execution, is an active tuple. It holds a description of a Grid application built (written) by a developer. Internally, that tuple represents a piece of code in the scripting language connected with an instance of the interpreter. Upon injection into the system the execution of the script commences automatically. The interpreter uses resources residing in the GridSpace at the moment to perform the task, and every meaningful result of computation is stored in the space as well. The useful part of the application may become (at the behest of the user) a contribution to the dynamic language library (see Fig. 2).

The concept of searching the space in order to meet a new request also helps data management. It is not unlikely that a piece of data needed by an application at a particular step of computations may become available after the execution starts. This data may be provided by another application, possibly run by another user of the same GridSpace. Hence, GridSpace is a means of providing a seamless cooperative environment for various applications and users to combine their efforts.

7 Dynamically Evolving Language Library

Most traditional programming languages employ a dedicated library as a source or ready and reusable solutions for common development problems. However, this approach is in contrast to the dynamic state of the Grid. Its changing environment makes any type of statically pre-built solutions insufficient. The GridSpace provides another concept of a programming language library as a repository where every user is able to contribute his own hand-made solutions. In fact, our tools make this process somewhat automatic and transparent to the developer.

This effect is achieved through the concept of hierarchical components. As previously described, every computational resource is represented in the GridSpace as an abstract description. Following this idea, every new script of a Grid application may also be inserted into the space as a new element. From that moment on every user of the tuple space may reuse this computation and even include it into his own application as one of its components. It is important to remember that due to the dynamic nature of routine call binding in the GridSpace, every dependency of the component may be fulfilled even when the environment changes – if a user wants to reuse some code he is not forced to provide all the dependencies, and it also concerns data, as all the data required by the reused component may still reside in the space.

8 Implementation Perspective

The presented system concept is not dependent on a single middleware technology. It may be built using various paradigms of distributed resource sharing, including components, services and objects. However, in order to provide a good execution platform for the GridSpace we plan to join our efforts with the MOCCA [17] initiative which focuses on lightweight Grid computation middleware for CCA components based on the H2O framework [13]. Since the H2O-based middleware is Java-dependant, we also plan to use this highly portable programming solution in the implementation of the GridSpace. Apart from the usual libraries and tools for XML handling or repository back-end support (e.g. database connectors) we will resue the communication solutions applied in the H2O metacomputing framework based on the RMIX protocol [14].

9 Summary

In this paper we presented the design of our solution to the emerging Grid application programming concept. After the analysis of the problem sources we give a thorough description of works in similar fields of computer science. Next, there is a detailed explanation of the GridSpace concept – a platform for component-based Grid programming with use of various tools employing semantic Grid concepts. The project presented in this work is not finished – it is still under development and is subject to changes.

Acknowledgements. This work was partly funded by the EC Project IST-2004-004265, CoreGRID. The authors would like to express their gratitude for Peter Sloot for valuable discussions and to Piotr Nowakowski for remarks.

References

1. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings of 8th International Symposium on High Performance Distributed Computing*, pages 13–14, 1999.

2. R. Badia, J. Labarta, R. Sirvent, J. Prez, J. Cela, and R. Grima. Programming grid applications with grid superscalar. *Journal of Grid Computing*, 1(2), 2003.
3. F. Baude, D. Caromel, and M. Morel. From distributed objects to hierarchical grid components. In *International Symposium on Distributed Objects and Applications*. Lecture Notes in Computer Science, Springer-Verlag, 2003.
4. E. Bruneton, T. Coupaye, and J. Stefani. Recursive and dynamic software composition with sharing. In *7th Workshop on Component-Oriented Programming*.
5. E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in griphyn. In J. Nabrzyski et al., editor, *Grid Resource Management: State of the Art and Future Trends*, pages 99–116. Kluwer Publishing, 2003.
6. T. Oinn et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
7. I. Foster, C. Kesselman, and S. Tuecke. The anatomy on the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.
8. V. Getov and T. Kielman, editors. *Component Models and Systems for Grid Applications*. Springer, 2005.
9. T. Gubala, M. Bubak, M. Malawski, and K. Rycerz. Abstract workflow composition in k-wfgrid project environment. In *4th Cracow Grid Workshop (CGW04) – Workshop Proceedings*. ACC Cyfronet AGH, 2005.
10. A. Hoheisel. User tools and languages for graph-based grid workflows. *Concurrency and Computation: Practice and Experience*, 17, 2005.
11. S. Krishnan and D. Gannon. Xcat3: A framework for cca components as ogsa services. In *9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 90–97. IEEE Computer Society Press, 2004.
12. S. Krishnan, P. Wagstrom, and G. von Laszewski. Gsfl : A workflow framework for grid services, 2002.
13. D. Kurzyniec, T. Wrzosek, D. Drzewiecki, and V. Sunderam. Towards self-organizing distributed computing frameworks: The h2o approach. *Parallel Processing Letters*, 13(2):273–290, 2003.
14. D. Kurzyniec, T. Wrzosek, V. Sunderam, and Aleksander Slominski. Rmix: A multiprotocol rmi framework for java. In *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS'03)*, pages 140–146. IEEE Computer Society, 2003.
15. D. Laforenza. Grid programming: Some indications where we are headed. *Parallel Computing*, 28(12):1701–1720, 2002.
16. C. Lee, S. Matsuoka, D. Talia, A. Sussman, M. Mueller, G. Allen, and J. Saltz. A grid programming primer, 2001.
17. M. Malawski, D. Kurzyniec, and V. Sunderam. Mocca – towards a distributed cca framework for metacomputing. In *Proceedings of the 10th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, 2005.
18. H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. Gridrpc: A remote procedure call api for grid computing, 2002.
19. H. Nakada, Y. Tanaka, S. Matsuoka, and S. Sekiguchi. Ninf-g: a gridrpc system on the globus toolkit. In F. Berman et al., editor, *Grid Computing: Making the Global Infrastructure a Reality*, pages 625–638. John Wiley & Sons Ltd, 2003.
20. M. Romberg. The unicore grid infrastructure. In *Proc. SGI'2000 conf.*, 2000.
21. S. Vadhiyar, J. Dongarra, and A. YarKhan. Gradsolve - rpc for high performance computing on the grid. In *9th Int. Euro-Par Conference*, volume 2790, pages 394–403. Lecture Notes in Computer Science, Springer-Verlag, 2003.
22. W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

Applications Control on Grid with Synchronizers

Damian Kopanski[1], Marek Tudruj[1,2], and Janusz Borkowski[1]

[1] Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., Warsaw
[2] Institute of Computer Science, Polish Academy of Sciences,
21 Ordona Str., Warsaw
{damian, tudruj, janb}@pjwstk.edu.pl

Abstract. The paper presents new Grid application workflow paradigm based on advanced control mechanisms provided by synchronizers. Synchronizers generate program execution control decisions based on predicates computed on global application states. We propose that execution control by synchronizers is used to co-ordinate parallel applications executed on Grid. For this, we extend the graphical parallel program design environment PS-GRADE by Grid level synchronizers. Grid and application level synchronizers co-operate by sending state reports and receiving control signals over Grid communication network. Open Grid Services Infrastructure, implemented using Globus toolkit, is used for these purposes.

1 Introduction

Grid computations require new computer-aided program design tools adjusted to specific features of the new executive environment. This paper describes a parallel program graphical design tool for the Grid in which new kind of program execution control, based on the analysis of global application states, has been embedded. These states are monitored inside applications by special control processes called synchronizers. The synchronizers collect state reports from application processes, construct global consistent states, evaluate predicates on them and send back control signals to application processes. Control signals influence application behavior in asynchronous manner. They can activate additional actions inside application processes or influence the execution order of existing actions, including canceling of further execution. Such features are included into the PS-GRADE graphical program design tool [3, 4], which is a synchronizer-extended version of the P-GRADE system [1]. PS-GRADE has inherited all features of P-GRADE, which enables a very convenient and flexible graphical specification of parallel applications based on message passing in clusters or parallel systems. A program is represented by a graph, which represents processes and inter-process communication channels. Process functioning is described by a control flow diagram, whose nodes can be filled with fragments of process code in C language. Communication is specified by source and target variables used in channels. Knowledge of standard communication libraries is not required to specify programs.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 180–187, 2006.

Synchronizers in PS-GRADE provide a flexible parallel program control paradigm where the structure of a program and its functionality can change at program run-time. Such features are especially important in Grid computing, with applications executed on many separate clusters. New version of P-GRADE (P-GRADE Workflow) has been created by inclusion of the concept of application workflow [2]. A workflow diagram defines the control flow between applications and the flow of necessary data as transfers of files of intermediate results. In standard workflow P-GRADE environment, the macro data flow paradigm has been used as the main control principle. In this paper, we extend the workflow paradigm towards more flexible control and introduce Grid-level state monitoring done by Grid-level synchronizers. This extends the workflow paradigm known in the previous work, by more general principles, in which application execution control is based on predicates computed on application execution states, globally available on the Grid.

The paper is composed of 3 parts. The principles of application control based on synchronizers are described in the first part. The second part discusses implementation issues for the proposed environment on the Grid. The third part presents an example of the use of this environment for solving the TSP problem on the Grid.

2 Grid-Level Control Based on Synchronizers

In the PS-GRADE system processes report their local states to special synchronizer processes, which construct consistent application states and compute predicates on the monitored states, Fig. 1. Depending on the predicates, control decisions are taken in synchronizers. The decisions are distributed among processes in the form of control signals. The signals can activate some control actions in processes where they arrive. These actions can constitute execution of an associated code, or they can cause a current computational task to be abandoned to make room for other ones. These ideas have been included into PS-GRADE at the level of a single parallel program. Process state messages have automatically attached timestamps based on partially synchronized processor clocks. Control signal reception is performed asynchronously, which means that processes do

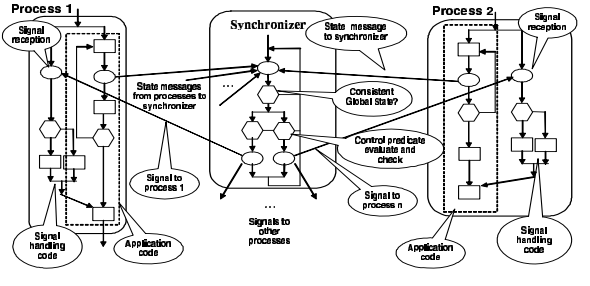


Fig. 1. Parallel processes co-operating with synchronizers

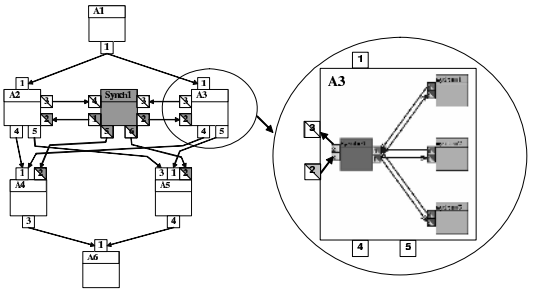


Fig. 2. A workflow graph with a Grid-level synchronizer (left), an application-level synchronizer inside an application internal graph (right)

not stop while waiting for the signals. They continue execution and react to incoming signals using a form of distributed interrupts. Synchronizers use strongly consistent states of sets of processes. They consist of pairwise concurrent local states according to a chosen state precedence relation. A strongly consistent state (SCS) is a state, which could be discovered without any doubt by a synchronizer based on received process event messages with timestamps of assumed accuracy [7, 8, 9]. The discovery of a new SCS in a synchronizer activates predicate evaluation procedures. The predicates can trigger control messages to be sent to processes. The programmer decides which states should cause sending the signals. Proper conditions and communication statements must be included into the code of predicate evaluation procedures of synchronizers.

In this paper, the program execution control by synchronizers has been extended towards co-ordination of parallel applications executed on a Grid. The Grid-oriented PS-GRADE environment is based on a modified version of PS-GRADE with process level synchronizers prepared for sending state messages to higher level synchronizers. The PS-GRADE windowing system is extended by the highest workflow level window. This level window is constructed by inclusion of application block icons and Grid-level synchronizers, which can be set in hierarchical structures. When clicking on an application block icon, the current PS-GRADE application level window opens, Fig. 2. The Grid-oriented PS-GRADE provides a much more flexible control than the standard workflow mechanism. In a standard workflow, the flow control system performs only one type of control actions it activates an application based on termination of a selected set of other applications. It corresponds to a macro data flow driven paradigm. In our system it is possible to specify other conditions, which can control the behavior of applications. For this purpose, we introduce explicit interapplication or Grid-level synchronizers. They co-operate with Grid applications, which are able to send their state reports over Grid communication network. Existing application level synchronizers are monitoring single application states and can report them to Grid-level synchronizers. A Grid level programmer decides which application states should be passed to the Grid-level synchronizer, and which have only internal meaning. Application state messages are assembled in the Grid level

synchronizers to form Grid level consistent states, using similar mechanisms as at the process level [1]. Control signals generated by the Grid-level synchronizers are sent to application level synchronizers. They respond by sending control signals down to its application processes. The programmer has to specify the process reaction in respective parts of process code.

The way in which the Grid-level synchronizers extend the standard workflow graph is shown as the left-hand side part of Fig 2. Applications A2 and A3 execute only after A1 terminates. The Grid-level synchronizer Synch1 collects state information from A2 and A3. Both applications are working in parallel to solve a search problem. When one of them finds the final solution, then Synch1 stops the other application and the application A4 is activated, taking the final solution as its input. If both A2 and A4 have terminated and no final solution has been found, then Synch1 activates A5. A6 starts when either A4 or A5 completes. All this control is graphically represented by different kinds of icons of interconnected application ports and synchronizers.

3 Grid-Level PS-GRADE Implementation

Application level synchronizers send state reports and receive signals to/from Grid-level synchronizers using new kinds of ports attached to block icons. For each application, dedicated Grid services will be started to provide inter-cluster communication. An application is coupled with such Grid services through its application-level synchronizer. The connection between the Grid services and the synchronizers is based on TCP sockets. We assume that multi cluster Grid applications will run for longer times, hours rather then seconds, the clock synchronization accuracy does not need to be high. Therefore, clocks synchronization for many clusters is not a serious technical problem [6]. The Globus Toolkit v3 (GT3) is used to implement the assumed web services infrastructure [5], Fig 3. A dedicated web service is used for signal and message delivery (sending and receiving) between GT3 sites. We call it Grade-Globus-Web-Service (GGWS). GGSW performs the intergrid communication with SOAP protocol a standard way for communication in GLOBUS Toolkit. Connections between applications and their GGWSs at different sites are provided by GRADE Server Processes

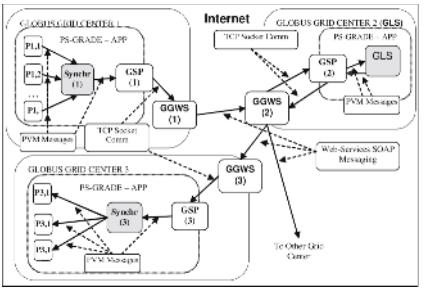


Fig. 3. Grid-oriented PS-GRADE architecture

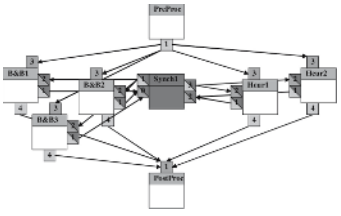


Fig. 4. The TSP application structure

(GSP). Processes P1,1 ,..., P1,n report their states to an application level synchronizer Synchr(1). Synchr(1) constructs global states and evaluates predicates on them. When a given predicate is true, then Synchr(1) sends a report about this fact to the Grid Level Synchronizer (GLS). Based on received state reports, the GLS reconstructs Grid-level global states and evaluates Grid-level predicates on them. Predicate evaluation can cause sending control signals from GLS to applications. It is done by message transfers between GLS and application-level synchronizers. The SCGS detection at the Grid-level will use state messages with timestamps originating at different clusters/sites with strongly varying message transmission time.

4 Example

The example presents how to design a TSP (Travelling Salesman Problem) program for the GRID using the extended PS-GRADE environment. The program looks for exact solutions or solutions with the preassumed quality for large number of cities. We have included more than one heuristic algorithm and an exact Branch and Bound (B&B) algorithm as parts of one application executed on many computer clusters in parallel. The heuristic algorithms find approximate solutions 10 (100) times faster than the B&B algorithm. The best solutions, found by the heuristic part of the application, are transferred to the B&B algorithm to support better bounding of the search tree in a more exhaustive search of the exact solution [9].

Our Grid application constitutes a workflow composed of 3 stages: *preprocessing*, which prepares data for the parallel speculative processing; *problem solving*, which solves the problem in parallel speculative manner. This stage consists of five grid applications (B&B1, B&B2, B&B3, Heur1, Heur2) and one global synchronizer (Synch1) controlling the execution; *post-processing*, which processes the results from the winner application.

In the middle stage, we use a parallel algorithm based on multi-grid exact B&B method with auto load balancing supported by two heuristic methods. The B&B part tries to find the exact solution. It consists of three, separate farms of worker processes, mapped to separate Grid clusters of workstations. It is controlled by the global synchronizer Synch1. In the same time, the heuristic search part tries to find the solution better than the requested quality. It is

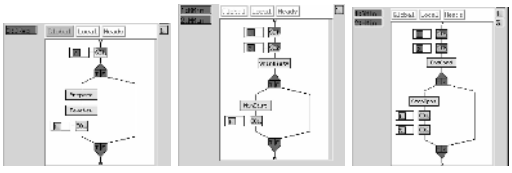


Fig. 5. Synch 1 conditions: DataRequest, MinDist, FinishCond (left to right)

composed of two search applications. They are also controlled by the Synch1 synchronizer. If one of them finds a solution with the requested quality, better than any previous search result, Synch1 transfers this solution to the B&B part. If none of the heuristic parts finds the requested solution in this stage, we wait for the B&B exact solution. After the B&B completes, the Synch1 synchronizer can activate the post-processing stage.

The Grid level synchronizer Synchr1 cooperates with local synchronizers of all search applications. It uses three conditions, Fig.5, based on three regional states: "Dreq, which consist of B&B application data requests (sent by the local synchronizer of selected B&B application), "Bmin", which consists of the best result (minimal distance) that comes from each B&B application, and "Hmin", which is the best result from the heuristic algorithms. The Data request condition, sends the new pieces of data for computing to the requesting B&B applications in response to reception of messages on "Dreq' state. The MinDist condition controls distribution of the best search results to the all the B&B applications. It is based on two states: BMin and HMin. This condition handling is activated when at least one of these states has changed. In the PS-GRADE condition activation statement it is written as REG_BMin || REG_HMin. The first two instructions of the condition flow diagram, see Fig. 5 center, collect the actual state vectors (from BMin and HMin state messages). These vectors consist of the current best result (minimal distance) obtained by the attached applications. In the first code block (SminDist=), the condition handling function computes the global minimal distance. After that, the global minimal distance is compared with the previous best result. If the new result is the best, the code block (MinDist) prepares a message with the best result to be sent to all other B&B applications (EO1 instruction). The condition FinishCond is based on two states: BMin and HMin, It monitors the best result of the whole algorithm and if it finds the requested solution. It stops all working applications by sending the stop signal to them, see Fig. 5 right.

A B&B application, Fig. 6, consists of one local application synchronizer LocSych and a number of worker processes W1, ... ,WN. The local synchronizer controls the application execution by distribution of input data to free worker processes (whose load is low) - condition DataRequest and by distribution of the best results coming from B&B applications or from the "Synch1" i.e. produced by the heuristic search condition MinDist. After reception of a data request from a worker process, the condition Data Request checks if it has unused pieces of input data for computing. If the data are available, the condition sends them to

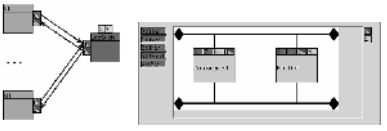


Fig. 6. B&B part: communication diagram (left),LocSynch condition window (right)

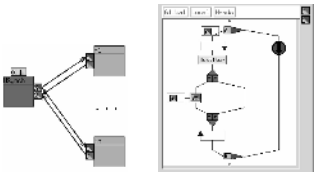


Fig. 7. Heuristic search application structure (application structure left, search process-right)

the requesting process. If the data are not available, the condition sends a data request to the global synchronizer.

Each heuristic search application should have the structure presented in Fig. 7. It should consist of one local synchronizer, which will control two aspects of the application. The synchronizer HSynch gathers the local best search results from the heuristic search applications and sends them as the application state to the global Grid synchronizer. Besides, it asynchronously waits for the end signal from the global synchronizer. If the end signal arrives, the HSynch synchronizer sends the termination signal to all working process. Each computing process is placed inside the signal sensitive region. It is delimited by the first and the last graphical instruction (start-end signal sensitive region) in the diagram shown in Fig. 7. This start instruction co-operates with signal port 1 and is followed by two control paths. The left path includes the ordinary TSP computation with iterative search block (NextIter) and the instruction for sending the best search result to the synchronizer if the comparison with the best-up-to-now result is positive (in the if block). The computations in the left path can be interrupted by the signal received by the port "1". The right search process path the signal handling function - is activated by the signal arriving to port 1. In this example, this function breaks computations in the left path and then ends the process (see the canceling icon with an arrow).

5 Conclusion

The new control paradigm included in the Grid-level PS-GRADE enables designing graphically much more flexible applications control than it is possible with the standard workflow mechanism. Grid-level application state monitoring based on Grid-level synchronizers has been introduced . State reports are collected from applications by the synchronizers, which reconstruct Grid-level

global states, evaluate predicates on them and send control signals to applications to modify applications behavior in an asynchronous manner. These features have been embedded in a graphical program design environment PS-GRADE. Globus Toolkit v3 (GT3) has been selected for implementation of web services infrastructure of the new program execution environment. Signal and message delivery between GT3 sites is done by dedicated web services. Intergrid communication is implemented with SOAP protocol. The presented example shows that the new programming environment provides convenient means for designing complicated Grid applications control. Being on the Grid, we can extend the time consuming parts of the applications and run it on any available clusters during the middle stage of the algorithm. The best results from the heuristic part of the application obtained in a shorter time than by the B&B computations, can support faster finding of the exact solution.

Full implementation of the described system is under construction in cooperation with the SZTAKI Institute of Hungarian Academy of Sciences.

References

1. P. Kacsuk, G. Dózsa, R. Lovas: The GRADE Graphical Parallel Programming Environment, In the book: Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments, Editors: P. Kacsuk, J.C. Cunha and S.C. Winter, pp. 231-247, Nova Science Publishers New York, 2001.
2. R. Lovas, G. Dózsa, P. Kacsuk, N. Podhorszki, D. Drótos, Workflow Support for Complex Grid Applications: Integrated and Portal Solutions, Proceedings of 2nd European Across Grids Conference, Nicosia, Cyprus, 2004.
3. J. Borkowski, D. Kopanski, M. Tudruj, Implementing Control in Parallel Programs by Synchronization-Driven Activation and Cancellation, Proc. of the 11-th Euromicro Conf. on Parallel, Distributed and Network-Based Processing, Feb 2003, Genova, pp. 316-323.
4. M. Tudruj, J. Borkowski, D. Kopanski, Graphical Design of Parallel Programs with Control Based on Global Application States Using an Extended P-GRADE System, in Distributed and Parallel Systems, Kluver, Vol. 777, 2004.
5. Globus Toolkit, www.globus.org/toolkit.
6. D. Mills, Network Time Protocol Specification, Implementation and Analysis, RFC1305.
7. S. D. Stoller: Detecting Global Predicates in Distributed Systems with Clocks. Distributed Computing, Volume 13 Issue 2 (2000) pp 85-98.
8. J. Borkowski, Strongly Consistent Global State Detection for On-line Control of Distributed Applications, 12-th Euromicro Conf. on Parallel Distributed and Network-Based Processing, PDP 2004, La Coruna, Spain, Feb 2004, pp. 126-133.
9. J. Borkowski, Hierarchical Detection of Strongly Consistent Global States, 3rd International Symposium on Parallel and Distributed Computing, ISPDC 2004, Cork, Ireland, July 2004, pp. 256-261.

Alchemi+: An Agent-Based Approach to Grid Programming

Roohollah Mafi[1], Hossein Deldari[1], and Mojtaba Mazoochi[2]

[1] Ferdowsi University of Mashhad, VakilAbad Bl., Mashhad, Iran
r_mafi@itrc.ac.ir, hd@ferdowsi.um.ac.ir
[2] Iran Telecommunication Research Center, End of North Kargar,
P.O. Box 14155-3961, Tehran, Iran
mazoochi@itrc.ac.ir

Abstract. Computational grids have provided the usage of computational distributed resources for computation intensive applications. The development of programs that use these capabilities is one of the challenging issues for grid computing. In this article, an effort has been made to solve this problem by presenting mobile-agent-based parallel programming on the grid. The presentation of this model which has been realized by extending Alchemi^{TM} grid system with adding agent properties and navigational commands that let user to develop his program by using mobile agents. To evaluate the system, algorithms of matrix multiplication and convex hull have been implemented in the mentioned system.

1 Introduction

Computational grids [4] have facilitated the coordinated and reliable deployment of geographically distributed resources for usage in computation intensives applications in a reliable, secure and managed situation. Development of applications are challenging because of the heterogeneity of resources.

Mobile agents [1] have been used for parallel programming on the grid in this work. Agents' autonomy and mobility will cause the deploying them in dynamic networks. Movement of code toward data is one of the major advantages of using mobile agents in parallel programming over message-passing, specially when exchanging data is massive, as well, the mobile-agent-oriented parallel programs are more eligible.

In this paper, for using grid amenities as well as gaining benefit from mobile agents in making parallel programs, a grid infrastructure called Alchemi^{TM} [7], has been used. Alchemi^{TM} is a .NET-based, low-volume, full-ability and open source system. Alchemi^{TM} supports objects' weak mobility[2]. The Object's weak mobility means that after moving an object we can only execute it from the beginning. While in programming by mobile agents, we need agent strong mobility, which provides the continuation of thread execution from the point where the previous execution was left.

According to the above points, fundamental changes in Alchemi^{TM} have been made at this paper and the capability of inter-thread communication and strong

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 188–195, 2006.

mobility have been added to it to enable the agent-oriented parallel programming. We have called the new system **Alchemi+**. The conducted tests for the evaluation of the performance of the new system indicate an achievement of a higher efficiency than MPICH-G2 (execution of MPI programs on Globus Toolkit [6] provider).

In the rest of this article, in Sect. 2, we will glance through the backgrounds to the focus of this paper, and then we the performed changes in Alchemi^{TM} will be mentioned in Sect. 3. Section 4 presents the performance evaluation of the new system, called Alchemi+, by two algorithms of matrix multiplication and convex hull. At the end, we will present the conclusion of the article and probable future works.

2 Background

We will state a brief background about grid, the role of agents in grid computing, Messengers and finally Alchemi^{TM} system.

Grid. The goal of grid computing is the illustration of a huge, powerful, virtual and self-managing computer which has been constructed from heterogeneous connected systems each of which shares a combination of resources. Ian Foster, one the founders of Globus^{TM} project has defined Grid Technology as follows:

Grid technologies seek to make this possible, by providing the protocols, services and software development kits needed to enable flexible, controlled resource sharing on a large scale. [4]

In the 90's Metacomputing projects provided the appropriate background for current grid projects. At present, abundant grid infrastructures have been developed, each one concentrating on special aspects of grid computing. Globus Toolkit^{TM} is the most famous grid core infrastructure that has been established based on Open Grid Services Architecture (OGSA) standard.

Agents and Grid Computing. Many efforts have been made to use agents in various aspects of grid computing. In [5] a four-layered architecture has been presented to make a computational grid that uses mobile agents in beneath layer for sharing computational resources of computers. In other effort, the agents have been used to resource management [3] in grid. In the present article, in quite different activity from the other mentioned previously, mobile agents have been used in parallel programming on grid infrastructure. This approach has brought about the combination of the advantages of using mobile agents in parallel programming with those of deploying grid infrastructures that are providers of abundant computational resources available to parallel applications in a reliable and secured manner.

One of the most important systems which has been used the mobile agents in parallel programming is **Messengers** [1]. In this system, the applications, which are called Messengers, develop as a set of threads, with the ability to immigrate. Each messenger can stop its execution, move to other node and then continue its execution from the previous point and status by using *hop()* statement. In

Messengers, the mobile agents is used as a programming model. Strong mobility has been enabled in Messengers system, but this system can be executed on one LAN.

In this article, suggested model of Messengers has been deployed for adding the parallel programming possibilities based on mobile agents on the grid with this difference that the modified system (Alchemi+) includes merits such as scalability.

Alchemi^{TM} Framework. One of the systems which provides grid computing on a network of personal computers is Alchemi^{TM} system[7]. This system has been implemented on Windows^{TM} and uses .NET Framework. Despite its low volume, it has provided a strong and flexible infrastructure for grid computing by using efficient architecture and components. Grid thread and Grid job models have been suggested for writing grid programs and execution of legacy applications on the grid. Also, this system can communicate with the other web service based grids. In this system, it is not possible to communicate between threads during execution that led to it is applied only in parallel applications are decomposable to the completely independent tasks.

3 Alchemi+: Agent-Based Distributed Computing on the Grid

To provide the agent-based parallel programming on the grid, inter-thread communication and strong mobility have been added to Alchemi^{TM}.

3.1 Inter-thread Communication

The communication among threads (*Gthread* objects) is not possible in Alchemi^{TM}. Due to need to inter-thread communication in mobile-agents-based programming, *GMonitor* class has been installed. The *GMonitor* class can performs on threads which are situated on different computers. A list of *GMonitor* class methods is presented in Table 1. The information of the lock and locked object transfers with the method of *GMonitor.Enter* through the interface of each agent. Whenever each agent wants to use the object, it will check it with the *GMonitor.EnterTry* method and if there is a lock on it, it will be suspended until the lock is released.

Table 1. GMonitor class members

Enter, TryEnter	Obtains a lock for an object. This action also marks the beginning of a critical section.
Wait	Releases the lock on an object in order to permit other threads to lock and access the object.
Pulse, PulseAll	Sends a signal to waiting threads. The signal notifies a waiting thread that the state of the locked object has changed.
Exit	Releases the lock on an object. This action also marks the end of a critical section protected by the locked object.

3.2 Strong Mobility Implementation

We have deployed the translation of strong mobile codes to weak mobile codes as our strategy for implementing agents' strong mobility. The translation destination is *GThread* class, which is one the Alchemi^{TM} API classes and it has weak mobility.

Each method is translated to a serializable inner class derived from *GAgent* class which contains the activation record of that method. Local variables, method parameters and program counter are converted to the fields of this class. The execution of expressions and program counter changing should be performed as atomic unit to permit the agent to move from one place to another with no information loss. ***Go()*** method will cause the movement of an agent to its new destination. Each mobile agent should implement **IAgent interface** (directly or indirectly). *Go()* function is one of the members of *IAgent* interface.

The **GAgent class** is the implementation of *IAgent* interface and provides various methods. A list of important methods of *GAgent* class has been presented in Table 2. The **user code** should be written in a *GAgent* derived class. Agent execution will begin from *start()* method and therefore the user executive code should be situated in this method. The code uses *GAgent* class methods to write agent-oriented parallel algorithms.

Table 2. GAgent class members

From	Returns the previous Node before Go
FromLink	Returns the previous Link before Go
getContextInfo	Returns the Agent's context Info
getNode	Returns the node on which the agent resides now
Go	Moves this Agent to another host.
Id	The unique id associated with this agent.
Replicate	Replicates Agent and Moves it to GNode destination.
Start	An agent does its main work in Start method.
Terminate	Terminates Current Agent

Method Translation. The pre-processor generate a class for each agent method. the object of this class represent the activation records of that method. Activation record class of each method is defined as the subclass of *MarshalByRefObject* class. Consider the sample function in Fig. 1. The x parameter, local variable y and program counter will be some of the properties of the class _sample. The existence of a method such as *setPCforMove()* is essential to the suspension of execution and the movement of a thread. This method stores the current program counter and then assigns -1 to it. *Run()* method includes the translated code from the *sample()* function. To dominate synchronization problems, each thread respectively requests and releases a lock before and after the execution of expression. This work is performed by *read_accomplished()* and *request_read()* functions. The generated activation record class from *sample()* function is shown in Fig. 2.

```
public void sample(int x) {
  // local variables
  int y;

  // blocks of statements

  BC1

  BC2

}
```

Fig. 1. Structure of sample() method

```
[Serializable] protected class
_sample : MarshalbyRefObject{
int x,y, progCounter = 0;
Object trgt;
void setPCForMove() { ...}
 void run() {
 try { ...
   this.request_read();
   if ((progCounter = = 0)) {
       progCounter+=1; BC1 }
   this.read_accomplished();
   this.request_read();
   if ((progCounter = = 1)) {
       progCounter+=1; BC2 }
   this.read_accomplished(); }
 catch(Exception e) { ... }} ... }
```

Fig. 2. Generated class from sample()

4 Performance Evaluation

To evaluate the performance of Alchemi+, two algorithms of "convex hull" and "matrix multiplication" have been implemented with it. In addition, the algorithms have been compared with the message-passing versions of them on MPICH-G2 and MPICH. The execution environments of the algorithms have been presented in Table 3.

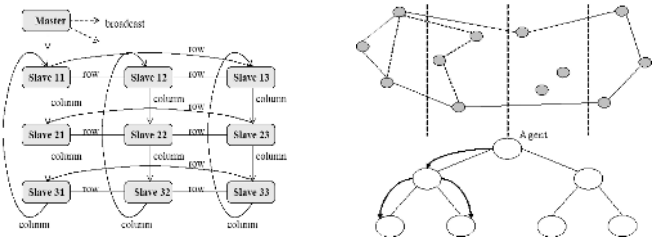
Table 3. Specifications of environments used for experiments

	Alchemi+	MPICH-G2	MPICH
Processors	8 * P4 2.4 GHz	1 * P4 2.4 GHz	8 * P4 2.4 GHz
OS	Windows XP	Redhat Linux 9.0	Redhat Linux 9.0
Middleware	.NET Framework	Globus Toolkit 3.2	OS Kernel

4.1 Matrix Multiplication

One of the methods for implementing distributed matrix multiplication is block circulating. Figure 3(a) shows how this algorithm operates. Figure 4 presents the pseudo-code of this algorithm. The obtained execution results of this algorithm on three environments are shown in Table 4. The size of the matrices is [160, 160] and [180, 180], respectively. The time unit is millisecond.

The reason for declining the performance of the Alchemi+ algorithm by increase of the number of processors and performs weaker than MPICH and MPICH-G2 is the amount of overheads and the nature of matrix multiplication algorithm. In detail, when the number of processors increases, the computing block size decreases, and as a result, the ratio of communication to computation will increase. Because of the overheads that this system adds to real data



(a) Matrix Multiplication (b) Convex Hull

Fig. 3. Distributed Algorithms on Alchemi+

```
1 matrix_mult(m)
2 {
3 master=this.getContextInfo();
4 procs=m*m;
5 for(k=0;k<m;k++){sync=0;
6 // goto all slave nodes
7 for(i=0;i<GNodeCollection
8             .getNodeCount();i++)
9 this.Replicate(GNodeCollection
10                      .Item(i));
11 if(node.j==(node.i+k)%m){
12 //copy A to nodes in same row
13 this.A=copy(this.getNode().A);
14 for(i=0;i<GNodeCollection
15            .getNodeCount();i++)
16 if(GNodeCollection.Item(i)
17     .getName().equals("row"))
18 this.Replicate(GNodeCollection
19                      .Item(i));
20 } this.Terminate();
21 multiply(); //Cij=Aij*Bij
22 try{ // barrier synchronization
23 GMonitor.Enter(sync); sync++;
24 if(sync!=proc) GMonitor.Wait(sync);
25 else GMonitor.PalseAll(sync); }
26 finally{ sync=0;
27 GMonitor.Exit(sync); }
28 this.B=copy(this.getNode().B);
29  //rotate B to columni 1
30 this.Go(this.FromLink("-column"));
31 this.getNode().B=copy(this.B);
32 try{//barrier synchronization
33 GMonitor.Enter(sync);
34 sync++;
35 if(sync!=proc)GMonitor.Wait(sync);
36 else GMonitor.PalseAll(sync);
37 } finally{
38 sync=0;
39 GMonitor.Exit(sync);
40 } } }
```

Fig. 4. Matrix multiplication pseudo-code in Alchemi+

Table 4. Results of matrix multiplication. execution on three environments

	[160,160]*[160,160]			[180,180]*[180,180]		
N	MPICH	MPICH-G2	Alchemi+	MPICH	MPICH-G2	Alchemi+
1	49.655	75.351	56.504	76.380	103.064	89.737
2	33.752	47.492	40.698	52.158	73.736	65.421
3	21.764	31.712	28.537	39.119	54.604	48.000
4	17.311	24.892	24.267	29.035	40.286	36.050
6	14.203	18.802	20.914	20.689	28.119	27.140
8	11.683	15.896	19.800	17.100	21.407	24.412

at the time of communication, which is a summation of overheads of .NET, Alchemi^{TM} and finally Alchemi+, the speed of program execution will decrease. While executed program on MPICH, will be executed with the least possible overhead. Nevertheless, despite this weakness, the ability to execute the code on a wide dispersed grid has been provided by his system. In general, the execution of the programs that consist high-related subtasks on the grid is not beneficial.

4.2 Convex Hull

The convex hull algorithm from "divide and conquer" class, needs the making of a dynamic search tree. As illustrated in Fig. 3(b), the program divides the points into two subset with a size of $\frac{n}{2}$. To solve this problem, in Fig. 5, a program written by using Alchemi+ API, has been presented. Table 5 contains the execution results. The size of problem is 30 and 50, respectively. In this algorithm, because the volume of the data transferred by the agent is lower than the former example, the performance of Alchemi+ has predictably moved closer to that of of MPICH and further away from that of MPICH-G2.

```
1 convex_hull(points)  {                10 } my_pt = convex(top_pt, tail_pt);
2 for (i = 0; i < max_level; i++) {     11 for (i = max_level; i > 0; i--) {
3 this.Replicate(new GLink("right")); 12 this.Go(this.From());
4 this.Replicate(new GLink("left"));  13 if (other_pt = = NULL) {
5 mid_pt=top_pt+(tail_pttop_pt)/2;     14 other_pt = my_pt;
6 if (FromLink().getName=="left'')     15 this.Terminate();
7     tail_pt = mid_pt1;                16 } else
8 if (FromLink().getName=="right'')    17 my_pt = merge( my_pt, other_pt);
9     top_pt = mid_pt1;                 18 }}
```

Fig. 5. Convex hull pseudo-code in Alchemi+

Table 5. Results of convex hull execution on three environments

	N=30			N=50		
N	MPICH	MPICH-G2	Alchemi+	MPICH	MPICH-G2	Alchemi+
1	0.360	0.462	0.377	2.055	2.363	2.158
2	0.258	0.297	0.271	1.104	1.430	1.270
3	0.192	0.238	0.212	0.651	0.984	0.810
4	0.159	0.207	0.190	0.462	0.787	0.693
6	0.134	0.178	0.151	0.344	0.551	0.481
8	0.087	0.155	0.113	0.238	0.503	0.385

5 Conclusion and Future Works

In this article, Alchemi+ system, which provides mobile-agent-based parallel programming on the grid, has been presented. This has been carried out by adding navigational statements to parallel programming by the use of agents on

Alchemi^{TM}. Some of the most important modifications in this system are: The addition of *GAgent* class, from which user programs with inheritance are constructed, and include navigational statements and other agent-based programming tools as well as *GMonitor* class which provides communication between threads. Moreover, by the execution of two algorithms of matrix multiplication and convex hull, the performance of this system has been evaluated.

By evaluation of the results, we can claim that above system has performed better than MPICH-G2 in most cases. The measured speedup of the system in the two foregoing tests has an almost linear and acceptable growth. The following are some of the works which can be done in the future in the same direction as that of the present article:

1. The execution of the full-applied algorithms such as CSG and BSP Tree on Alchemi+ system to evaluate the performance of the system more exactly.
2. Optimization in the synchronization of the mobile agents.
3. Using the performance-measurement tools to evaluate the system.
4. The implementation of algorithmic skeletons on the Alchemi+ system.

References

1. Bic, F., Fukuda, M., Dillencourt, M.: Distributed Computing using Autonomous Objects. IEEE Computer. **29(8)** (1996) 55–61
2. Cabri, G., Leonardi, L., Zambonelli, F.: Weak and Strong Mobility in Mobile Agent Applications. Universit di Modena e Reggio Emilia Via Campi, ITALY (2000).
3. Cao, J., et al.: Agent-Based Resource Management for Grid Computing. 2nd IEEE Int. Sym. on Cluster Computing and the Grid. (2002) 350–351
4. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (1999)
5. Fukuda, M., Tanaka, Y., Suzuki, N., Bic, L., Kobayashi, S.: A Mobile-Agent-Based PC Grid. Active Middleware Services (2003) 142–150
6. Globus project web site, University of Chicago. (2002) http://www.globus.org
7. Luther, A., Buyya, R, Ranjan, R., Venugopal, S.: Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids. U. of Melbourne(2003)

Bridging the Gap Between Cluster and Grid Computing

Albano Alves and António Pina

[1] ESTiG-IPB, Campus Sta. Apolónia, 5301-857, Bragança-Portugal
albano@ipb.pt
[2] UMinho, Campus de Gualtar, 4710-057, Braga-Portugal
pina@di.uminho.pt

Abstract. The Internet computing model with its ubiquitous networking and computing infrastructure is driving a new class of interoperable applications that benefit both from high computing power and multiple Internet connections. In this context, grids are promising computing platforms that allow to aggregate distributed resources such as workstations and clusters to solve large-scale problems. However, because most parallel programming tools were primarily developed for MPP and cluster computing, to exploit the new environment higher abstraction and cooperative interfaces are required. $Ro_cme\mu$ is a platform originally designed to support the operation of multi-SAN clusters that integrates application modeling and resource allocation. In this paper we show how the underlying resource oriented computation model provides the necessary abstractions to accommodate the migration from cluster to multicluster grid enabled computing.

1 Introduction

Many researchers have contributed to important aspects of grid computing allowing applications to share and aggregate distributed resources that cross the cluster boundary. However, because most parallel platforms were built for MPP and cluster computing, programmers still don't have tools to exploit the grid in a straightforward way.

1.1 Distributed Parallel Applications

By its nature, the Grid seems to be the perfect platform to run application systems that integrate multiple cooperative parallel applications distributed across the Internet. Each parallel application should exploit local high performance hardware.

A parallel application should be able to launch other applications or instantiate specific remote components. In addition, components of distinct applications should be able to exchange data through message passing or by accessing remote memory. Parallel applications started independently, eventually running on different computing systems and under the control of different users, should be able to establish some kind of relationship. To achieve cooperation between distinct applications we need mechanisms to represent application entry points and to discover application components.

The concept of application system aims to aggregate collaborative application components spread among multiple architectures and machines. Independently of the resources used and the administrative domains and users involved, the programmer should be able to produce a unified system view of the application system. Moreover, different

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 196–203, 2006.

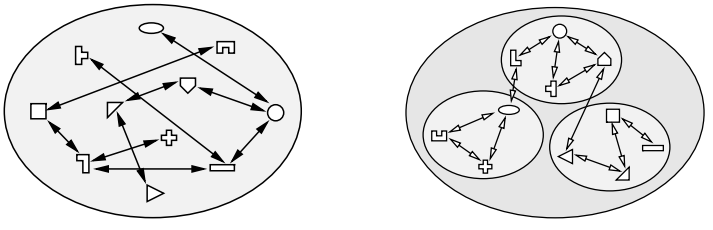
programmers/users would benefit from the existence of multiple distinct views shaped according to their different interests.

1.2 The $Ro_cme\mu$ Approach

$Ro_cme\mu$ combines the facilities of two systems: R_OCl [2] and $m_\varepsilon\mu$ [3].

R_OCl acts as a base layer interfacing low-level communication subsystems generally available in clusters. It constitutes a basic single system image, by providing interconnectivity among communication entities despite their location. Those application entities we name resources may exchange messages despite the underlying communication subsystems. The R_OCl dispatching mechanism is able to bridge messages from GM to M-VIA, for instance. A cluster oriented directory service allows programmers to announce and locate application resources thus turning R_OCl into a convenient platform to drive multi-SAN clusters that integrate multiple subclusters (Myrinet and Gigabit subclusters, for instance) interconnected by multihomed nodes.

$m_\varepsilon\mu$ was implemented over R_OCl and provides higher-level programming abstractions. Basically, it supports the specification of physical resources, the instantiation of logical resources accordingly to the real organization of physical resources and high-level communication operations between logical resources.



(a) R_OCl (b) $m_\varepsilon\mu$

Fig. 1. Resource interaction in R_OCl and $m_\varepsilon\mu$

Figure 1 depicts the main difference between R_OCl and $m_\varepsilon\mu$ abstraction models; while R_OCl (figure 1(a)) provides unstructured communication between resources, $m_\varepsilon\mu$ (figure 1(a)) offers mechanisms to organize resources and exploit locality.

$Ro_cme\mu$, as an overall platform, is well suited to the development of applications aimed to exploit multi-SAN clusters that integrate multiple communication technologies such as Myrinet and Gigabit. The corresponding programming model has proved to be very useful to manage the distinct locality levels intrinsic to that system architecture. The approach provides the necessary abstractions at the application level to accommodate the evolution from cluster to multicluster grid enabled computing.

2 Resource Oriented Communication

R_OCl is an intermediate-level communication library that allows system programmers to easily develop higher-level programming environments. It uses existing low-level communication libraries to interface networking hardware, like Madeleine [4].

2.1 General Concepts

R_OCl defines three major entities: contexts, resources and buffers. Contexts are used to interface the low-level communication subsystems. Resources permit to model both communication and computation entities whose existence is announced by registering them in a global distributed directory service. To minimize memory allocation and pinning operations, R_OCl uses a buffer management system; messages are held on specific registered memory areas to allow zero-copy communication.

Every application should initialize a R_OCl context, register some resources, query the directory to find remote resources, request message buffers, address messages to resources previously found and retrieve received messages from a local queue.

R_OCl includes a fully distributed directory service where resources are registered along with their attributes; a directory server is started at each cluster node and all application resources are registered locally. A basic interserver protocol allows applications to locate remote resources; query requests are always addressed to a local server but, at any moment, servers may trigger a global search by broadcasting the request.

Resources are animated by application threads which share the communication facilities provided by contexts. On the other hand, R_OCl supports the simultaneous exploitation of multiple communication technologies being responsible for selecting the most appropriate communication medium to deliver messages to a specific destination, for aggregating technologies when the target resource of a message can be reached through distinct technologies and for routing messages at multihomed nodes that interconnect distinct subclusters. To provide the above facilities, R_OCl uses a multithreaded dispatching mechanism and includes native support for multithreaded applications.

2.2 Multicluster Operation

The evolution of R_OCl from the cluster environment (multisubcluster) to the grid environment (multicluster) is highly dependent on the capabilities offered by its distributed directory service. The ability to locate resources at cluster level is primarily based on broadcast which does not scale to the grid environment.

R_OCl was improved to include resource location in grid environment by means of a directory proxy installed at each cluster. A directory server unicasts query requests to all known proxies, whenever it fails to obtain replies from servers inside its cluster.

Resource lookup in a multicluster environment comprises the following stages: *(1)* the application sends a request to the server running in the same node, *(2)* that server searches its local database, *(3)* the request is broadcast to all cluster servers, *(4)* those servers search their databases, *(5)* the request is sent to all known proxies that represent the known remote clusters, *(6)* proxies broadcast the requests to servers, *(7)* those servers search their databases and finally *(8)* the replies are sent to the proxy of the cluster where the query process started. The query process may conclude after stages 2 or 4 depending on the location of the resource. It is also possible to specify the scope of a particular query (*node*, *cluster* or *multicluster*) thus bounding the lookup process.

The dispatching mechanism uses a similar approach in order to deliver messages to resources instantiated on nodes from remote clusters. In effect, the directory proxy also acts as a message forwarder; when the dispatching mechanism knows that a resource is from a remote cluster, it sends the message to the proxy of that remote cluster.

3 Unified Modeling and Exploitation

$m_{\varepsilon}\mu$ programing methodology includes three phases: *(1)* the definition and organization of the concepts that model the parallel computer - physical resources -, *(2)* the definition and organization of the entities that represent applications - logical resources - and *(3)* the mapping of the logical entities into the physical resources.

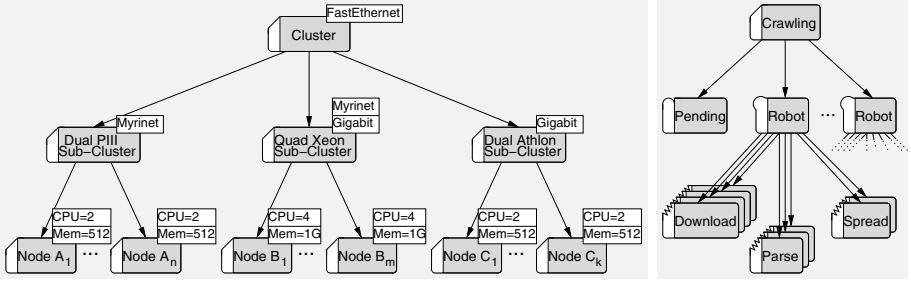
The programing interface is organized around six basic abstractions (see figure 2) designed for modeling both logical and physical resources: *(1)* domains - to group or confine a hierarchy of related entities; *(2)* operons - to delimit the running contexts of tasks; *(3)* tasks - to support fine-grain concurrency and communication; *(4)* mailboxes - to queue messages sent/retrieved by tasks; *(5)* memory blocks - to define segments of contiguous memory; *(6)* memory gathers - to create the illusion of global memory.

Fig. 2. $m_{\varepsilon}\mu$ entities

3.1 Representation of Resources

As an instantiation of the first phase of the methodology, figure 3(a) shows how the physical resources of a parallel machine made of distinct technological partitions may be modeled by a hierarchy of domains. At the top-level, the domain *Cluster* has as its direct descendants three subclusters *Quad Xeon*, *Dual PIII* and *Dual Athlon*. At each subcluster, nodes are modeled by *Node* A_x, *Node* B_x and *Node* C_x.

The second phase is illustrated in figure 3(b) by a high-level specification of a simple distributed web crawler. The crawling process is modeled by a top domain that represents the application whose descendants are a certain number of *Robots* modeled by operon resources. Further refinement introduces three different sorts of tasks (*Download*, *Parse* and *Spread*) to model the three well known crawling stages: *(1)* downloading of pages from the web, *(2)* parsing the pages to obtain new URLs and *(3)* distribution of URLs for future crawling, according to a certain partitioning scheme.



(a) (b)

Fig. 3. Logical and physical resources

Finally, *Pending* models a general global accessible mailbox used by *Download* tasks to store/retrieve pending URLs.

Resources are named and globally identified entities to which we may attach lists of properties. In figure 3(a), *FastEthernet*, *Myrinet* and *Gigabit* are properties that qualify *Cluster* and *Subcluster* domains, while *CPU=2* and *Mem=512* are another sort of properties used to quantify characteristics of nodes A_x and C_x.

$m_{\varepsilon}\mu$ resources accumulate properties inherited from their chain of ancestors. Inheritance allows resource properties to be propagated top-down from ancestors to descendants, while synthesis is the reverse mechanism. $m_{\varepsilon}\mu$ also introduces the concept of resource alias which is a virtual resource node, used as a proxy to one or more existent nodes. In addition to the ancestor and descendants, an alias also has one or more originals, which share with it their own full set of accumulated properties. Aliases are represented as dashed shapes and the dashed arrow coming from each original in the direction of an alias represents the mechanism that allows for the sharing of the original accumulated properties with the target resource (see figure 4).

3.2 Integration of Hierarchies

To map the abstract logical representation, derived from the second phase, into physical resources, programmers should use $m_{\varepsilon}\mu$ primitives. Next, we describe the mapping process used to produce the hierarchy presented in figure 4, which corresponds to the fusion of two hierarchies - the hierarchy that represents the cluster (figure 3(a)) and the hierarchy that represents the application (figure 3(b)).

First, the application identifies the target physical resources - the subclusters *Quad Xeon* and *Dual Athlon*. Then, the alias domain *Crawling* is created to express the effective selection of physical resources. This domain represents a particular view of the available physical resources. Subsequently, operons *Robot* may be launched by specifying the alias *Crawling* as the target. The $m_{\varepsilon}\mu$ runtime system will automatically select a node among the nodes of subclusters *Quad Xeon* and *Dual Athlon*, instantiate an operon on that node and create an alias under the domain *Crawling*.

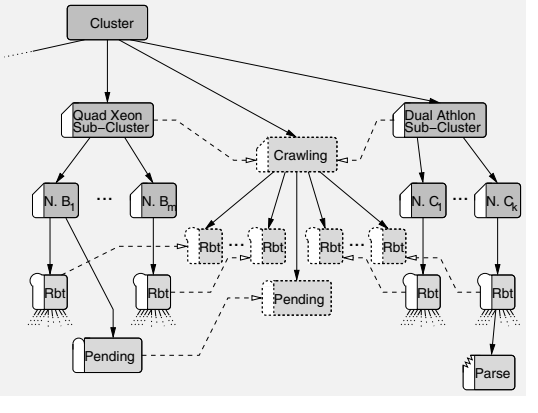


Fig. 4. Resource mapping

It is important to note that the integration of hierarchies preserves the original organization of the logical hierarchy through aliases but introduces a system view which combines physical and logical resources. The system view does not take into account aliases but allows to reach any component without knowing the application structure.

3.3 Multicluster Exploitation

In a grid environment, the physical hierarchy of the application example depicted in figure 4 will be extended to include, at least, an additional top-level domain. The new root domain would have multiple descendant domains representing each of the clusters.

The multicluster environment enforces the need to control the access to resources. $m_{\varepsilon}\mu$ allows to specify access control lists for each resource which are used to validate *(1)* the creation of descendants and aliases and *(2)* the access to resource properties. The selection/allocation of physical resources will thus be denied for unauthorized users as well as the access to logical resources (messaging and memory access).

$m_{\varepsilon}\mu$ offers high-level structured message passing that allows any received message to be implicitly broadcast to the totality of the active descendants of a logical domain. This is a valuable feature of the resource oriented paradigm particularly suited for a cluster environment where broadcast can be efficiently implemented at the hardware/software level. In a multicluster environment programmers must be aware of potential bottlenecks caused by logical domains that traverse distributed clusters.

Resource selection, through the creation of an alias that aggregates multiple physical domains, requires several $m_{\varepsilon}\mu$ lookup operations. Since those operations are implemented using global R_OCl queries, the use of this functionality must be occasional.

4 Deploying Applications

$Ro_cme\mu$ constitutes a very simple approach for the development and execution of applications in some particular grid scenarios - multicluster systems where the nodes at each cluster may be interconnected by Myrinet, Gigabit and/or FastEthernet. Cluster administrators are responsible for installing $Ro_cme\mu$ libraries and services while programmers use $m_{\varepsilon}\mu$ programming interface to develop applications. Users are also responsible for defining a virtual cluster to support the execution of the application.

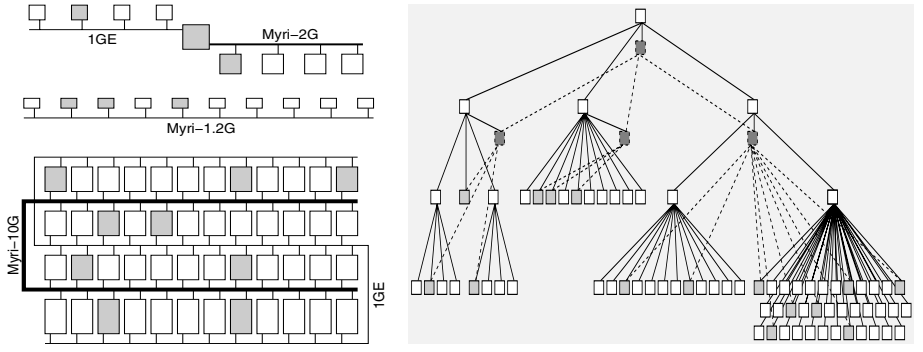
4.1 Virtual Clusters

Users may define virtual clusters – subsets of the totality of multicluster resources – using a web portal where the nodes of each cluster are graphically represented. Figure 5(a) depicts the reservation of 15 nodes spread through three clusters.

Based on user's selection, *(1)* the system manages to create a global user account, *(2)* the $m_{\varepsilon}\mu$ physical hierarchy is updated to include the right access control lists and *(3)* some $m_{\varepsilon}\mu$ aliases are created in order to produce a suitable view (see figure 5(b)).

4.2 Multicluster Wide Access

In a multicluster environment, the first problem to solve is TCP/IP connectivity, because of private networks, firewalls, etc. Our approach uses OpenVPN to provide full connectivity among multicluster nodes despite the composition of each virtual cluster.



(a) User selection (b) Corresponding $m\varepsilon\mu$ hierarchy

Fig. 5. Creating a virtual cluster

Since $Ro_cme\mu$ allows to dynamically start application components, we also needed to extend *rsh/ssh* to support multicluster operation. $Ro_cme\mu$ maintains a pool of shadow accounts at each cluster and thus the virtual global account created when the user reserves multicluster nodes is mapped into multiple (real) shadow accounts.

Finally, access to binaries and application data requires some kind of multicluster file system. Our approach is to extend NFS by introducing appropriate proxies. This idea has already been investigated and deployed in the context of PUNCH [7], a wide-area networking computing environment.

5 Discussion

In the last decade PVM and MPI have dominated the field of parallel computing. The paradigm still remains the main choice for grid programming as attested by current ports like MPICH-G2 [8] and other platforms to run unmodified PVM/MPI applications [9].

Message passing is more adequate than the connection based approach for exploiting the parallelism of distributed memories in grid and web services [6]. However, fine grain synchronization is not easily achieved in a grid environment, meaning that grids cannot be programmed as flat huge machines made of a larger number of processors.

$Ro_cme\mu$ offers programmers the possibility of structuring the communication between resources in a multicluster environment following the same principles that lead to the organization of a cluster into subclusters. Taking into account the hierarchy of a multicluster grid and the available interconnection technologies, programmers may take advantage of locality at: *(1)* SMP nodes, where processors share memory, *(2)* subclusters, where nodes are interconnect by a sole high performance communication technology, *(3)* clusters, where subclusters are interconnect by multihomed nodes and *(4)* multicluster grid, where clusters are interconnected by Internet links.

Programmers need the power of grids to build and deploy applications that break free of single resource limitations to solve large-scale problems. Although much work have been done in the context of Globus [5] to exploit grid resources, $Ro_cme\mu$ offers as a unique feature the integration of: physical resource specification, application

modeling and logical components instantiation, providing much of the operations identified in [1] as key functionality for developing grid applications.

References

1. G. Allen, T. Goodale, M. Russell, E. Seidel, and J. Shalf. *Grid Computing: Making the Global Infrastructure a Reality*, chapter Classifying and enabling grid applications. John Wiley & Sons, Ltd., 2003.
2. A. Alves, A. Pina, J. Exposto, and J. Rufino. RoCL: a Resource oriented Communication Library. In *Euro-Par 2003*, number 2790 in LNCS, pages 969–979. Springer, 2003.
3. A. Alves, A. Pina, J. Exposto, and J. Rufino. $m_{\varepsilon}\mu$: unifying application modeling and cluster exploitation. In *SBAC-PAD'04*, pages 132–139. IEEE Computer Society, 2004.
4. O. Aumage, L. Bougé, A. Denis, J.-F. Méhaut, G. Mercier, R. Namyst, and L. Prylli. Madeleine II: A Portable and Efficient Communication Library for High-Performance Cluster Computing. In *CLUSTER'00*, pages 78–87, 2000.
5. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *HPDC'01*, pages 181–194, 2001.
6. G. Fox. Messaging Systems: Parallel Computing the Internet and the Grid. In *Euro-PVM/MPI'03 (invited talk)*, 2003.
7. N. Kapadia, R. Figueiredo, and J. Fortes. Enhancing the Scalability and Usability of Computational Grids via Logical Accounts and Virtual File Systems. In *HCW'01*, 2001.
8. N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Parallel and Distributed Computing*, 63(5):551–563, 2003.
9. D. Royo, N. Kapadia, and J. Fortes. Running PVM Applications in the PUNCH Wide Area Network-Computing Environment. In *VECPAR'00*, 2000.

A Broker Based Architecture for Automated Discovery and Invocation of Stateful Services*

Marian Babik and Ladislav Hluchy

Dept. of Parallel and Distributed Computing, Institute of Informatics,
Slovak Academy of Sciences
Marian.Babik@saske.sk, Ladislav.Hluchy@savba.sk

Abstract. Web Service Resource Framework (WSRF) is a recent effort of the grid community to facilitate modeling of the stateful services. Semantic Web Services initiative is proposing a standard called OWL-S to help the automated discovery, composition and invocation of the services. This article describes a broker based architecture, which tries to combine WSRF and OWL-S in a way to enable automated discovery and invocation of the stateful services. Analysis of the requirements are provided and an application scenario is described, which has been used to evaluate the architecture.

1 Introduction

Recently, Web service (WS) technologies are gaining importance in the implementation of the distributed systems, especially grids. One such example is the Web Service Resource Framework (WSRF) [4], which extends the current WS technologies by modeling the stateful services. Design and development of the service oriented distributed system is quite common and there are several emerging WS initiatives, which tries to automate the process of discovery, composition and invocation of services. The semantic web services are a typical example, showing the potential of how ontological modeling can improve the shortcomings of the service oriented computing.

In this paper we propose the semantic description of the stateful web services introduced by the WSRF specifications. We also describe a corresponding distributed broker based architecture for discovery and invocation of both stateful and stateless services. We provide a brief overview of the semantic web services specification (OWL-S) and show how it can be used to model the basic stateful services as described by the WSRF.

* The research reported in this paper has been partially financed by the EU in the project IST-2004-511385 K-Wf Grid and Slovak national projects, VEGA No. 2/3132/23: Effective tools and mechanisms for scalable virtual organizations and Tools for acquisition, organization and maintenance of knowledge in an environment of heterogeneous information resources 2004-2007 SPVV 1025/04.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 204–211, 2006.

2 Semantic Web Services

One of the challenges of the loosely coupled distributed systems is the ability to dynamically discover and integrate the services needed by the applications. Interoperability among services is especially important in the distributed environments hosting a large number of services, i.e. grids. Semantic descriptions facilitates the process by expressing the characteristics of the service, which can later be used for their discovery, composition and invocation. The rich semantics needed for such descriptions can be provided by the ontologies.

OWL-S is an ontology-based approach to the semantic web services [1]. The structure of the ontology consists of a service profile for advertising and discovering services, a process model which supports composition of services, and service grounding, which associates profile and process concepts with underlying service interfaces. Service profile (OWL-S profile) has functional and non-functional properties. The functional properties describe the inputs, outputs, preconditions and effects (IOPE) of the service. The non-functional properties describe the semi-structured information intended for human users, e.g. service name, service description, and service parameter. Service parameter incorporates further requirements on the service capabilities, e.g. security, quality-of-service, geographical scope, etc. Service grounding (OWL-S grounding) enables the execution of the concrete Web service by binding the abstract concepts of the OWL-S profile and process to concrete messages. Although different message specifications can be supported by OWL-S, the widely accepted Web Service Description Language (WSDL) is preferred [2].

Other frameworks for semantic descriptions of the services, such as WSMF [16] and IRS [17], provide more elaborate service descriptions. Since we have focused mainly on the discovery and invocation, OWL-S was sufficient for our purposes. We have also considered the level of implementation and the number of tools available for WSMF and IRS.

3 WSRF

WS-Resource Framework (WSRF) is a set of specifications, which are based on the concept of modeling state as stateful resource and codify the relationship between Web Services (WS) and stateful resource in terms of a set of conventions on current (i.e. stateless) WS technologies [4]. A stateful resource is defined as having specific state data expressible as an XML document and a well defined life-cycle. It can be acted upon by one or more Web services, e.g. files in a file system or rows in a relational database are considered a stateful resource. Fig. 1 shows a sample implementation scenario for the stateful service in the globus toolkit environment (GT4) [5].

In the example a virtual bookstore, which can buy and sell books is modeled. A sample stateful resource is a book, which is defined by various resource properties, e.g. title, author, price, etc. Adding a particular book is performed by contacting a service factory, which creates an instance of the resource (binding of

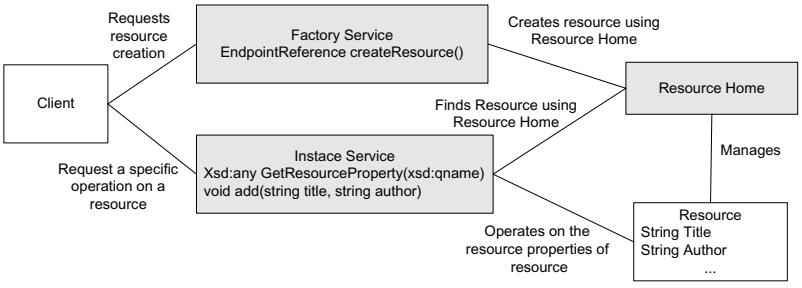


Fig. 1. Sample implementation scenario for the WS-Resource

service and particular resource) and returns an identifier, i.e. so-called endpoint reference. The client can interact with the given resource by calling the methods exposed by the service instance interface. The pairing of the service instance and the resource is called a WS-Resource. There can be various different interfaces for the same resource, e.g. for buying books, selling books, listing books, etc. The actual resource instances are homed by the Resource home, which can be implemented as either persistent or transient store. Detailed description of the WSRF is beyond the scope of this paper; for additional details see [4].

4 Overview

In this section we provide an overview of the requirements of the broker based architecture motivated by the OWL-S Broker [13]. Essentially, broker based communication involves three parties, namely, provider of the services and resources, a client (requestor) requesting realization of its goals, and broker, which mediates the communication between provider and requestor.

The proposed broker is focused only on the automated discovery and invocation of services. The operations which should be performed by the broker include (1) the interpretation of the capabilities of the services, advertised by the provider, (2) interpretation of the requests from the clients, (3) discovery of the service, both keyword and semantic based, and (4) execution of the corresponding services.

In OWL-S, the service capabilities are described by the corresponding IOPE (inputs, outputs, preconditions and effects). Such description can be partly annotated from the WSDL description of the service and is sufficient for the stateless service. Stateful services, i.e. WS-Resources, however, are composed of a service and a stateful resource. Since resource as well as corresponding service are well defined, it is possible to semi-automatically map the resource description with a domain specific ontology. This can enable automated discovery and execution for the stateful services. A possible ontological mapping supporting OWL-S and WSRF is described in the next section.

5 Service Discovery

In our system, the process of service discovery has three parties, client, provider and broker. The provider is publishing the profile's functional and non-functional properties, called advertisements, to a broker. The client queries the broker for the given functional and non-functional properties and receives descriptions of all the relevant services. Service registry is comparing the request to the advertisements. This is called matchmaking of services. Since requests usually do not match exactly the advertisements, it is necessary to derive the corresponding match based on the subsumption, e.g. service reserving places in Cantonese restaurant should also match queries for Chinese restaurant. There are different ways of performing semantic matching of services [15]; however, in our case we'll only consider hierarchy of classes, properties and explicitly defined equivalence. In the implementation we have defined the degrees of matching by subsequent matching rules. The rules are based on the subsumption of classes, properties and also consider explicitly stated equivalence, i.e. the OWL construct owl:sameAs.

We have designed a simple ontology, which describes WS-Resource, i.e. resource, resource properties and its association with instance service and service factory, see Fig. 2. A resource class was defined having one or multiple instances of resource properties. The instances of resource properties are described by the identification (xsd:qname) and association to a relevant domain ontology concept. The resource instance has two object properties, which map the given resource to a service instance and a service factory. Since resource properties are already defined as an XML Schema, the ontological description can be created semi-automatically. The datatype mappings for the WS-Resource is described in the next section. It should be noted that both ontological and datatype mappings describe only the resource properties types, the actual instances of resource properties are implementation dependent.

In the process of implementation, there were several well known issues in the OWL-S profile, which were related to the OWL language and the limitations of

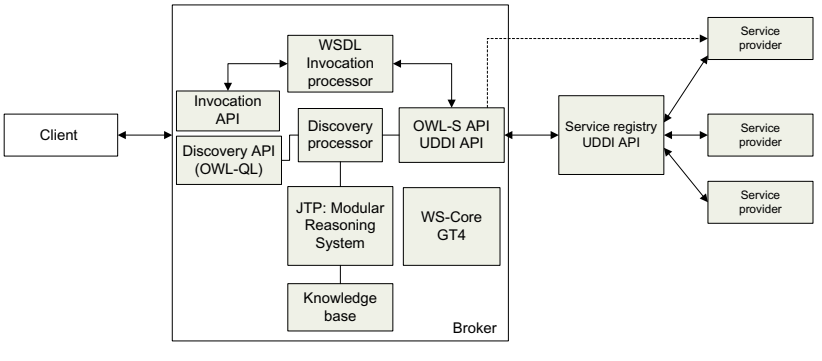


Fig. 2. A simplified schema of the ontological model (right) and datatype mapping (left) for WSRF services

the description logics in the description of services. A good survey of the OWL-S shortcomings is described in [8]. One of the major issues in the profile description is the conditional model. Since conditions refer to the concrete parameter instances, which are not known before execution, it is necessary to define the variables.

The latest version of OWL-S is addressing the issue and specifies the conditions by providing a class *Expression*. *Expression* is defined by the language used, e.g. SWRL, RDF [3], KIF [6], and the actual expression. Based on the language, the expression is incorporated as either XML Literal or String Literal. In our case we have used KIF for expressing conditions, therefore it was necessary to execute the given KIF expression on the instances of the service profiles.

6 Service Invocation

Service grounding for the stateful service differs in the description of the resource properties document, which is specified in XML Schema [10]. It is thus essential to define a datatype mapping between XML Schema and OWL-S. Datatype mapping in OWL-S grounding is based on the RDFS and eliminates previous shortcomings, which were based on the XSLT approach. Currently, a mapping between XML schema datatypes and OWL, is based on semantics and uses a sample RDFS ontology, as described in [8]. The mapping is depicted in Fig. 2 . It can be seen that the mapping reflects the nested data structures of the XML Schema. XML Schema SimpleType is mapped to a corresponding OWL type, which is bound to the datatype property of the OWL-S grounding. Together with complex and array types, a mapping through RDF property is used to map the elements to XSDTypes.

The basic WS-Resource has a well defined interface for accessing and modifying its resource properties and their values. This interface has three main methods, namely, getResourceProperty, setResourceProperty and getMultipleResourceProperties. The datatype mapping for the stateful services is based on mapping the inputs and outputs of these methods to a datatype property of the domain concept. The actual mapping is based on the RDFS mapping schema, which is complemented by the ontological mapping to the *Resource* class. Schematic representation of the process is shown in Fig 2.

7 Architecture

The architecture is shown in Fig. 3. The overall schema has three main components, namely, the client, the broker and the service providers. The broker's role is to mediate client requests for the service capabilities and match them with the description received from the service providers. The broker has knowledge base (KB) and reasoning engine based on the JTP [9], which is used to store the service profiles and groundings and to perform the reasoning necessary for the process of matchmaking. The process of matchmaking is coordinated by the discovery processor, which receives the client's requests and queries the KB to

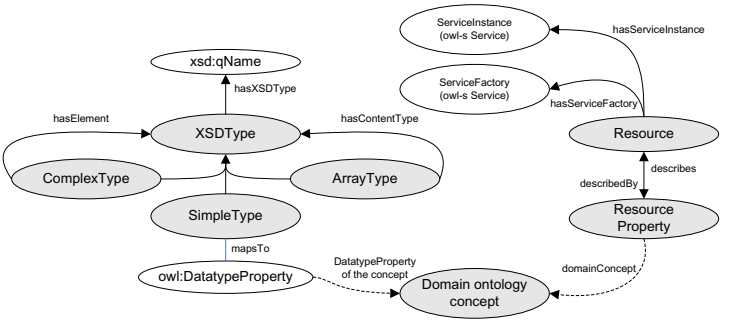


Fig. 3. Main components of the broker based architecture

find the relevant results. It is also used to process the descriptions from the service registries. In order to coordinate parsing of the description of the stateful service, access to the GT4 WS-core libraries is necessary [5]. The actual invocation of the service is performed by the WSDL invocation processor, which is also interconnected with the GT4 WS-Core.

The broker needs to interact with the service providers to get the WSDL and OWL-S specifications. This is accomplished by using the UDDI standard for service registries [7]. The corresponding mapping from OWL-S to UDDI was based on [12] and extended to support the stateful service descriptions. The client is accessing the broker through discovery and invocation API. Discovery API supports semantic matchmaking as well as keyword based lookups, which are implemented using the native UDDI API. Further, in order to access the OWL-S ontologies directly, the client can use the OWL-QL language to query the knowledge base [11]. This is supported by the fact that in the current implementation OWL-QL is transformed into a KIF query, which is native to our reasoning engine. The invocation API combines the OWL-S API [1] and GT4 invocation methods [5].

8 Application Scenario

The Flood Forecast Simulation Cascade as shown in Fig. 4 is a hydrometeorological application, already well tested and developed during the project CrossGrid [14]. The application is a cascade (a natural workflow) of several simulation stages, which leads to the final result - a prediction of a potential flood. The cascade begins with meteorological prediction of weather for a short future period. This prediction is then recomputed into possible watershed, which in turn affects the water level of the target river. This water level is computed in the hydrological stage of the application. The resulting hydrograph (time series of water level values) is fed into the final stage of the cascade - a hydraulic prediction. This - using detailed terrain model of the target flooded area - computes the water flow in the target area, water depth and flow vectors.

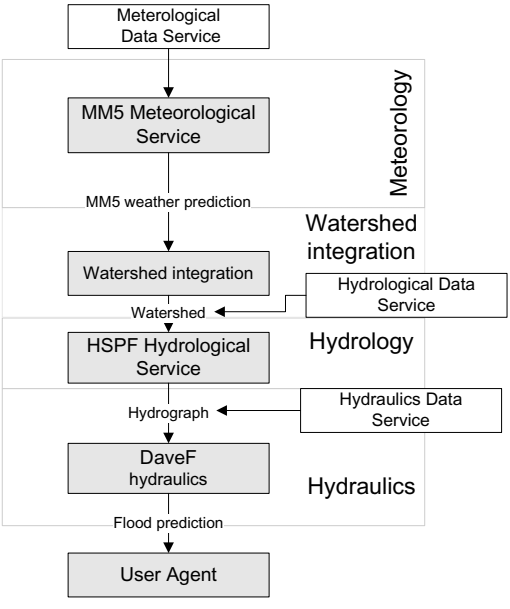


Fig. 4. Flood prediction scenario showing both stateful and stateless services

In our application scenario the meteorological, hydrological and hydraulic methods are simple web (stateless) services. The inputs and outputs of the services are associated with the domain ontology concepts. In contrast, meteorological hydrological and hydraulic data services are stateful services, which are storing resources (in this case files in a file system). The services are also described by the non-functional parameter, i.e. geographic scope, which corresponds to the geographical scope of the data they are able to provide. The resource properties, i.e. descriptions of the file content, are modeled by the domain ontology. The goal of the discovery is to find the appropriate stateful services for a given region providing data in the required data format (specified by the resource properties). The subsequent invocation procedure executes the best suitable instances of services. Although only a static workflow is assumed, employing the OWL-S process model can enable more complex and dynamic workflows.

9 Conclusion

We have described a broker based architecture for automated service discovery and invocation. We have shown a flood prediction application scenario, that we have used to evaluate the initial implementation. In the future we would like to extend our work to consider OWL-S process model as well as further WSRF specifications, such as WS-Notification, WS-Security, WS-GRAM, WS-transfer, etc.

References

1. A. Ankolekar et.al, OWL-S: Semantic Markup for Web Service, 2003, http://www.daml.org/services/owl-s/1.1
2. E. Christensen, F. Cubera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, Technical report, WWW Consortium, 2001, http://www.w3c.org/TR/wsdl
3. Resource Description Framework, http://www.w3.org/RDF/
4. Web Service Resource Framework, http://www.globus.org/wsrf/
5. Globus Toolkit, http://www-unix.globus.org/toolkit/
6. Knowledge Interchange Format, draft to American National Standard, 1998, http://logic.stanford.edu/kif/dpans.html
7. Universal Description, Discovery and Integration Protocol Specification, UDDI Consortium, 2001, http://www.uddi.org/specification.html
8. Pitfalls of OWL-S – A practical Semantic Web Use Case, S. Balzer and T. Liebig and M. Wagner, Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC004), 2004
9. JTP: A System Architecture and Component Library for Hybrid Reasoning, R. Fikes, J. Jenkins, and G. Frank, Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics. Orlando, Florida, USA, 2003
10. Extensible Markup Language (XML), XML Protocol Working Group, 2004, http://www.w3.org/TR/2004/REC-xml11-20040204/
11. R. Fikes, P. Hayes, I. Horrocks, OWL-QL: A Language for Deductive Query Answering on the Semantic Web, KSL Technical Report 03-14, 2003
12. N. Srinivasan, M. Paolucci and K. Sycara, Adding OWL-S to UDDI, implementation and throughput, First International Workshop on Semantic Web Services and Web Process Composition, (SWSWPC 2004) 6-9, 2004, San Diego, California, USA
13. M. Paolucci, J. Soudry, N. Srinivasan, and K. Sycara, Untangling the Broker Paradox in OWL-S, Proceedings of AAAI 2004 Spring Symposium
14. CrossGrid consortium, CrossGrid Technical Annex, 2004, http://www.crossgrid.org
15. L. Li and I. Horrocks, A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth Inl. Word Wide Web Conference (WWW 2003), pp. 331-339, ACM, 2003
16. D. Fensel and C. Bussler. The Web Services Modelling Framework. In Proc. of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises April, 2002
17. J. D. E. Motta, L. Cabral, and M. Gaspari IRS-II: A Framework and Infrastructure for Semantic Web Services. In Proc. of the 2nd Int. Semantic Web Conference (ISWC2003), Sanibel Island, USA October, 2003

Remote Parallel I/O in Grid Environments

Rudolf Berrendorf[1], Marc-André Hermanns[2], and Jan Seidel[1]

[1] Department of Computer Science,
University of Applied Science Bonn-Rhein-Sieg,
53754 St. Augustin, Germany
[2] Central Institute for Applied Mathematics,
Research Centre Jülich,
52425 Jülich, Germany

Abstract. Although processor speed and memory bandwidth as well as capacities of persistent storage devices evolved rapidly in the last years, the bandwidth between memory and persistent storage devices could not match that pace. As current scientific applications tend to process an enormous amount of data at runtime, the access to a local disk might become the main performance bottleneck.

The communication infrastructure for local area networks has also evolved rapidly, thus modern filesystems for supercomputing are using storage area network solutions to distribute the load of application I/O to several special purpose I/O nodes. These SAN solutions, however, are often bound to a specific organizational structure, such as different locations of the same company or several institutes of a university. This often implies a common user base and accounting information at each site. In a highly variant grid environment these demands might be hard to meet.

This paper describes the definition of two ADIO devices for ROMIO, a publicly available MPI I/O implementation, to provide transparent access to remote parallel I/O and additionally access to remote I/O on files in the memory of a remote server. The architecture of these devices allows for a definition of remote servers on a per job basis, and can be configured by the user before runtime.

1 Introduction

In 1996, the MPI forum defined an extension to the existing message passing interface [1], formulating a standardized API to I/O for scientific applications. With its definition, the MPI I/O extension decouples problems of I/O with distributed data from the user application, and introduces a new layer for the user's view of a file.

Modern applications in computational science fields have a growing need for computer resources. Not only do they need more computational power to meet greater challenges, they also need the I/O infrastructure to efficiently read and write the computed data to and from persistent storage. The data used by the application is, when loaded into memory, often scattered among the processes to be able to increase the problem size.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 212–219, 2006.

This implies that, in order to save this data to disk, the filesystem has to be shared with all nodes or the application itself has to deal with different strategies to efficiently store the data local to one of the nodes. On a very static configuration of one or two clusters a shared filesystem may still be practical, but in a highly variable grid of different clusters, a shared filesystem among different sites may become an administrative challenge.

Traditionally clusters have a shared filesystem using NFS. But NFS was not designed to fit the needs of access patterns for scientific applications. Multiple concurrently accessing processes on a single file need a parallel filesystem to support that kind of concurrency. For example overall throughput can be increased by using multiple independent communication channels. This is one of the reasons why filesystem in scientific computing move on towards storage area networks and parallel filesystems like PVFS2 [2], LUSTRE [3], CLUSTERFILE [4] or GPFS [5]. Multiple I/O servers handle I/O requests by multiple clients in parallel and share workload among the servers.

Each of these filesystems needs some kind of system level configuration, before it can be used, though. The EU project DEISA [6] for example is deploying a distributed GPFS parallel filesystem between several european IBM clusters. With a distributed filesystem a supercomputer inherits several constraints like a common user database within the grid, so this is usually only practical in a fairly static cluster environment with closely cooperating sites. On clusters that are only coupled for one specific job or application run, this might be too much overhead in administration. In the CLUSTERIX project [7], e.g. it is planned to easily integrate many different clusters on a very fluctuating basis. A static distributed filesystem over these clusters is not practical. Here a flexible interface to the different filesystems in the grid is needed.

Besides parallel I/O to increase performance and reduce execution time of the application, another challenge in a highly variant grid environment is the location of data. In grid middleware like GLOBUS [8] or UNICORE [9], the data is copied to the desired location before and/or after the job execution. If the application needed only a small portion of a larger file, this might lead to significant overhead in job preparation.

But persistent storage is not the only need of scientific applications. During an application run, the need to migrate large amounts of data temporarily to disk can also arise. Normally applications would use a temporary filesystem local to a node to store this data, but these filesystems are usually not shared for performance reasons. With modern high-bandwidth LAN and WAN interconnects collective communication to distant nodes in the cluster can be faster than a transfer to a shared storage device, as shared resources involve data transfer anyway. To save the time needed to write the data to disk, it can be stored much faster, if the remote site keeps them in main memory, i.e. a main memory based filesystem. This implies that the amount of data stored is more restricted than for disk based filesystems but today larger clusters have tens to hundreds of gigabytes of main memory available.

2 Virtual Filesystems for Transparent I/O

The VIOLA testbed [10] interconnects several partners with a dedicated high bandwidth optical backbone providing multi-Gbits/s bandwidth. Although the optical network provides a high bandwidth the latency is bound mainly by the geographical distance between the different sites. The core backbone used for a test compute grid of four geographically distant clusters is located within a 100 km radius. The resulting latency has to be taken into account when designing client-server applications using this network.

The clusters at the distributed sites are multi-purpose linux clusters based on AMD Opteron and Intel CPUs with different local network topologies, forming a slightly inhomogeneous network of clusters. To support this inhomogeneity, METAMPICH [11] is used to provide a common MPI middleware. METAMPICH itself is based on the commonly available MPICH [12] MPI implementation for linux. It is designed to interconnect several inhomogeneous clusters to provide a common user view on the grid.

To form a global cluster with different communication interconnects, the solution with the basic MPICH would be to use the `p4` communication device, using TCP/IP as the underlying protocol, as TCP/IP will probably the least common denominator. This would not honor any local communication interconnect, which might support faster non-TCP/IP-based protocols for intra-cluster communication. METAMPICH defines special routing processes to enable intra-cluster communication over the fast local communication device and inter-cluster communication routed over those special processes, using TCP/IP over the WAN interconnect. Although the special router processes imply a small overhead in latency, asynchronous data transfers can perform better than a direct TCP/IP link to the distant process, as the routers can usually be reached with the fast local interconnect.

Being based on MPICH, METAMPICH also supports MPI I/O based on the ROMIO [13] MPI I/O implementation, developed and available from Argonne National Laboratory. Similar to MPICH, ROMIO defines several independent devices for the supported filesystem, to deal with filesystem specific data handling. On top of these low level devices is a common layer called ADIO (Abstract Device Interface for I/O) [14], which again serves as the API for the MPI implementation. To support new filesystems, only the creation of a corresponding ADIO device is needed, and eventually minor changes to the ADIO and MPI layer.

2.1 Remote I/O on Harddisks

The parallel I/O project within VIOLA deals with utilizing the provided optical network for remote parallel I/O. As a static shared filesystem is not feasible in a highly variant grid, our approach extends the existing middleware to provide transparent parallel I/O on remote servers without explicit involvement of distributed parallel filesystems on operating system level.

The TUNNELFS ADIO device redirects a local MPI I/O call and its corresponding data to the remote server instead of issuing a local I/O call. It uses a client-server message exchange pattern to coordinate the data exchange and uses MPI

function calls for data manipulation and is therefore able to use the provided MPI datatypes sending the data to the remote server. With the use of MPI functionality, data buffers do not need to be modified on client or server end, other than it is done by the MPI function anyway. This enables the use of optimized communication functions for sending non-contiguous data buffers, already existing in the MPI implementation. Additionally MPI derived datatypes and file views defined on the client side can be reused on the server side.

On the receiving end of the communication, the TUNNELFS I/O server then issues the I/O call itself, working as a proxy to the client. The TUNNELFS I/O server naturally has to use a different communicator for the MPI file handle, but file-views and datatypes can be reused on the server side. Figure 1 shows the layers of MPI and ADIO as they are used by the client and server.

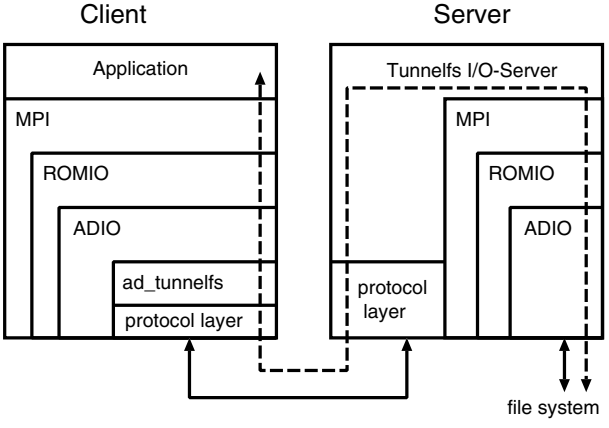


Fig. 1. Layer model of the TUNNELFS device

With I/O clients and servers being processes in the same MPI environment, separate I/O server processes have to be started with the application. Through the integration of the I/O server binaries with the METAMPICH middleware, this can be done transparently for the user. In this approach a node running I/O server processes in one application run can be used for normal computation purposes in the next run. This results in a highly flexible grid environment, where the user can declare the amount of I/O servers to support the I/O needs of his application.

2.2 Remote I/O on Memory

When dealing with I/O hierarchies and their performance, I/O to disk implies the most penalties to application run time. Therefore I/O to persistent storage is avoided where possible. As the problem sizes increase, memory local to a process is a precious resource and usually is very limited for temporary storage of buffers. Thus applications are often forced to write temporary data to disk, to be used again later in the application run.

With the available bandwidth of modern and especially future wide area networks, another solution for this type of temporary data I/O is amenable. Instead of writing to a local disk, the use of (multiple) wide area network links and a distributed remote main memory filesystem for high speed I/O can deliver the requested high I/O rates.

The MEMFS ADIO device implements a user-level, main memory based file system for non-persistent data within an MPI environment. While the current implementation of the MEMFS device is limited to a single node, future versions will implement a multi-server distributed file system across node boundaries.

3 Integration

When developing MPI applications the user normally starts with a global communicator `MPI_COMM_WORLD`, containing all MPI processes in the environment. As I/O clients and servers are present in the same global MPI environment, either the user application has to have explicit knowledge about the server processes, to exclude them when communicating with other user clients, or the servers must not appear in the global view of the user's application. To enable the transparent use of our remote I/O, we decided for the latter approach. From within the `MPI_Init` calls, the reference to the global communicator is saved in a separate handle, called `TUNNELFS_COMM_WORLD`. A new communicator is created containing either only clients or only servers. This new communicator is then used as a reference for the `MPI_COMM_WORLD` handle in the user application. The communication between I/O clients and servers is handled with the former global communicator through the special reference, which is implicitly known to the client ADIO device and the server process. The client itself uses the newly created handle, redefined to `MPI_COMM_WORLD` by including a special header file. The domains of the communicators used are shown in figure 2. The arrows depict the flow of data for a user process to the I/O server. If the server is within the local metahost, the client can directly send the data to the I/O server. If the I/O server is located on another metahost, the client sends it to its local corresponding router process, and it gets forwarded from there.

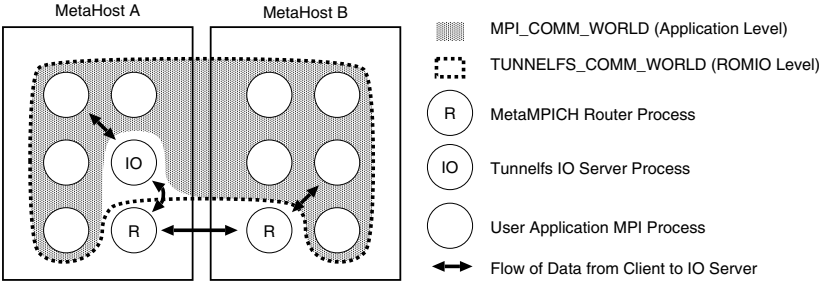


Fig. 2. MPI processes and their affiliation with global communicators

In MPICH the user is able to specifically select an ADIO device through a filename-prefix like "nfs:" or "ufs:". The two new ADIO devices developed in VIOLA parallel I/O use the same scheme. They are selected by the prefixes "tunnelfs:" for the remote I/O and "memfs:" for the I/O to memory device. Each device in ADIO truncates its own prefix from the filename before further usage. The TUNNELFS device does the same, sending only the modified filename to the remote server. There a separate MPI I/O call is issued using the transfered filename, also recognizing a valid filesystem prefix. Therefore the user can also specify a specific filesystem on the server by combining the TUNNELFS prefix with another prefix. To specify remote I/O to memory the user can therefore use the combined prefix "tunnelfs:memfs:".

The TUNNELFS device currently supports remote datatype definition for MPI derived datatypes. With tracking local datatypes and their remote definition, the server is able to set its local file view accordingly to the user defined file view for a specific process. The user can access any file on a filesystem local to the remote I/O server, provided the filesystem is accessible by the MPI subsystem.

4 Results

The current prototypes of TUNNELFS client-device and server are able to handle a single I/O server in the MPI environment. We have successfully tested intra-cluster I/O without a preconfigured shared filesystem. The target filesystem can be chosen to be any filesystem supported by the ADIO layer.

The current version of the MEMFS device supports accesses only to the local main memory the ADIO device runs on. In future versions the MEMFS device will implement a distributed main memory based file system between multiple MEMFS servers that distribute data and load.

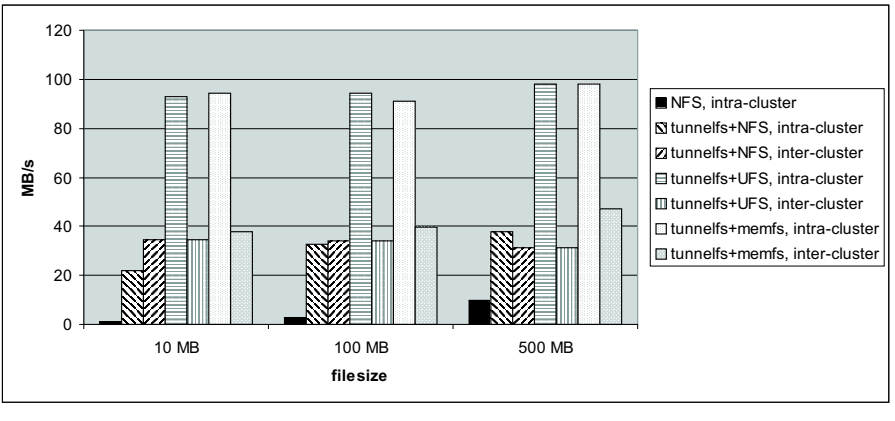


Fig. 3. Bandwidth with 12 I/O client processes and 1 I/O server on a shared file

To evaluate the TUNNELFS and MEMFS we ran an MPI benchmark program that measured 8 different MPI I/O operations. Results are given as average bandwidth numbers over these operations. We ran our tests on 2 clusters connected with a dedicated 100 km distance 10 GigE fibre network and 1 GigE network cards in each cluster node connected to a cluster switch. All cluster nodes are 4-way SMP nodes with 10.000 rpm SCSI disks accessed locally on a node by a UFS device, and an NFS-mounted RAID-5 based shared filesystem hosted by a cluster file server.

Performance results as given in figure 3 show that with TUNNELFS on the selected bundle of MPI I/O operations up to 50% of the nominal network bandwidth (GigE for inter-cluster as well as Myrinet for intra-cluster) can be sustained. Currently, bandwidth other than for NFS is limited by the network adapter (1 GBit) on the single server node rather than by an I/O device. With multiple TUNNELFS I/O servers in the future, this bottleneck will be eliminated. The MEMFS device accessed locally on a node as a usual local ADIO-device reaches several GB/s bandwidth.

5 Related Work

The problem to access data directly on a remote site, for example from a central file server, was already addressed in 1997 with the RIO filesystem, and was continued by the work on RFS [15], both of which use TCP/IP to access the remote server. Although TCP/IP is available to almost all modern clusters, it might not always be the most efficient way to communicate. Almost all cluster or interconnect manufacturers supply an MPI implementation that can make use of special communication hardware, without using the generic TCP/IP interface. Using these special communication interfaces will usually result in improved performance.

6 Conclusion and Future Work

We defined a communication protocol between I/O client and server based on MPI point-to-point messages and implemented two virtual filesystem devices in ROMIO for remote parallel I/O and I/O to main memory. The filesystem device for remote parallel I/O can be used to tunnel I/O requests to a remote server, and to be issued at the remote location. At the remote location any other supported filesystem but TUNNELFS itself can be used. The MEMFS filesystem device for I/O to main memory can be used with or without the TUNNELFS device. Both devices support most of the MPI I/O calls defined. Almost all MPI I/O functions of MPI-2 but remote error handling are currently supported.

Current development efforts concentrate on the multi-server environment. The I/O servers will be aware of other I/O servers in the environment, and if suitable share workload and data automatically. The workload sharing for the TUNNELFS device will be implemented on the level of MPI file views, whereas the MEMFS device will also support other data distribution methods.

References

1. MPIF, M.P.I.F.: MPI-2: Extensions to the Message-Passing Interface. University of Tennessee, Knoxville (1996)
   →http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html
2. Latham, R., Miller, N., Ross, R., Carns, P.: A Next-Generation Parallel File System for Linux Clusters. LinuxWorld (2004)
3. Schwan, P.: Building a cluster file system for 1,000 node clusters. In: Proceedings of the 5th Linux Symposition. (2003)
4. Isaila, F., Tichy, W.F. In: Clusterfile: A flexible Physical layout Parallel File System. Volume 15. John Wiley & Sons Ltd (2003) p.653–679
5. Barrios, M., et al.: GPFS: A Parallel File System. IBM Red Book (1998)
6. Niederberger, R., Alessandrini, V.: The DEISA Project: Motivations, Strategies, Technologies. In: International Supercomputer Conference 2004, Heidelberg, Germany. (2004)
   →http://www.deisa.org/
7. Wyrzykowski, R.: CLUSTERIX - National CLUSTER of LInuX Systems. In: Grid Workshop, Cracow (2004)
8. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl J. Supercomputer Applications **11(2)** (1997) p.115–128
9. Streit, A., Erwin, D., Lippert, T., Mallmann, D., Menday, R., Rambadt, M., Riedel, M., Romberg, M., Schuller, B., Wieder, P.: UNICORE – from Project Results to Production Grids (2005)
   →http://unicore.sourceforge.net/docs/paper_streit_et_al.pdf
10. DFN: Viola – Vertically Integrated Optical Testbed for Large Scale Applications (2005)
    →http://www.viola-testbed.de/
11. Pöppe, M., Schuch, S., Bemmerl, T. In: A Message Passing Interface Library for Inhomogeneous Coupled Clusters, Nice, France (2003)
12. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Technical report, MCS Division - Argonne National Laboratory (1996)
    →http://www-unix.mcs.anl.gov/mpi/mpich/
13. Thakur, R., Ross, R., Lusk, E., Gropp, W.: Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Technical Report ANL/MCS-TM-234, MCS Division, Argonne National Laboratory (2004)
    →http://www-unix.mcs.anl.gov/romio/
14. Thakur, R., Gropp, W., Lusk, E.: An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In: Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation. (1996) p. 180–187
15. Lee, J., Ma, X., Ross, R., Thakur, R., Winslett, M.: RFS: Efficient and Flexible Remote File Access for MPI-IO. In: Proceedings of the International Conference on Cluster computing. (2004)
    →http://dais.cs.uiuc.edu/~{}jlee17/papers/cluster.pdf

Remote Task Submission and Publishing in BeesyCluster*: Security and Efficiency of Web Service Interface**

Paweł Czarnul, Michał Bajor, Marcin Frączak, Anna Banaszczyk, Marcin Fiszer, and Katarzyna Ramczykowska

Faculty of Electronics, Telecommunications and Informatics,
Gdansk University of Technology, Poland
pczarnul@eti.pg.gda.pl
http://fox.eti.pg.gda.pl/∼pczarnul

Abstract. We present a new system BeesyCluster which can be seen as an easy-to-use access portal to an expandable network of services deployed and published on clusters or PCs with virtual payments for the use of services. Administrators/users can attach their clusters/PCs available via SSH with a click of the button without any need for further configuration on the provider's machine. Further, users can publish console, queued applications or files from their accounts. Services run on the provider's account but access to the services is granted through BeesyCluster either via WWW or Web Services with proper authorization. Providers earn points for their services invoked by users which allows them to use services offered by others. We compare the set of features to other systems, especially grid systems, pointing out the proposed security concept, interfaces and API. We also benchmark the Web Service interface in BeesyCluster by measurement of latency and remote task submission times on large 32-bit 128-processor and 64-bit 256-processor clusters available in the ACC network, Gdansk, Poland. We compare the results to the performance of standard Web Services with HTTP Basic Authentication and HTTPS deployed on Tomcat/AXIS.

1 Introduction

Publishing services for remote clients and remote task invocation are well known concepts in the literature. CORBA, RMI allow for the client-server interaction via remote calls ([1]) as do multi-tier applications with thin/thick clients and servers like J2EE or Tomcat. For Web Services ([2]) input/output arguments are wrapped in SOAP and carried over protocols like HTTP(S) or SMTP. While conceptually equivalent to CORBA/RMI (remote procedure call), SOAP/HTTP enables interaction between parties via usually unblocked HTTP ports making Web Services more versatile and accessible than the other technologies.

* Available on three servers at http://beesycluster[1,2].eti.pg.gda.pl

** Calculations were carried out at the Academic Computer Center, Gdansk, Poland. Work partially sponsored by the Polish National Grant KBN No. 4 T11C 005 25.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 220–227, 2006.

2 Related Work vs. Our Contribution

Controlled resource sharing between, usually large, universities and institutions, has become possible thanks to grid systems ([3]) examples of which are CrossGrid ([4]), CLUSTERIX ([5]) and EuroGrid ([6]). It must be assured that the client can neither overuse the remote resources allocated to them nor succeeds in an attempt to gain unauthorized access to other resources. Furthermore, secure data transmission, accounting and resource discovery must also be supported. Globus Toolkit ([7]) provides grid system developers with many of these functions allowing to focus on the higher level, the actual grid implementation. Legion allows users to spawn tasks remotely on a unified virtual metacomputer ([8]). H2O ([9]) is a component-level distributed system in which components can be deployed in containers, also by authorized external entities and made available to clients. While large grid systems allow the user to run tasks remotely, even via complex and easy-to-use interfaces like Migrating Desktop ([10]), still the configuration of remote sites, often difficult and time-consuming ([11] includes configuration of Worker Nodes for CrossGrid, [8] for Legion), is required.

We see the above as an obstacle in building large, open systems of services with easily configurable, detachable remote sites which could very well be single PCs or clusters offering unique applications or services.

We addressed these goals in a new system BeesyCluster, the continuation of our PVMWebCluster initiative ([12], [13]), which offers the following features:

1. ease of addition of a remote site (cluster/PC) to the system (requires only an account with SSH access with no additional configuration on the access node),
2. one-click service publishing from the provider's account on a cluster/PC (support for queuing systems like PBS, LSF),
3. one-click service (application, files etc.) rental capability from the provider's account(s) by any user of BeesyCluster,
4. accounting for the use of resources – the provider earns points for offering services which can be spent on services offered by others,
5. access through both WWW and Web Services and discovery through BeesyCluster's own UDDI server (based on soapuddi).

3 New Concept – BeesyCluster

3.1 Access Portal to HPC Facilities

On the one hand, BeesyCluster can be thought of as an easy-to-use access portal to HPC facilities allowing the user to run/publish their own applications and edit files on the registered clusters/PCs via WWW (Figures 1 and 2) and Web Services with single sign-on for all clusters/PCs. It is the high-level middleware that executes commands on clusters/PCs on behalf of authorized users. BeesyCluster aims at making powerful ACC[1] clusters available via WWW and Web Services and allow inexperienced users to invoke published MPI/PVM applications with a mere click of the button (Figure 1).

[1] `http://www.task.gda.pl/english/kdm.html`

Fig. 1. BeesyCluster's File Manager

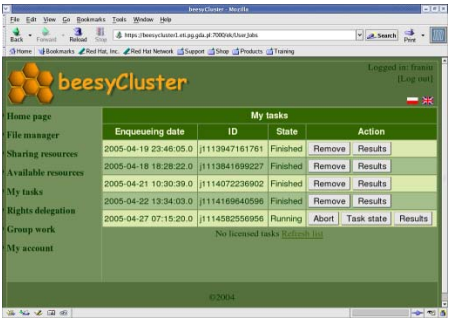


Fig. 2. Tasks (Queued) by the User and Tasks Rented from His/Her Account

3.2 Open, Distributed Network of Services

On the other hand, BeesyCluster offers a truly open network of services offered by providers to clients (Figure 3) with the following features:

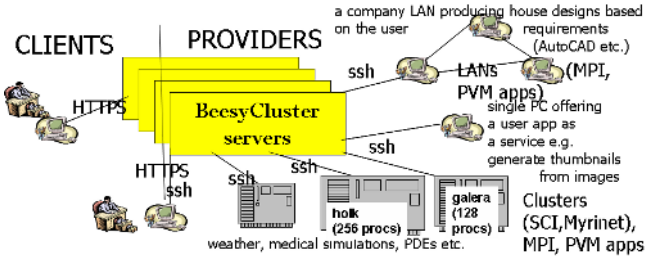


Fig. 3. Client-Server Interaction in BeesyCluster

Computing Model: Many providers, many clients, the client may also offer services as a provider; users can access and manage their accounts/files, run tasks through WWW or Web Services.

Setting Up an Account: A new user-provider sets up a BeesyCluster account by submitting logins/passwords of accounts available through SSH on the selected cluster/PC and is able to log in via BeesyCluster just after the administrator has granted access. It is only required that the account(s) are accessible via SSH. No other configuration is necessary which is possible thanks to the BeesyCluster security model explained in Paragraph 3.3. This gives BeesyCluster an advantage over the often complex process of deployment of new sites in grid systems ([11], [8]). The user can set up a BeesyCluster account without any cluster/PC accounts for use of free services published by others.

Publishing Services: The user can publish any of the applications (both run from the shell and to be submitted to the queue like LSF/PBS) or files from any of the registered accounts of any of the PCs/clusters registered to other users/groups (e.g. student groups) in BeesyCluster. The user-provider who grants access to a resource

can allow right delegation to other users. Publishing a service does not give shell access and thus does not sacrifice security (Paragraph 3.3).

Running Services:

1. The user has full access to any of the files/applications on their registered accounts on the registered PCs/clusters, can: access/edit files, run commands via an easy-to-use WWW interface (Figure 1) or Web Services (Figure 6).
2. For every published service, users see a link in the WWW interface. By clicking the user-client invokes the service which runs on the provider's account as may require a proper environment (libraries etc.). The Web Service interface is meant for more advanced users for integration of BeesyCluster services into their programs, the WWW interface for novice users. In the latter case, a sample scenario may involve clicking on a link corresponding to a climate modelling application, uploading files with input data (to a special directory for the client user on the provider account) and either running the task or specification of the queue, email notification, exact number of processors etc. for LSF or PBS. Afterwards, the user can check the status of the task and upon completion view results in File Manager and download or copy them to their own accounts.

BeesyCluster was designed as a J2EE application with the following components (Figure 4): KC (Cluster Commander) – WWW front-end, TS (Team Support) module, AS (Authorization), RA (Run Anywhere for SSH communication with PCs/ clusters), PS (Payment Service for accounting of rented services – each user has a virtual wallet to which points are added after other users executed paid services or subtracted after the user has used others' services), Web Service front-end which are proper EJBs deployed as Web Services in JBoss/AXIS.

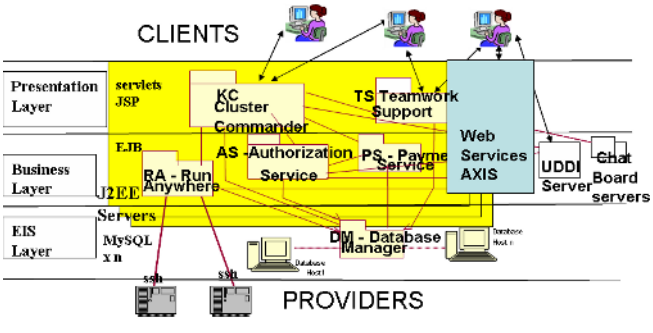


Fig. 4. BeesyCluster's Architecture

3.3 Security Concept

In Globus ([7]) and Legion ([8]) the authorization of the user is actually done by the resource provider/resources. In H2O proxy-objects authorize the user based on the policies given by the kernel owner or the user who loaded the pluglet ([9]). The BeesyCluster server itself acts as a centralized proxy which both authenticates and authorizes

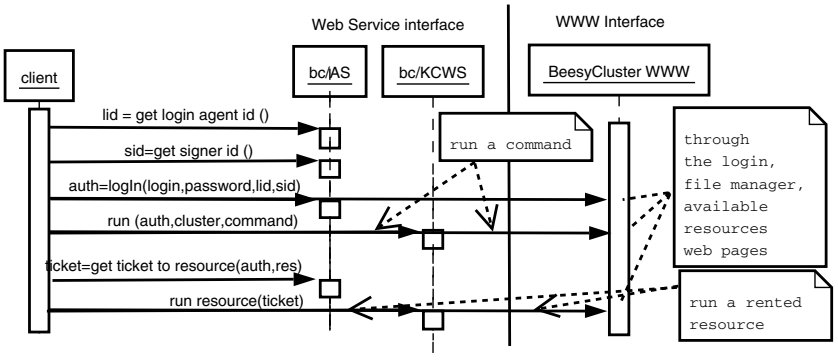


Fig. 5. Sequence to Invoke an Application in BeesyCluster via Web Services and WWW

the user to invoke a given service. It is assumed the provider trusts BeesyCluster invokes services on their account only for the users they granted access to or for the account owner. Thus the attachment of a new cluster/PC and a user-provider account requires just database entries with the IP of the cluster and the account login/password. The commands are issued on the clusters via SSH by module RA (Figure 4).

Transmission from the client to BeesyCluster uses HTTP or HTTPS which requires proper configuration ([14]) and the keystore file for the Web Service client.

The authentication and authorization are implemented within BeesyCluster and require actions presented in Figure 5 for both the Web Service and WWW front-ends. For the Web Service access, after having obtained ids of login (handle logins) and signer modules (handle various encryption algorithms), the client must login with a password and obtain an authenticator which contains the id of the user, expiration date and is digitally signed by the system. The authenticator can be used to launch a command on a cluster the user has access to. Based on the authenticator, the user can obtain a ticket valid to use the given resource. Both the authenticator and the ticket are valid for a definite period of time thus it is safer to use them rather than login/password pairs.

3.4 Interfaces and API

BeesyCluster offers an easy-to-use WWW interface which allows the user to: manage files/directories through an interface similar to Midnight Commander (Figure 1) including (de)compression, editing files, one-click task launch (interactively or queued on a cluster), browsing available resources, publishing own resources with a single mouse click, viewing task results from queued tasks (own and rented tasks) – Figure 2, delegating rights to other users and managing personal data. BeesyCluster allows FTP from the BeesyCluster server to any location where FTP is available. All file operations on the accounts registered in BeesyCluster are executed through a Java SSH library.

For a rented task, a unique directory is created by BeesyCluster before running to which the task can print output (this can be assured by the programmer-provider). The task can read the directory name from an environment variable set by BeesyCluster. In fact, application could also be copied to this directory before execution in effect releasing the programmer from using the variable, just writing output to the directory

```
1 import pl.gda.pg.eti.beesycluster2.*;
  import as.interfaces.auth.*; import as.interfaces.sign.*;
3 ...
  ASService service = new ASServiceLocator(); AS port = service.getAS();
5 try { // first get login agent and signer module ids
      LoginAgentDescription[] lad=port.listLoginAgents();
7     SignerDescription[] sd=port.listSigners();
      String[] auth = port.logIn(new String[] {"<login>","<password>"},
9     lad[0].getID(),sd[0].getID());      // try to login now
      // now try to call a method from the KCWS interface
11    KCWSService service1=new KCWSServiceLocator();KCWS port1=service1.getKCWS();
      System.out.println(port1.runCommand(auth,2,args[0]));//run the given command
13 } catch (IOException e) { ...
```

Fig. 6. Client Code to Invoke a Command on a Cluster via BeesyCluster/Web Services

where the application was spawned. The user-client can view the task results, copy them but does not see other directories nor has access to the shell on the provider's account.

The code for running a given command on the cluster via Web Services is shown in Figure 6. Apart from the account owner, the client user would only run one of resources previously made available to them (include predefined paths to the execution file, maximum number of processors on which the user can run the task etc.).

4 Performance Tests

For benchmarking Web Services, we used the following configurations:

1. **Standard Web Services, No Security.** The client calls a service deployed as a JWS (Java Web Service) file on the Tomcat 4.1.18/AXIS 1.1 server.
2. **Standard Web Services, Basic Authentication, HTTPS.** Configuration as above.
3. **Web Services in BeesyCluster.** The user logs in and runs the command (Figures 5 and 6). We measure times for the latter call which validates the authenticator and invokes the command on the cluster via SSH. Case 2 indicates the HTTPS overhead when added to this scenario. BeesyCluster uses JBoss 3.2.3/AXIS 1.1.

For the server, we used a Pentium4, 2GHz, 1GB RAM machine, for the client a Pentium4 2.8GHz, 1GB RAM laptop, both running Fedora 1. We have measured:

Web Service Latency: Measured by running the `String getString(String)` method with a 3-char String given as input and returned as output. Figure 7 shows the latencies measured on the client and server on a single node, through LAN and WAN (Internet via ADSL) for all three configurations above. For BeesyCluster, we measured just the `getString` method of the KCWS (Figure 6) interface.

Time Required to Run a Task on the Server: For configurations 1 and 2 the command is run on the Tomcat/AXIS server. For BeesyCluster it is run on a cluster account registered for the given user (actually on an access node for 128-processor `galera` and 256-processor `holk` at ACC, Gdansk, Poland). We run command `cat <filename>` which prints the contents of files of varying sizes to the standard output. Figure 8 presents the task run times (until calls return on the client side) on BeesyCluster and Tomcat/AXIS through LAN and WAN for both servers.

For BeesyCluster, the login phase costs as much as the times of returning the string of the authenticator size (around 512 bytes).

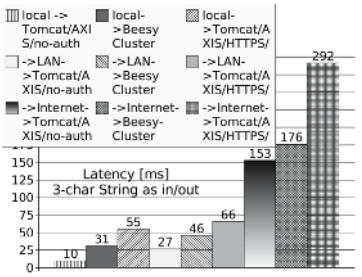


Fig. 7. Latency for 3-char String Used as Input/Output Web Service Parameter

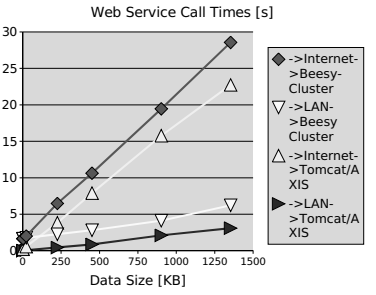


Fig. 8. Task Submission/Run Times via Web Service on Local Node, LAN, WAN (Internet)

It is important to note that BeesyCluster can serve client requests from many servers, in fact implementing clustering thanks to a distributed database with replication, used by all BeesyCluster servers. The client authenticator or ticket are passed in the request for both WWW and Web Service interfaces. We have three servers running at `beesycluster[1,2].eti.pg.gda.pl`.

Analysis of latency of SOAP implementations, including AXIS, can be found in [15]. Our times for AXIS are slightly shorter (understandable given the faster hardware). We also present the latency for SOAP over HTTPS which is around 5 times larger than for HTTP (for small message sizes). We conclude that for latency, the BeesyCluster Web Services (`KCWS.getString` on JBoss/AXIS) are slightly slower than pure Tomcat/AXIS, both of which are faster than HTTPS on Tomcat/AXIS.

For remote task submission, BeesyCluster's call is slightly slower than pure Tomcat/AXIS but implements security and runs a command and fetches standard output from the remote cluster.

5 Conclusions and Future Work

We have explained the main ideas and the security concept of BeesyCluster which offers an access portal to an expandable network of distributed clusters/PCs with easy resource publishing and interface. We have shown that the performance of the Web Service interface is comparable to that of pure Web Services deployed on Tomcat/AXIS.

We are currently deploying the system for the ACC users as an access point for the HPC resources in Gdansk, Poland. We plan on testing the clustering features of BeesyCluster, especially under the large number of incoming requests.

References

1. Buyya, R., ed.: High Performance Cluster Computing, Programming and Applications. Prentice Hall (1999)
2. Streicher, M.: Creating Web Services with AXIS: Apache's Latest SOAP Implementation Bootstraps Web Services. Linux Magazine (2002) `http://www.linux-mag.com/2002-08/axis_01.html`.

3. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications **15**(3) (2001) 200–222 http://www.globus.org/research/papers/anatomy.pdf.
4. CrossGrid: (http://www.crossgrid.org/)
5. R. Wyrzykowski, coordinator: (Clusterix) http://clusterix.pcz.pl.
6. EuroGrid: (http://www.eurogrid.org)
7. Globus System: (http://www.globus.org)
8. The Legion Group: Legion 1.8. System Administrator Manual. (University of Virginia, VA, USA) http://legion.virginia.edu/documentation/sysadmin_1.8.pdf.
9. Kurzyniec, D., Wrzosek, T., Drzewiecki, D., Sunderam, V.: Towards self-organizing distributed computing frameworks: The h2o approach. Parallel Processing Letters **13**(2) (2003) 273–290
10. CrossGrid: (Migrating Desktop - User Guide, Portals and Roaming Access, WP3) wp3.crossgrid.org/CG3.1-D3.6-v2.1-PSNC-MDUserGuide.doc.
11. Floros, V., Markou, C., Mastroyiannopoulos, N.: LCFGng Cluster Installation Guide Version 2.0 (LCG-2). (2004) http://cgi.di.uoa.gr/∼xgrid/cgfiles/LCFGng_v2.0.pdf.
12. Czarnul, P.: Pvmwebcluster: Integration of pvm clusters using web services and corba. In: Proceedings of the 10th European PVM/MPI Users' Group conference EuroPVM/MPI 2003. Volume LNCS 2840., Venice, Italy (2003) 268–275
13. Czarnul, P.: Architecture and implementation of distributed data storage using web services, corba and pvm. In: Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics. Volume LNCS 3019., Czestochowa, Poland (2003) 360–367
14. Gopalakrishnan, U., Ravi, R.K.: Web services security, Part I (2003) www-106.ibm.com/developerworks/webservices/library/ws-sec1.html.
15. Davis, D., Parashar, M.P.: Latency Performance of SOAP Implementations. In: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), Berlin, Germany (2002) 407–412

Open MPI: A Flexible High Performance MPI

Richard L. Graham[1], Timothy S. Woodall[1], and Jeffrey M. Squyres[2]

[1] Advanced Computing Laboratory, Los Alamos National Lab
{rlgraham, twoodall}@lanl.gov
[2] Open System Laboratory, Indiana University
jsquyres@osl.iu.edu

Abstract. A large number of MPI implementations are currently available, each of which emphasize different aspects of high-performance computing or are intended to solve a specific research problem. The result is a myriad of incompatible MPI implementations, all of which require separate installation, and the combination of which present significant logistical challenges for end users. Building upon prior research, and influenced by experience gained from the code bases of the LAM/MPI, LA-MPI, FT-MPI, and PACX-MPI projects, *Open MPI* is an all-new, production-quality MPI-2 implementation that is fundamentally centered around component concepts. Open MPI provides a unique combination of novel features previously unavailable in an open-source, production-quality implementation of MPI. Its component architecture provides both a stable platform for third-party research as well as enabling the run-time composition of independent software add-ons. This paper presents a high-level overview the goals, design, and implementation of Open MPI, as well as performance results for it's point-to-point implementation.

1 Introduction

The face of high-performance computer systems landscape is changing rapidly, with systems comprised of thousands to hundreds of thousands of processors in use today. These systems vary from tightly integrated high end systems, to clusters of PCs and workstations. Grid and meta computing add twists such as a changing computing environment, computing across authentication domains, and non-uniform computing facilities, such as variations in processor type and bandwidths and latencies between processors.

This wide variety of platforms and environments poses many challenges for a production-grade, high performance, general purpose MPI implementation, requires it to provide a high degree of flexibility in many problem axes. One needs to provide tunable support for the traditional high performance, scalable communications algorithms, as well as address a variety of failure scenarios. In addition items such as process control, resource exhaustion, latency awareness and management, fault tolerance, and optimized collective operations for common communication patterns, need to be dealt with.

These types of issues have addressed in one way of another by different projects, but little attention has been given to dealing with various fault scenarios. In particular, network layer transmission errors—which have been considered

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 228–239, 2006.

highly improbable for moderate-sized clusters—cannot be ignored when dealing with large-scale computations [4]. This is particularly true when O/S bypass protocols are used for high performance messaging on systems that do not have end-to-end hardware data integrity. In addition, the probability that a parallel application will encounter a process failure during its run increases with the size of they system used. For an application to survive process failure it either must regularly write checkpoint files (and restart the application from the last consistent checkpoint [1, 10]) or the application itself must be able to adaptively handle process failures during runtime [3] and use an MPI implementation that deals with process failure. These issues are current, relevant research topics. While some have been addressed at various levels by different research efforts, no single MPI implementation is currently capable of addressing all these in a comprehensive manner.

Therefore, a new MPI implementation is required: one that is capable of providing a framework to address important issues in emerging networks and architectures. The Open MPI project was initiated with the express intent of addressing these, and other issues. Building upon prior research, and influenced by experience gained from the code bases of the LAM/MPI [13], LA-MPI [4], FT-MPI [3], and the PACX-MPI [5] project, *Open MPI* is an all-new, production-quality MPI-2 implementation. Open MPI provides a unique combination of novel features previously unavailable in an open source implementation of MPI. Its component architecture provides both a stable platform for cutting-edge third-party research as well as enabling the run-time composition of independent software add-ons.

1.1 Goals of the Open MPI Project

While all participating organizations have significant experience in implementing MPI, Open MPI represents more than a simple merger of the LAM/MPI, LA-MPI, FT-MPI, and PACX-MPI code bases. While influenced by previous implementation experiences, Open MPI uses a new software design to implement of the Message Passing Interface. Focusing on production-quality performance, the software implements the MPI-1.2 [7] and MPI-2 [8] specifications and supports concurrent, multi-threaded applications (i.e., MPI_THREAD_MULTIPLE).

To efficiently support a wide range of parallel machines, high performance "drivers" for established communications protocols have been developed. These include TCP/IP, shared memory, Myrinet (GM and MX), and Infiniband (MVAPI and OpenIB). Support for more devices will likely be added based on user, market, and research requirements. For network transmission errors, ideas first explored in LA-MPI are being extended with optional support is being developed for checking data integrity. In addition, by utilizing message fragmentation and striping over multiple (potentially heterogeneous) network devices, Open MPI is capable of both maximizing the achievable bandwidth to applications and is developing the ability to dynamically handle the loss of network devices when nodes are equipped with multiple network interfaces. Handling of these network failovers is completely transparent to MPI applications.

The Open MPI run-time layer provides basic services to start and manage parallel applications in interactive and non-interactive environments. Where possible, existing run-time environments is leveraged to provide the necessary services; a portable run-time environment based on user-level daemons is used where such services are not already available.

2 The Architecture of Open MPI

Open MPI's primary software design motif is a component architecture called the Modular Component Architecture (MCA). The use of components forces the design of well contained library routines and makes extending the implementation convenient. While component programming is widely used, it is only recently gaining acceptance in the high performance computing community [2, 13]. As shown in Fig. 1, Open MPI is comprised of three main functional areas:

- MCA: The backbone component architecture that provides management services for all other layers;
- Component frameworks: Each major functional area in Open MPI has a corresponding back-end component framework, which manages modules;
- Components: Self-contained software units that export well-defined interfaces and can be deployed and composed with other components.

The MCA manages the component frameworks and provides services to them, such as the ability to accept run-time parameters from higher-level abstractions (e.g., `mpirun`) and pass them down through the component framework to individual components. The MCA also finds components at build-time and invokes their corresponding hooks for configuration, building, and installation.

Each component framework is dedicated to a single task, such as providing parallel job control or performing MPI collective operations. Upon demand, a framework will discover, load, use, and unload components. Each framework has different policies and usage scenarios; some will only use one component at a time while others will use all available components simultaneously.

Components are self-contained software units that can configure, build, and install themselves. Components adhere to the interface prescribed by the framework that they belong to, and provide requested services to higher-level tiers.

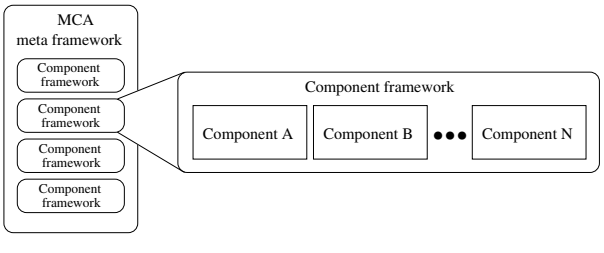


Fig. 1. Three main functional areas of Open MPI: The MCA, its component frameworks, and the components in each framework

The Open MPI software has three classes of components: Open MPI components, Open Run Time Environment (ORTE) components, and Open Portable Access Layer (OPAL) components.

The following is a partial list of MPI component frameworks in Open MPI (only MPI functionality is described; ORTE and OPAL frameworks and components are not covered in this paper):

- Point-to-point Management Layer (PML): This component manages all full message delivery. It implements the the semantics of a given point-to-point communications protocol, such as MPI.
- Byte-Transfer-Layer Layer (BTL): This component is handles point-to-point data delivery over the network, and is unaware of upper-level point-to-point communications protocols, such as MPI.
- BTL Management Layer (BML): This component provides services during job startup and dynamic process creation to discover and maintain the set of BTLs that may be used for point-to-point communications between a given pair of end-points.
- Collective Communication (COLL): The back-end of MPI collective operations, supporting both intra- and intercommunicator functionality.
- Process Topology (TOPO): Cartesian and graph mapping functionality for intracommunicators. Cluster-based and Grid-based computing may benefit from topology-aware communicators, allowing the MPI to optimize communications based on locality.
- Parallel I/O: I/O modules implement parallel file and device access. Many MPI implementations use ROMIO [14], but other packages may be adapted for native use (e.g., cluster- and parallel-based filesystems).

The wide variety of framework types allows third party developers to use Open MPI as a research platform, a deployment vehicle for commercial products, or even a comparison mechanism for different algorithms and techniques.

The component architecture in Open MPI offers several advantages for end-users and library developers. First, it enables the usage of multiple components within a single MPI process. For example, a process can use several networks simultaneously. Second, it provides a convenient possibility to use third party software, supporting both source code and binary distributions of components. Third, it provides a fine-grained, run-time, user-controlled component selection mechanism.

2.1 Module Lifecycle

The Byte Transfer Layer (BTL) framework provides a good illustrative example of the complete usage and lifecycle of a module in an MPI process:

1. During MPI_INIT, the BTL Management Layer (BML) framework (described below) discovers all available BTL components. Components may have been statically linked into the MPI library or can be loaded from shared libraries located in well-known locations.

2. Each BTL component is queried to see if its want to run in the process. Components may choose not to run; for example, an Infiniband-based component may choose not to run if there are no Infiniband NICs available.
3. Components that are selected and successfully initialized will return a set of BTL modules to the BML, each representing distinct network interfaces or ports. Each module may cache resources and addressing information required for communication on the underlying transport.
4. The BML queries each of the BTL modules to determine the set of processes with which they are able to communicate. For each peer, the BML maintains a list of BTLs through which that peer is reachable. These tables are exposed to the upper layers for efficient message delivery and striping.
5. BTL modules exist for the duration of the process. During MPI_FINALIZE, each BTL module is given the opportunity to cleanup any allocated resources prior to closing its corresponding component.

3 Implementation Details

Several aspects of Open MPI's design are discussed in this section.

3.1 Object Oriented Approach

Open MPI is implemented using a simple C-language object-oriented system with single inheritance and reference counting-based memory management using a retain/release model. An "object" consists of a structure and a singly-instantiated "class" descriptor. The first element of the structure must be a pointer to the parent class' structure.

Macros are used to effect C++-like semantics (e.g., new, construct, destruct, delete). Upon construction, an object's reference count is set to one. When the object is retained, its reference count is incremented; when it is released, its reference count is decreased. When the reference count reaches zero, the class; destructor (and its parents' destructor) is run and the memory is freed.

The experience with prior software projects based on C++ and the according compatibility and compilation problems on some platforms has encouraged us to take this approach instead of using C++ directly. For example, C++ compilers may layout the same class or structure differently in memory. This can lead to problems when serializing data and sending it across a network if the sender was compiled with a different compiler than the receiver.

3.2 Component Discovery and Management

Open MPI offers three different mechanisms for adding a component to the MPI library (and therefore to user applications):

– During the configuration of Open MPI, a script traverses the build tree and generates a list of components found. These components will be configured, compiled, and linked statically into the MPI library.

– Similarly, components discovered during configuration can also be compiled as shared libraries that are installed and then re-discovered at run-time.
– Third party library developers who do not want to provide the source code of their components can configure and compile their components independently of Open MPI and distribute the resulting shared library in binary form. Users can install this component into the appropriate directory where Open MPI can discover it at run-time.

At run-time, Open MPI first "discovers" all components that were statically linked into the MPI library. It then searches several directories to find available components and sorts them by framework type.

Components are identified by their name and version number. This enables the MCA to manage different versions of the same component, ensuring that the components used in one MPI process are the same—both in name and version number–as the components used in a peer MPI process. Given this flexibility, Open MPI provides multiple mechanisms both to choose a given component and to pass run-time parameters to components: command line arguments to `mpirun`, environment variables, text files, and MPI attributes (e.g., on communicators).

3.3 Point-to-Point Components

The Open MPI point-to-point (p2p) design and implementation is based on multiple MCA frameworks. These frameworks provide functional isolation with clearly defined interfaces. Fig. 2 illustrates the p2p framework architecture.

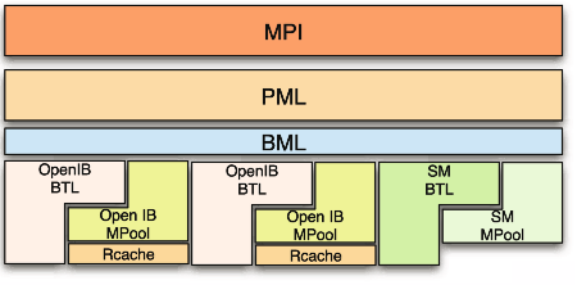


Fig. 2. Open MPI p2p framework

As shown in Fig. 2 the architecture consists of four layers. Working from the bottom up these layers are the Byte Transfer Layer (BTL), BTL Management Layer (BML), Point-to-Point Messaging Layer (PML) and the MPI layer. Each of these layers is implemented as an MCA framework. Other MCA frameworks shown are the Memory Pool (MPool) and the Registration Cache (RCache). While these frameworks are illustrated and defined as layers, performance-critical send/receive paths bypass the BML, as it is used primarily during initialization and BTL component selection.

MPool. The memory pool provides memory allocation/deallocation and registration/deregistration services. OS-bypass networks such as Infiniband and Myrinet require memory to be registered (physical pages present and pinned) before send/receive or RDMA operations can use the memory as a source or target. Separating this functionality from other components allows the MPool to be shared among various layers. For example, MPI_ALLOC_MEM uses these MPools to register memory with available interconnects.

RCache. The registration cache allows memory pools to cache registered memory for later operations. When initialized, MPI message buffers are registered with the MPool and cached via the RCache. For example, during an MPI_SEND the source buffer is registered with the memory pool and this registration may be then be cached, depending on the protocol in use. During subsequent MPI_SEND operations the source buffer is checked against the RCache, and if the registration exists the PML may RDMA the entire buffer in a single operation without incurring the high cost of registration.

BTL. The BTLs expose a set of communication primitives appropriate for both send/receive and RDMA interfaces. The BTL is not aware of any MPI semantics; it simply moves a sequence of bytes (potentially non-contiguous) across the underlying transport. This simplicity will enable early adoption of novel network devices and encourages vendor support.

BML. The BML acts as a thin multiplexing layer allowing the BTLs to be shared among multiple upper layers. Discovery of peer resources is coordinated by the BML and cached for multiple consumers of the BTLs. After resource discovery, the BML layer is bypassed by upper layers for performance.

PML. The PML implements all logic for p2p MPI semantics including standard, buffered, ready, and synchronous communication modes. MPI message transfers are scheduled by the PML based on a specific policy. This policy incorporates BTL specific attributes to schedule MPI messages. Short and long message protocols are implemented within the PML. All control messages (ACK/NACK/MATCH) are also managed at the PML. The benefit of this structure is a separation of transport protocol from the underlying interconnects. This significantly reduces both code complexity and code redundancy enhancing maintainability.

Althought there are currently three PML components available in the Open MPI, this paper only discusses the `OB1` PML component. `OB1` is Open MPI's latest generation PML, reflecting the most recent communications research. There is currently only one BML component – "`R2`." Finally, there are several BTL modules available, providing support for the following networks: TCP, shared memory, Portals, Myrinet/MX, Myrinet/GM, Mellanox VAPI, and OpenIB VAPI.

These components are used as follows: during startup, a PML component is selected and initialized. The PML component selected defaults to `OB1` but may be overridden by a run-time parameter. Next the BML component `R2` is selected (since there is only one available). `R2` then opens and initializes all available BTL modules. During BTL module initialization, `R2` directs peer resource discovery on a per-BTL component basis. This allows the peers to negotiate which set of interfaces they will

use to communicate with each other for MPI communications. This infrastructure allows for heterogeneous networking interconnects within a cluster.

4 Performance Results

The data obtained in this section was using Open MPI version 1.0.1rc6 using the `OB1 PML` with the MVAPI, GM, Shared Memory, and TCP BTL components. These runs are compared with data from three other MPI implementations on the same hardware. GM data was obtained with MPICH-GM version 1.2.6; the shared memory and TCP data with MPICH2 version 1.0.3 [9]; the Infiniband data with MVAPICH version 0.9.5 [6]. Latency is measured as half the round trip time of zero byte MPI messages, using a ping-pong benchmark code, and the bandwidth data is measured using NetPIPE version 3.6.2 [12]. The systems used to make these measurements are described in the Table 1.

Table 1. Hardware and software setup used to generate results data

Interconnect	CPU	RAM	Bus	Kernel	Stack
Infiniband	(2) Intel Xeon 3.6 GHZ	6GB	PCIe	2.6.12	IB Gold 1.0
Myrinet	(2) Intel Xeon 2.8 GHz	2GB	PCI-X	2.6.11	GM 2.0.22
TCP/SM	(2) AMD Opteron 2 GHz	8GB	PCI-X	2.6.9	

4.1 Point-to-Point Latency

Ping-pong latencies are listed in Table 2. As this table shows, Open MPI has extremely competitive latencies. The latency of 6.86 μsec using GM is slightly better than that obtained by MPICH-gm. The shared memory latency of 1.23 μsec is about 1 microsecond better than MPICH2, but the TCP latency of 32.0 μsec, is 3 microseconds higher than MPICH2 (tuning is ongoing). The latency of 5.64 μsec using the MVAPI verbs is about 1.5 higher μsec than MVAPICH's remote queue management scheme. However, because Open MPI uses the Shared Receive Queue support provided by the MVAPI verbs, it scales much better than

Table 2. Latency of zero byte ping-pong messages

Implementation	Latency
Open MPI GM	6.86μs
MPICH-GM	7.10μs
Open MPI Shared Memory	1.23μs
MPICH2 Shared Memory	2.21μs
Open MPI TCP	32.0μs
MPICH2 TCP	29.0μs
Open MPI OpenIB	5.13μs
Open MPI MVAPI	5.64μs
MVAPICH MVAPI (RDMA)	4.19μs
MVAPICH MVAPI (Send/Recv)	6.51μs

MVAPICH by using far less memory. MVAPICH's latency increases linearly with the number of processes in MPI_COMM_WORLD; Open MPI's latency remains essentially constant (and is lower than MVAPICH's) [11]. Open MPI's latency is about one microsecond lower than MVAPICH's send/receive semantics.

4.2 Point-to-Point Bandwidth

Fig. 3 shows Open MPI obtaining slightly higher bandwidths than MPICH-GM, peaking out around 235 MB/sec – very close to the raw GM bandwidth.

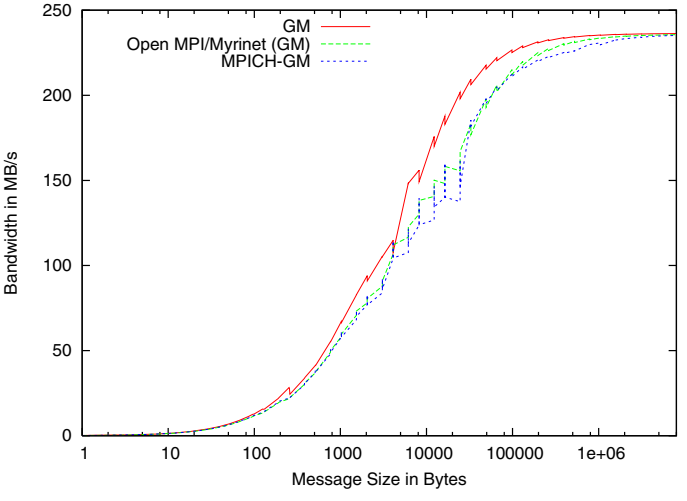


Fig. 3. Myrinet / GM bandwidth of Open MPI vs. MPICH-gm

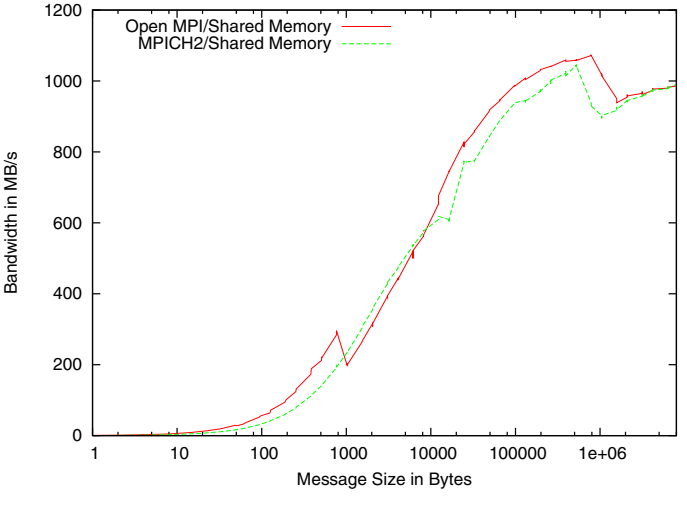


Fig. 4. Shared memory bandwidth of Open MPI vs. MPICH2

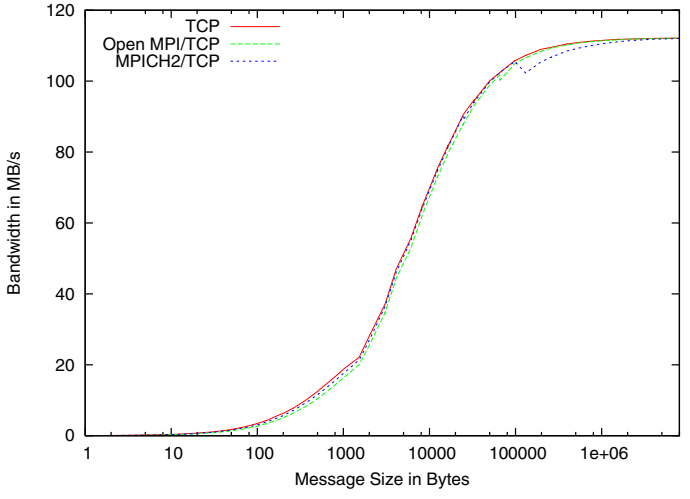


Fig. 5. TCP (Gigabit Ethernet) bandwidth of Open MPI vs. MPICH2

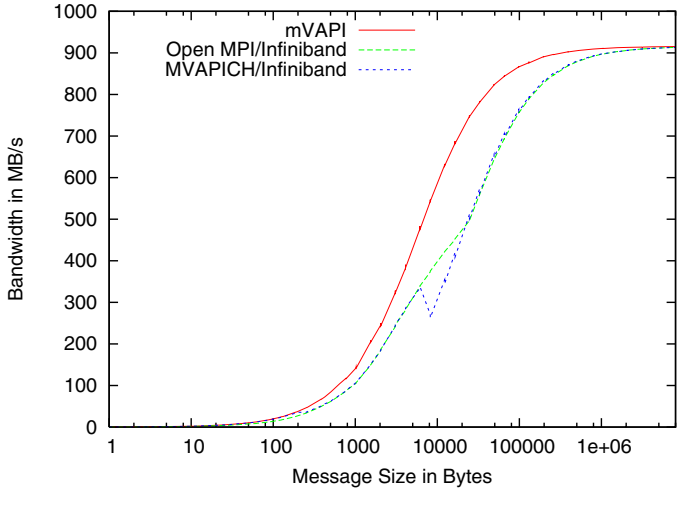


Fig. 6. Mellanox VAPI bandwidth of Open MPI vs. MVAPICH

Similarly, Fig. 4 shows that Open MPI shows better shared memory bandwidths than MPICH2, peaking out at 1020 MB/sec, with a message size of about 1 MB. MPICH2 and Open MPI over TCP (Gigabit Ethernet) exhibit similar bandwidths, peaking out at 112 MB/sec, shown in Fig. 5. Open MPI's MVAPI bandwidth is shown in Fig. 6; it is quite similar to those obtained by MVAPI, peaking out at 915 MB/sec. Finally, Open MPI's Open IB bandwidth is shown in Fig. 7; no other Open IB-native MPI is available to compare to.

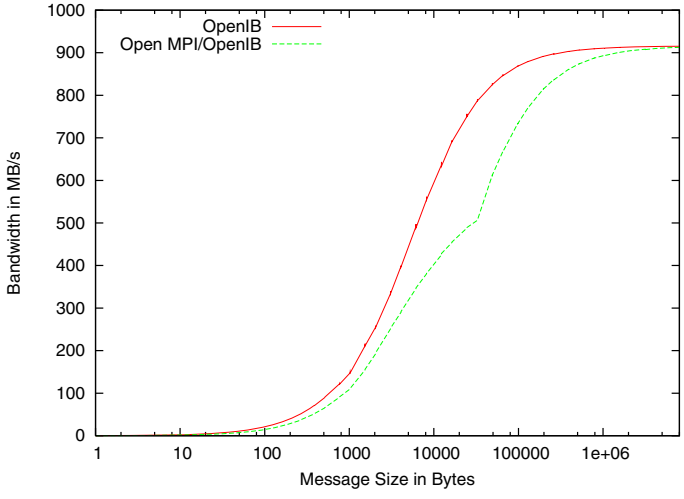


Fig. 7. Open IB bandwidth of Open MPI (no other MPI implementation to compare to)

5 Summary

Open MPI is a new implementation of the MPI 2.0 standard. It provides functionality that has not previously been available in any single, production-quality MPI implementation, including support for all of MPI-2, multiple concurrent user threads, and multiple options for handling process and network failures. Open MPI uses a flexible component architecture, and it's point-to-point design is such that it provides excellent point-to-point communications performance for wide variety of interconnects, all within a single library implementation.

Acknowledgments

This work was supported by a grant from the Lilly Endowment, National Science Foundation grants 0116050, EIA-0202048, ANI-0330620, Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36. Project support was provided through ASC/PSE and ASC/S&CS programs. LA-UR-05-9219.

References

1. G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In *SC'2002 Conference CD*, Baltimore, MD, 2002. IEEE/ACM SIGARCH. pap298,LRI.
2. D. E. Bernholdt et. all. A component architecture for high-performance scientific computing. *Intl. J. High-Performance Computing Applications*, 2004.

3. G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovski, and J. J. Dongarra. Fault tolerant communication library and applications for high perofrmance. In *Los Alamos Computer Science Institute Symposium*, Santa Fee, NM, October 27-29 2003.
4. R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4):285–303, August 2003.
5. Rainer Keller, Edgar Gabriel, Bettina Krammer, Matthias S. Mueller, and Michael M. Resch. Towards efficient execution of parallel applications on the grid: porting and optimization issues. *International Journal of Grid Computing*, 1(2):133–149, 2003.
6. Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. High performance RDMA-based MPI implementation over infiniband. In *ICS '03: Proceedings of the 17th annual international conference on Supercomputing*, pages 295–304, New York, NY, USA, 2003. ACM Press.
7. Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, June 1995. http://www.mpi-forum.org/.
8. Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, July 1997. http://www.mpi-forum.org/.
9. MPICH2. http://www.mcs.anl.gov/mpi/mpich2/.
10. Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, and Eric Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing. *International Journal of High Performance Computing Applications*, To appear, 2004.
11. Galen M. Shipman. Infiniband scalability in Open MPI. Master's thesis, University of New Mexico, December 2005.
12. Q.O. Snell, A.R. Mikler, and J.L. Gustafson. NetPIPE: A Network Protocol Independent Performace Evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.
13. J.M̃. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, Venice, Italy, Sept. 2003. Springer.
14. Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective I/O in ROMIO. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pages 182–189. IEEE Computer Society Press, Feb 1999.

ClusteriX Data Management System and Its Integration with Applications*

Lukasz Kuczynski, Konrad Karczewski, and Roman Wyrzykowski

Institute of Computational and Information Sciences,
Czestochowa University of Technology
{neron, xeno, roman}@icis.pcz.pl

Abstract. Grid applications deal with large volumes of data. As a consequence, effective data management solutions are vital for grids. The Clusterix Data Management System (CDMS) has been developed after a careful analysis of end-user requirements and existing implementations. A special attention has been paid to making the system user-friendly and efficient, aiming at the creation of a reliable and secure Data Storage System. For example, taking into account network parameters, CDMS tries to optimize data throughput via replication and replica selection techniques. Another key feature considered during the development is fault tolerance. In CDMS the distributed operation model and modular design assure elimination of single point of failure. In particular, multiple instances of the data broker are running concurrently, and their coherence is ascertained by the synchronization subsystem. This paper presents the concept and details of implementation of data management mechanisms adopted in CDMS, as well as its integration with applications.

1 Introduction

Data management issues are amongst the most important in modern grid environment [1, 12]. The applications being run on grids become more real-life oriented, they base on and generate data sets of growing importance and confidentiality.

One of the principal goals of data management systems in grids [1] is to provide transparent and efficient access to globally distributed data. Among the most important issues that need to be solved are: optimization of the data transfers over the WAN, reliability and security of data access [7] and ease of use.

The most frequently encountered approach to solving these problems bases on use of metadata and mechanism of data replication [2, 10]. Metadata are used, e.g., for the translation of a logical filename to its physical location. The replication mechanism should provide optimization of data access and reliability.

* This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098 "ClusteriX - National Cluster of Linux Systems".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 240–248, 2006.

An example of a modern data management system based on the above-mentioned mechanisms is the Reptor system [9], developed as a part of the EU DataGrid Project. Being one of the most advanced grid data management systems, it still does not provide full transparency, and ease of use. Its other shortcomings are: the lack of mechanisms of adaptation to the network infrastructure, and presence of single points of failure, e.g., single Metadata Repository.

ClusteriX (National Cluster of Linux Systems) is a distributed national computing infrastructure with 12 sites (local Linux clusters based on 64-bit Itanium2 processors) located accross Poland [3, 13]. ClusteriX sites are connected by the Polish Optical Network PIONIER providing a dedicated communication infrastructure. This paper presents our experience in building the ClusteriX Data Management System (CDMS) [8] and its integration with applications.

The paper is organized as follows. In Section 2, we introduce the concept and functionality of CDMS and the system architecture minutes are presented in Section 3. Section 4 is devoted to system interface, while Sections 5 and 6 describe respectively the integration of CDMS with end-user applications and the GRMS Resource Broker. The paper finishes with conclusions in Section 7.

2 Concept and Functionality

During the design phase of the CDMS system development, a particular attention was paid to the creation of a fault-tolerant, secure and scalable architecture.

The system was equipped with many mechanisms aimed at providing fast data access, as well as adequate security of the stored and transferred data. A high-security level was achieved via a robust data access control system comprised of system-level attributes and access control lists (ACLs).

The development of an intuitive and effective data access and system administration toolkit was seen as an equally important task. It is particularly necessary when the end-user is not expected to be aware of the low-level mechanisms and in the CDMS an Virtual Filesytem (VFS) abstraction layer was implemented, creating an illusion of working with a local file system.

The CDMS has a modular design which allows for an easy replacement of the standard decision-making modules. The current set of modules is based on simple heuristics, and they can be substituted with more advanced ones. For example, an artificial intelligence based approach (using neural networks, genetic algorithms, etc.) can be used, or a dedicated set aimed at a specific grid environment can be developed. The use of the dynamic loading mechanism available in most of modern operating system allows to change the CDMS functionality at runtime, without shutting down the Data Broker.

In order to increase the fault tolerance, there is a possibility to run multiple Data Broker instances. One of them acts as a master process, while the remaining ones are passive until a failure occurs. The Synchronisation Module is responsible for maintaining data integrity between the instances. Implementation of the heart-beat mechanism makes it is possible to detect a failure and initiate a recovery procedure instantly.

The Transport Subsystem consists of a set of Transport Agents, which allows for implementation of advanced transfer-time mechanisms like data encryption and partitioning. This can be achieved again by reimplementation of stamdard Transport Agents. As a further improvement of efficiency of the Transport Subsystem, the implementation of so-called chain-transfer has been proposed [7], which would improve the bandwidth utilisation and network load-balancing.

The CDMS has been implemented in the C language (Optimisation an Replication Modules excepted) using the gSOAP package[6]. An adequate data transmission security, x509 certificates infrastructure and gsiftp protocol support have been achieved using Globus Toolkit 3.x libraries[4].

In the grid infrastructure based on the Globus Toolkit, users are identified using x509 certificates, which have unique subjects. This fact is the foundation of user namespaces introduced in CDMS, which are named after the subject of an user certificate. This approach eliminates possibility of collisions in file and directory names. Every user in the CDMS system has his own filesystem root (/) located in his namespace. The Universal Resource Locator (URL) for the CDMS system is defined as follows: `cdms://[user-namespace]/url-path`. The user-namespace part can be omitted, in such case the subject of the certificate used to access the data will be used automatically.

3 System Architecture

The architecture of the ClusteriX Data Management System was introduced in [8]. It has a modular design and consists of (Fig. 1):

- Main Management Module (CDMS Core)
- Global Data Catalogue (GDC)
- Local Data Catalogue (LDC)
- Transport Subsystem
- Synchronization Module
- Statistic Module
- Optimization Module
- Replication Module

The main part of the system is the CDMS Core responsible for data collections management, data coherence, running the Optimizer and Replicator processes and data transfer initialization. Using data stored in the Global Data Catalogue, the Main Management Module performs the mappings of logical filenames to the Storage Element holding the data. Proper functioning of the GDC is crucial for reliable operation of the CDMS, which makes replication of this data vital for the entire system.

The responsibility of the Replication Module is to perform data replication on the CDMS Core request. It currently allows for the initial and automatic replication.

The initial replication process consists of three stages: choice of the suitable Storage Elements, replication planning, and the replication itself. Accepting a

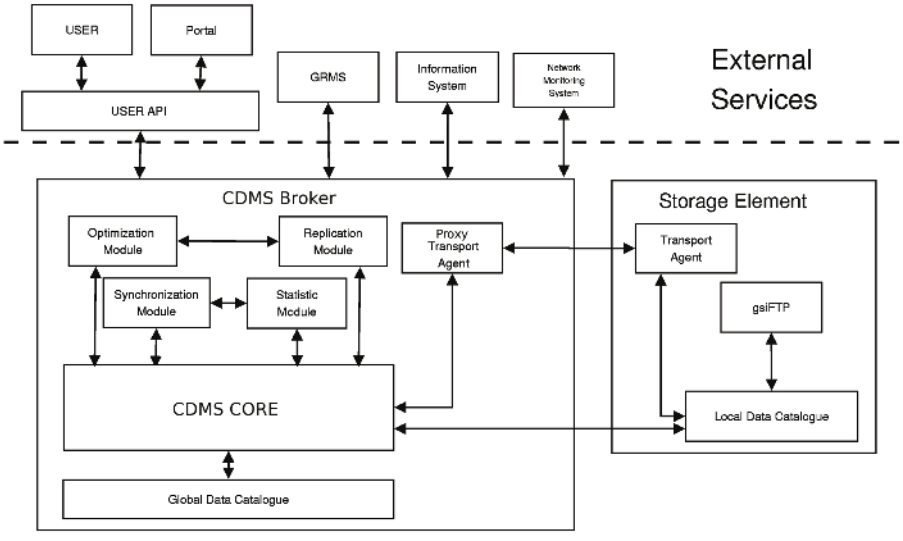


Fig. 1. Architecture of the CDMS

request for an incoming data transfer – from a user into the CDMS – the Main Management Module queries the Optimization Module for possible locations for the incoming data. Next it takes the first two entries from the returned list, and initiates the parallel data transfer.

The automatic replication is carried out by the Replication Module when the system load is low. It decides upon decreasing or increasing the number of replicas using information provided by the Statistic Module. When the demand for a given dataset increases, the number of replicas is increased as well. When the data are no longer needed, the number of replicas is decreased accordingly by removing the least accessed copies [11].

The main task of the Optimization Module is determining the best data location from available replicas. The application of this module decreases delays in data access and balances the load between Storage Elements. The Optimization Module uses such statistical data as network thoroughput and performance of Storage Elements, as well as measured values like current network load, system load and available disk space on Storage Elements.

The Transport Subsystem has been introduced to increase the CDMS performance during data transfers. It consists of the Proxy Transport Agents (PTAs) and Transport Agents (TAs). The PTA is responsible for transferring data between the user and the CDMS. It runs as a standalone process, accepting data transfer requests from the CDMS Core. Such a solution allows the CDMS Core to select the agent located closest (in networking terms) to the served user.

The main task of TA is transferring data between Storage Element and the Proxy Transport Agents. Data sent by the user to PTA are directed to a suitable TA. The ClusteriX Core asks the Optimization Module for the suggested data locations, and then it requests thr proper TA to perform the required operations.

An important feature of the Transport Subsystem is parallel data transfer between the Proxy Agent and Transport Agents. It enables data replication in the very moment they enter the Data Management System without any considerable overhead.

4 System Interface

Access to the system resources is possible via a WebService interface using the SOAP protocol. Such a solution allows to make client applications independent from the operating system and programming language. An example of a CDMS client application is an administrative toolkit implemented in the C language for the Linux/UNIX platform. Another example is the GridSphere[5] portlet, offering a rich user-level access funtionality, implemented in the Java language.

Every interface function returns a message which consists of two parts. The first one contains the error code and the error message. The second part is strictly dependent on the called function, and contains the relevant data, for example, directory listing. The system interface is split into the end-user and administrator parts.

The basic functionality of the CDMS is accessible by a set of functions belonging to the end-user interface. They allow user applications to create and remove directories, copy data between CDMS and local filesystem, list contents of directories, etc. The basic set of user utilities is a part of the CDMS package. They have been deliberately implemented to resemble standard Unix utilities. For example, `clx_ls /` displays contents of a user home directory.

A package of administration utilities has been provided as well. They use the WebService interface to communicate with the CDMS broker. The provided functionality includes: creation of user account and removal, quota manipulation and modification of access control lists.

The CDMS administration will be greatly simplified by a GridSphere portlet which is currently under development. It will allow the system to be administered via a web browser, making this task completely independent from the operating system.

5 Integration of End-User Applications with CDMS

Computational applications can use CDMS directly or indirectly. The most common situation is when an application works with files stored on a local filesystem. In such case, input files of the application can be staged in from the CDMS, and the results stored in the CDMS during the stage-out phase. This is the "indirect" use, and does not require any modification of the application itself.

Another case is when the application is modified to use the CDMS directly. This involves use of the WebService interface via the SOAP protocol. Additionally any interaction with the CDMS must be authenticated and later encrypted via the GSI layer, so support for this feature is another requirement for the

application. After satisfying these requirements, the end-user would be able to request data transfer to and from CDMS during computations.

The general scheme of the application execution is very similar in both cases. First the user places a request to the Resource Broker, e.g. GRMS, specifying resource requirements, input and output data, and providing it with a credential, allowing it to interact on the user behalf with CDMS as well as the local job-managers. The Resource Broker selects an apropriate computational resource, requests input data transfer and commits the application to a local job manager. After the computations are finished, the obtained results are retrieved from the local filesystem and placed in the CDMS. The sole difference is that the CDMS-aware application can fetch additional data during run-time, for example, after assessing results obtained at a specific point of computations. Also, such an application can place partial results in the Data Management System allowing the user to check on the application progress periodically, or use them as the input data for another application.

6 Integration with GRMS

A very important part of the CDMS development was to implement mechanisms of cooperation with the Resource Broker. The final result is almost a complete transparency of this cooperation from the user point of view. The only difference is a modification of the URL. In a basic grid infrastructure data management is based on a ftp server accessed via the gsiftp protocol, and the URLs point to such a server. In a CDMS-enabled infrastructure URLs point to logical file names (Fig.2), which are further resolved by the Data Broker.

```
<grmsjob appid = "demo">
  <simplejob>
     <executable type="single" count="1">
        <file name="exec-file" type="in">
          <url>cdms:///demo.pl</url>
        </file>
     </executable>
  </simplejob>
</grmsjob>
```

Fig. 2. Sample job description including CDMS URL. The `demo.pl` file is located in the root directory of the user running this job

The GRMS analyses the job description, and decides whether an initial data transfer is necessary. When the application specifies a remote source of data in a standard grid infrastructure, GRMS is responsible for copying the data via a third-party transfer to a computational resource. With CDMS such a scenario is not possible because the physical data location is unknown. In this case, the GRMS connects to the CDMS Broker and requests the data to be transfered on the designated node.

In the CDMS WebService interface, multiple functions for copying data to computational nodes are defined. For the integration with Resource Brokers, the following functions are designated:

```
1)  enum CopyStatus { COPYING, FINISHED, FAILED };
2)  CopyStatus copyToCEBlocking( string lfn, string url );
3)  CopyStatus copyFromCEBlocking( string lfn, string url );
4)  string copyToCE( string lfn, string url );
5)  string copyFromCE( string lfn, string url );
6)  CopyStatus getCopyStatus( string sid );
```

The blocking functions (2 and 3) require as the parameters the logical file name (URL in CDMS), and external data locations, e.g., URL pointing to the computational resource storage. They return the status of copy operation (FINISHED or FAILED). The non-blocking functions (4 and 5) accept exactly the same parameters as their blocking versions, but they return an unique data transfer session identifier (SID). It can be used to check the current status of a data transfer via the `getCopyStatus` function. It may return one of the states defined in the `CopyStatus` enum. Such an approach allows for the CDMS integration with any Resource Broker.

6.1 Stage-In Scenario

The sequence diagram (Fig.3) presents the CDMS actions during a data transfer request from the Resource Broker.

In the first step, the GRMS decides upon the resource assignment, and requests a data transfer to be performed. At the moment, GRMS uses blocking functions so the `copyToCEBlocking` function will be called (1).

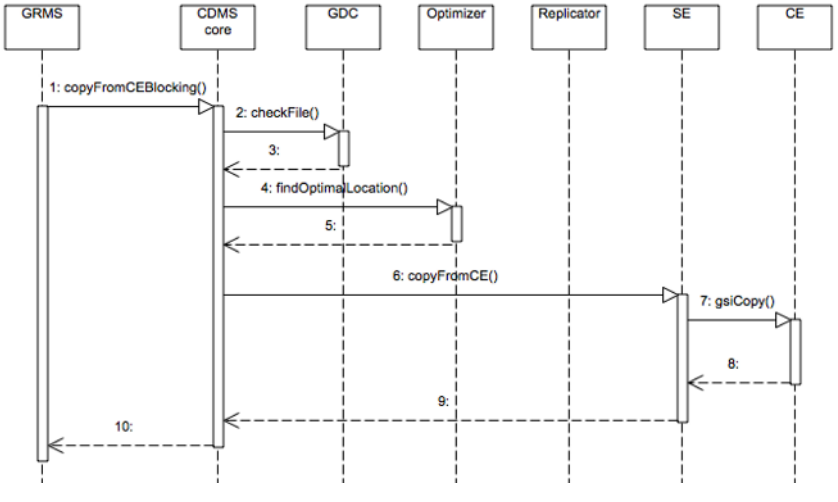


Fig. 3. Stage-in sequence diagram

The Management Module of CDMS verifies if the requested file exists in CDMS (2), and calls the Optimization Module to determine the best physical data location to be used (4). The Optimization Module requires, as its input, the destination of requested data, list of Storage Elements holding replicas of the file, and file size. Using these parameters and querying the Network Resource Manager, the Optimization Module orders the list of Storage Elements by feasibility and returns it to the Management Module. The CDMS Broker delegates user credentials (obtained from the GRMS) to the selected Storage Element, and requests it (6) to perform data transfer to a computational resource (7). When this data transfer is finished (or it has failed), the `copyToCEBlocking` function returns (10) with a proper status code, and the GRMS continues with the job preparation and execution.

6.2 Stage-Out Scenario

After the job is finished, the results have to be retrieved from the computational resource. If the user requested them to be placed in the CDMS, the GRMS again contacts the Data Broker to request data transfer. In Fig.4 the sequence diagram for such a scenario is shown.

First the GRMS calls (1) a blocking function `copyFromCEBlocking` with the proper parameters (logical file name and physical data location). The CDMS Broker checks whether such a file exists (2). If so, all the replicas will be updated to the new version and if it is a new file, CDMS creates its logical instance in the Global Data Catalogue, and starts the optimization process (4). In this case, the Optimization Module requires only the file size and physical data location as

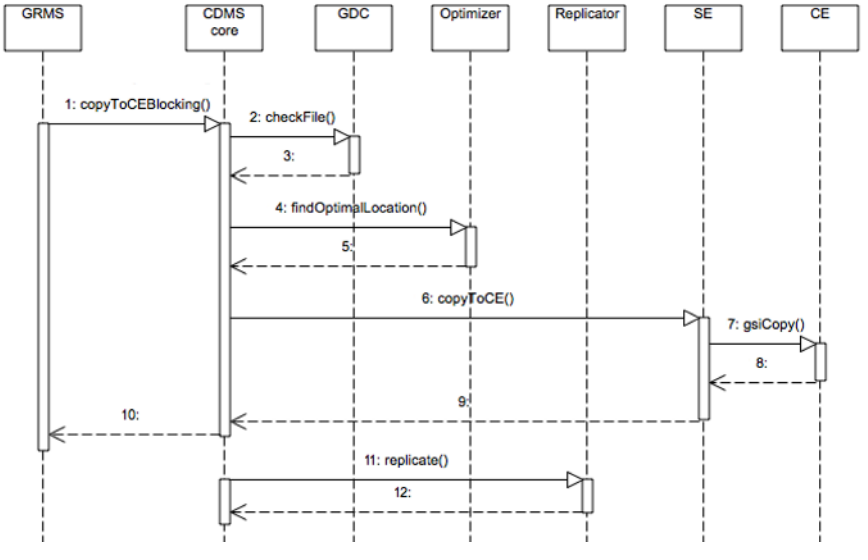


Fig. 4. Stage-out sequence diagram

parameters. The Management Module receives an ordered list of feasible Storage Elements, then delegates user credentials and requests the data retrieval to be performed (6), using the WebService interface of Storage Element. After the transfer is finished, the `copyFromCEBlocking` function returns (10), and GRMS continues with the job finalizing procedures, while CDMS initiates the replication process for the newly received data (11).

7 Conclusions

The CDMS is an advanced grid data management system, providing the end-user with efficient mechanisms for data transfer and storage. A very important feature of this system is its near complete transparency to users and seamless integration with the Resource Manager. On the other hand, advanced users are able to efficiently utilize the CDMS for inter-application data transfer, and to implement modules adapting CDMS to their needs.

References

1. Allcock, B., et al.: Data Management and Transfer in High Performance Computational Grid Environments. Parallel Computing Journal **28**, **5** (2002) 749–771
2. Bruin, R.P., Dove, M.T., Calleja, M., Tucker. M.G.: Building and Managing the eMinerals Clusters: A Case Study in Grid-Enabled Cluster Operation. Computing in Science & Engineering **7**, **6** (2005) 30–37
3. ClusteriX Project Home Page, http://clusterix.pcz.pl
4. GLOBUS Toolkit Homepage, http://www.globus.org/toolkit
5. GridSphere Project Home Page, http://gridsphere.org
6. gSOAP Project Home Page, http://www.cs.fsu.edu/ engelen/soap.html
7. Karczewski, K., Kuczynski, L., Wyrzykowski, R.: Secure Data Transfer and Replication Mechanisms in Grid Environments. Proc. Cracow Grid Workshop - CGW'03, Cracow, 2003, 190–196
8. Karczewski, K., Kuczynski, L., Wyrzykowski, R.: CDMS - ClusteriX Data Management System. Proc. Cracow Grid Workshop - CGW'04, Cracow, 2004, 241–247
9. Kunszt, P., Laure, E., Stockinger, H., Stockinger, K.: Advanced Replica Management with Reptor. Lect. Notes in Comp. Sci. **3019** (2004) 848–855
10. SDSC Storage Resource Broker, http://www.sdsc.edu/srb/
11. Slota, R., Skital, L, Nikolow, D., Kitowski, J.: Algorithms for Automatic Data Replication in Grid Environment (this volume)
12. Valentin, O., Lombard, P., Lebre, A., Guinet, Ch., Denneulin, Y.: Distributed File System for Clusters and Grids. Lect. Notes in Comp. Sci. **3019** (2004) 1099–1104
13. Wyrzykowski, R., Meyer, N., Stroinski, M.: Concept and Implementation of ClusteriX: National Cluster of Linux Systems. Proc. LCI Int.Conf. on Linux Clusters: The HPC Revolution 2005, Chapel-Hill, NC, April 2005

Grid Access and User Interface in CLUSTERIX Project

Piotr Kopta[1,2], Tomasz Kuczynski[1,2], and Roman Wyrzykowski[2]

[1] Poznan Supercomputing and Networking Center
{pkopta, docentt}@man.poznan.pl
[2] Institute of Computer and Information Sciences,
Czestochowa University of Technology
roman@icis.pcz.pl

Abstract. This paper presents the SSH Session Server Framework for the dynamic generation of interfaces of user applications. The original goal of this framework, developed for the CLUSTERIX grid project [2], is to create a flexible portal interface that can be easily extended and adapted for utilization with different shell applications. The approach is based on describing command line interface models with XML files. During the development, the main pressure is put on separation of the visualization layer from the application logic, as well as providing possibility of the framework extension at run-time. The support for VRML, X3D, SVG, charts (JPEG, PNG) formats of visualization is also an important feature. Full language localization is supported with any type of the visualisation. The SSH Session Server Framework is integrated with the GridSphere portal framework [5] - one of the products of the GridLab project [4].

1 Introduction

In the CLUSTERIX national grid project [2], one of key tasks is to deliver a consistent, secure and easy-to-use interface for end-users and administrators, providing a high scalable and seamless integration with the "core" grid services.

This paper presents our solution to this task, namely the SSH Session Server Framework. The original goal of this software is to create a flexible web portal inteface that can be easily extended and adopted for utilization with different applications. During the development the main pressure is put on separation of the visualization layer from the application logic, as well as providing possibility of the framework extension at run-time. It is required to provide an easy adaptation of existing applications, and utilization of them seemlesly. These requirements constraint us to use the SSH protocol with SSH sessions and pseudo-terminals as a communication concept.

To create this framework, we utilize our experience gained during the development of the WebCI advanced portal interface [6] for distributed computing systems. In comparison with WebCI, one of new features is using XML application extensions, called by us parsers, which allow for describing rules of interaction between users and applications.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 249–257, 2006.

The paper is organized as follows. In Section 2 we introduce the architecture of the framework, while Section 3 presents the way how an application interface is described, using the XML format. Sections 4 and 5 present respectively two main parts of the framework: the SSH Session Server, and Portlets/Services layer. The paper finishes with final remarks in Section 6.

2 Framework Architecture

The SSH Session Server Framework is a system that allows for making dynamic transformations of Command Line Interface (CLI) into Graphical User Interface (GUI). This framework operates in a distributed way, since it allows for aggregating outputs of multiple applications executed on multiple resources.

The framework consists of two main parts (see Fig.1). The first one is the SSH Session Server - a multithreaded server implemented in Perl. This part of the framework is responsible for persistent SSH connections, and making transformations (XML into commands and input parameters, application output into XML, etc). The SSH Session Server Portlet and Services are the second part

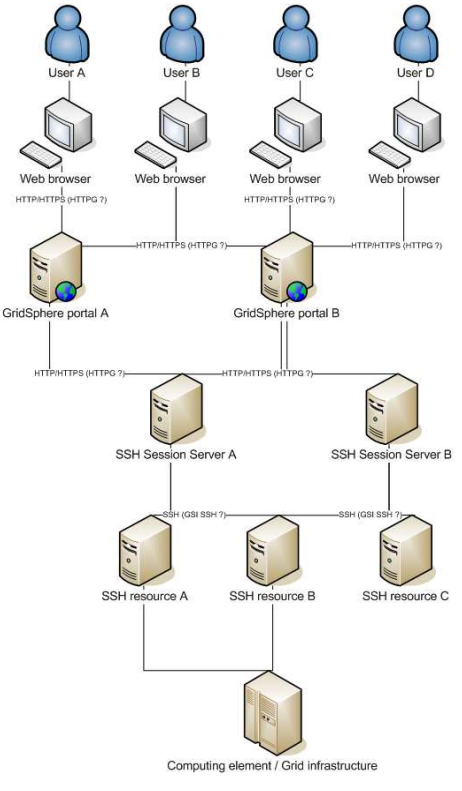


Fig. 1. Architecture of SSH Session Server Framework

of the framework. They are implemented as GridSphere API compliant portlet, and several GridSphere services. This part of the framework is responsible for the generation of forms with input parameters, transformation of input parameters into XML format, generation of output presentations using various formats - VRML, X3D, SVG, charts (JPEG,PNG), HTML, XML, etc.

Amongst many unique features of the framework, the most important are:

- adding user-defined interfaces at the portal run-time, so each user can customize his portal, either by uploadings created by him, or downloaded XML interface descriptions;
- easy adaptation of existing applications, so in most cases no modifications in applications are needed;
- seamless installation achieved as a result of communication by SSH with utilization of pseudo TTY (PTYs);
- XML-based application interface description, where XML describes the application interface model in the form of a finite-state automaton, as well as input parameters, output presentations, localization, etc.;
- support for HTML, VRML, X3D, SVG, charts (JPEG, PNG).

The SSH Session Server Framework is an outcome of collaboration between the CLUSTERIX [2] and GridLab [4] projects.

3 Description of Application Interface

Before executing any application through the SSH Session Server Framework, an XML-based application interface descriptor has to be prepared.

The descriptor consists of numerous ParsingChain elements. Each of ParsingChain elements describes one full interaction with the application, like input parameters, pattern for the application invocation, regular expressions for transformation of output to XML, output presentation XSLT style sheets bound to

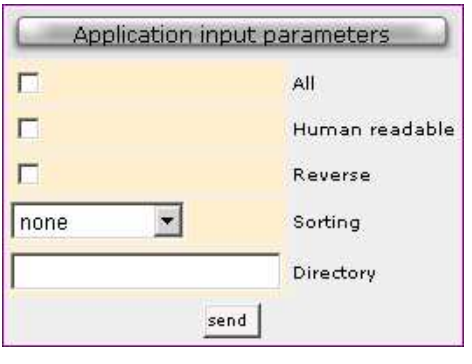


Fig. 2. Dynamically generated input form - Unix “ls” command

the next ParsingChain element, state transition table with mappings of prompt pattern to identifiers of the next ParsingChain element, etc.

Application interface descriptors are loaded and shared by users at run-time. Since each descriptor is versioned by the SSH Session Server with a timestamp, it is possible to copy it from one SSH Session Server to another SSH Session Server using portals. Descriptors copied in this way are treated by portals as identical, and can be utilized with any session that has been bound to a descriptor with the same ID and version.

Both the SSH Session Server itself and the SSH Session Server Framework portal side (portlet and services) utilize application interface descriptors to generate plugins.

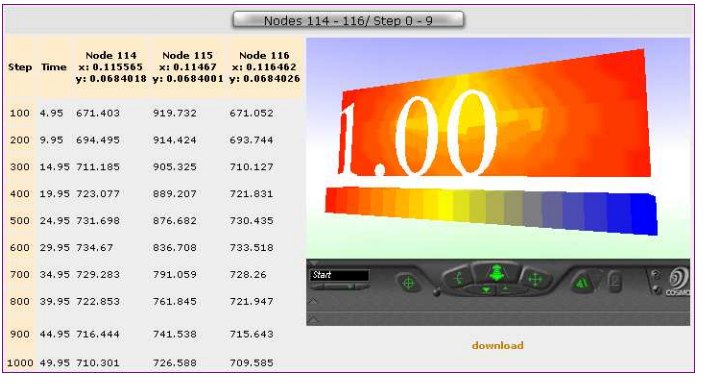


Fig. 3. Dynamically generated output presentation for the application "Heat Transfer Simulation"

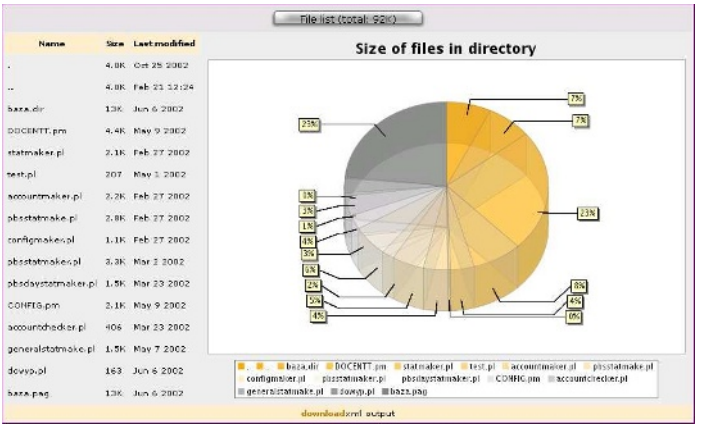


Fig. 4. Dynamically generated application output presentation - Unix "ls" command

In case of the SSH Session Server, plugins take care about transforming XML input parameters to real application parameters, wrapping them into the pattern for the application invocation, execution of command, transformation of output to XML (with regular expressions), and going over the finite state automaton generated for the application interface, in accordance with the state transition table.

The portal part of the framework is responsible for the presentation layer, including generation of forms with input parameters (Fig.2), transformation of parameters to XML, sending them to the SSH Session Server, and processing the application output (Fig.3 and Fig.4) in accordance with descriptors and parameters of output presentation, as well as the state transition table. This part also handles the proper application interface localization (complying with user settings in the portlet framework). Properties files with translations to different languages are bound to the application interface descriptor.

4 SSH Session Server

The SSH Session Server is a key element of our framework. Its main purpose is to intermediate between a user interface (WWW) and SSH sessions. It is responsible for creating and managing user account in the SSH Session Server, storing and managing user parsers, as well as managing SSH sessions with remote hosts. The SSH Session Server is implemented in Perl.

4.1 Users

Because the SSH Session Server can be used by many users, each user must possess an account with informations about created parsers, available parsers created by other users, and open SSH sessions.

The creation of such accounts is performed in two steps: (i) the user sends a request to create an account on the server; (ii) the server sends an email with confirmation to the user. If an answer does not arrive to the server in a specific time, the request is erased and the entire process has to start from the beginning. This procedure prevents remote robot attacks (programs that create plenty of accounts).

4.2 Parsers

The parser in the framework is a program that translates data incoming from the SSH protocol (from remote hosts) to XML format; the resulting document is forwarded to the SSH Session Server Portal.

The task of parsers is to validate data received from a remote program, and to transform that data to a more formalized format - XML. In a simple case, for example, for the “ls” Unix command, the parser has to listen for data until it encounters a system prompt (end of “ls” data), and translate the list of files and directories to the XML format. In a more complicated case, for example, for the “ssh” command, the parser must receive data until it encounters one of possible

cases: request for password, request to accept remote host public key or system prompt. In any of these cases, the parser has to transform the received text in a different way.

Parsers are defined by user as XML documents that contain: (a) a text that will be send by the SSH protocol to a remote host (usually command or program), (b) arguments for this text (values of arguments are defined when the parser is running), (c) what response the parser expects, and (d) rules for transforming incoming data to the XML format.

The internal structure of parsers is based on finite state automaton, since any parser can have states linked with states of a remote program. Transitions between these states are carried out after matching correct patterns with data received from SSH sessions.

In the framework, parsers are represented by Perl programs. Parsers created by one user can be private, when only the author of a parser has access to it, or public, when any user can use this parser.

4.3 SSH Sessions

A session in the SSH Session Server is a connection with a remote host by the SSH protocol. To send and receive data, we use pseudo-terminals (terminals associated with the "ssh" program).

To create a session, the user must pass remote user name, password for remote account, and host name. After successfully logging in, the user can run an available parser for this session (usually a remote command).

A session exists until the user closes it. This means that the user after logging out from the SSH Session Server portal, can log in again and use an earlier created session at any time.

4.4 Implementation

The SSH Session Server is implemented in Perl. The main reasons for this choice are: implementation speed and the very efficient regular expression implementation in Perl.

A running implementation of the Server is composed from several processes: listening processes that wait for user requests, as well as processes responsible for managing user/parser data, and data describing open sessions. The data transformations from the SSH output to XML format are placed in separate processes, so in any time several transformations can take place.

Most of volume-consuming data (parser data, data received by SSH) are stored in hashed files, so the memory footprint of the Server is not too big.

4.5 Security

Since user authorization data are sent without encryption, we choose the HTTP over SSL (https) protocol for communication between the SSH Session Server portal and Server.

5 Portlets and Services

SSH Session Server Portlets and Services are the second part of the framework. They are responsible for the generation of user interfaces.

An SSH Session Server Portlet is a GridSphere API compliant portlet that invokes appropriate methods of services described in details below.

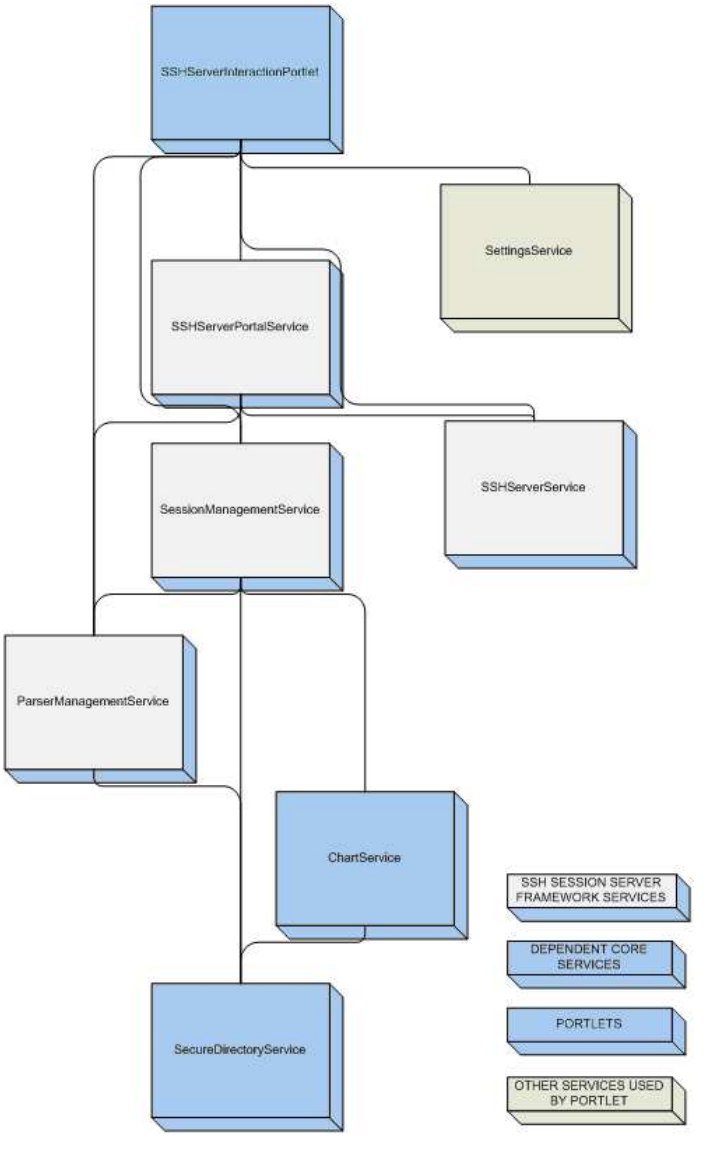


Fig. 5. Architecture of the portal part of the SSH Session Server Framework

5.1 Services

There are 4 main services (Fig.5.):

- SSHServerService: responsible for portal and SSH Session Server communication (1 to 1 mapping);
- ParserManagementService: responsible for the management of application descriptors on the portal side, input form generation, management of output visualization style sheets, dynamic loading of properties with localization;
- SessionManagementService: responsible for the management of session descriptors, transformation of input parameters, output visualization;
- SSHServerPortalService: the same interface as SSHServerService but taking care about invoking appropriate methods of ParserManagementService and SessionManagementService.

SessionManagementService and ParserManagementService are dependent on the following core services:

- SecureDirectoryService: responsible for caching content on the portal side, including session and application descriptors, XSLT style sheets, localization, and output visualization files; this service provides the portal with a secure space for each user;
- ChartService: responsible for generation and caching of charts, and utilized during output visualization.

5.2 Security

To prevent users from performing insecure operations in JVM of the portal, all XSLTs encapsulated in application interface descriptors are checked for occurrence of XML namespaces which correspond to names of Java classes that can be executed using Xalan [1] extensions.

The checking process utilizes XSLTs containing the list of allowed namespaces. The security policy for all XSLTs is: "what is not allowed is disallowed". In fact it is easy to extend the set of allowed classes with custom ones allowing for their utilization (from the XSLT level) during input forms generation and output visualization.

6 Final Remarks

Although building a grid system based on the SSH Session Server Framework is seamless (see Fig.1), a lot of effort is needed in order to fully integrate it with existing grids. Amongst others the following features are necessary:

- refactoring the SSH Session Server to GSI-enabled web service,
- support for gsissh,
- support for file transfers (both scp and gridftp),
- credential delegation.

The SSH Session Server Framework can be also used as a basis for the development of systems with wider functionalities, that support execution of end-user applications in various grid environments.

Acknowledgements

This work has been supported in part by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098 "CLUSTERIX - National Cluster of Linux Systems".

References

1. Apache Xalan Project Home Page, http://xalan.apache.org/
2. CLUSTERIX Project Home Page, http://www.clusterix.pl/
3. Grid Computing: Making the Global Infrastructure a Reality. F. Berman, G. Fox, A. Hey (editors), Wiley, 2003
4. GridLab Project Home Page, http://www.gridlab.org/
5. GridSphere Portal Framework Home Page, http://www.gridsphere.org/
6. Kuczynski, T., Wyrzykowski, R., Studzinski, G.: Cluster Monitoring and Management in the WebCI Environment. Lect. Notes in Comp. Sci. **3018** (2004) 375–382
7. Olas, T., Wyrzykowski, R.: Porting Thermomechanical Applications to Grid Environments. Lect. Notes in Comp. Sci. (this volume)
8. Wyrzykowski, R., Meyer, N., Stroinski, M.: Concept and Implementation of Clusterix: National Cluster of Linux Systems. 6th Int. Conf. on Linux Clusters: The HPC Revolution 2005, University of North Carolina, Chapel Hill, North Carolina, USA, 2005

An Architecture for Reconfigurable Iterative MPI Applications in Dynamic Environments

Kaoutar El Maghraoui, Boleslaw K. Szymanski, and Carlos Varela

Rensselaer Polytechnic Institute, Troy, NY 12180, USA
elmagk@cs.rpi.edu
http://www.cs.rpi.edu/

Abstract. With the proliferation of large scale dynamic execution environments such as grids, the need for providing efficient and scalable application adaptation strategies for long running parallel and distributed applications has emerged. Message passing interfaces have been initially designed with a traditional machine model in mind which assumes homogeneous and static environments. It is inevitable that long running message passing applications will require support for dynamic reconfiguration to maintain high performance under varying load conditions. In this paper we describe a framework that provides iterative MPI applications with reconfiguration capabilities. Our approach is based on integrating MPI applications with a middleware that supports process migration and large scale distributed application reconfiguration. We present our architecture for reconfiguring MPI applications, and verify our design with a heat diffusion application in a dynamic setting.

1 Introduction

A wide variety of computational environments are increasingly available to host the execution of distributed and parallel applications. Examples include large scale supercomputers, shared or dedicated clusters, grid environments, and metacomputing environments. Performance variability in such environments is the rule and not the exception. For application developers, this variability poses new challenges that go far beyond those of parallelism and scalability. The issue here is not only how to optimize large applications to run on a given set of distributed resources, but also how to maintain the desired performance anytime there is a change in the pool or characteristics of the resources or in the application's demands during its lifetime. In conventional distributed and parallel systems, achieving high performance was a matter of application-level scheduling or application-specific tunings. Such techniques relied on the following assumptions: 1) the application's performance model is known, 2) the number of resources is static, and 3) the characteristics of the resources are known. An obvious solution is dynamic application reconfiguration; i.e., adjusting the allocation of resources as the demand on the system and its availability varies.

While this new generation of computational environments presents multiple resource management challenges, it also provides an abundant pool of resources that is appealing to large scale and computationally demanding distributed applications. Examples are

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 258–271, 2006.

parallel computational science and engineering applications that arise in diverse disciplines such as astrophysics, fluid dynamics, materials science, biomechanics, or nuclear physics. These applications often involve simulating multi-scale problems and exhibit an insatiable need for computational resources. Many of these applications have been implemented with the Message Passing Interface (MPI) [1]. MPI is a widely used standard to develop parallel applications that harness several processors. However, the issues of scalability, adaptability and load balancing still remain a challenge. To maintain a good performance level, MPI applications need to be able to scale up to accommodate new resources or shrink to accommodate leaving or slow resources. Most existing MPI implementations assume a static network environment. MPI implementations that support the MPI-2 Standard [2, 3] provide some support for dynamic process management by allowing running processes to spawn new processes and communicate with them. However, developers still need to handle explicitly issues such as resource discovery, resource allocation, scheduling, profiling, and load balancing. Additional middleware support is therefore needed to relieve application developers from non-functional concerns while allowing high performance.

The Internet Operating System (IOS) [4, 5] is a distributed middleware framework that provides support for dynamic reconfiguration of large-scale distributed applications through opportunistic load balancing capabilities, resource-level profiling and application-level profiling. IOS has a modular nature that allows developers to create easily various reconfiguration policies. One key ingredient to application reconfiguration is the support for process migration. Applications should support process mobility to be able to benefit from IOS reconfiguration policies.

We target in this work the broad class of iterative applications. A large number of scientific and engineering applications exhibit an iterative nature. Examples include partial differential equation solvers, particle simulations, and circuit simulations [6]. We have chosen to experiment initially with the class of iterative applications for two reasons: this class is important in the scientific and engineering communities, and 2) It exhibits predictable profiling and reconfiguration points that could easily be automated through static software analysis or code-level annotations. To allow such applications to benefit from the reconfiguration capabilities of IOS middleware, we have developed a user-level library on top of MPI that allows process migration [7]. Our strategy achieves portability across different implementations of the the MPI standard. MPI/IOS is a system that integrates IOS middleware strategies with existing MPI applications. MPI/IOS adopts a semi-transparent checkpointing mechanism, where the user needs only to specify the data structures that must be saved and restored to allow process migration. This approach does not require extensive code modifications. Legacy MPI applications can benefit from load balancing features by inserting just a small number of calls to a simple application programming interface. In previous work [7], we described in detail the IOS architecture and evaluated our migration scheme. We take this work further in this paper by demonstrating the capability of IOS to adapt the class of iterative MPI applications to changing load conditions.

The remainder of the paper is organized as follows. Section 2 presents related work. In Section 3, we give an overview of the IOS middleware. Section 4 presents the MPI/IOS architecture with details on how MPI has been extended to support reconfigu-

ration with IOS. Section 5 discusses the adopted reconfiguration policies. In Section 6, we present performance evaluation. We conclude with discussion and future work in Section 7.

2 Related Work

There are a number of conditions that can introduce computational load imbalances during the lifetime of an application: 1) the application may have irregular or unpredictable workloads from, e.g., adaptive refinement, 2) the execution environment may be shared among multiple users and applications, and/or 3) the execution environment may be heterogeneous, providing a wide range of processor speeds, network bandwidth and latencies, and memory capacity. Dynamic load balancing (DLB) is necessary to achieve a good parallel performance when such imbalances occur. Most DLB research has targeted the application level (e.g., [8, 9]), where the application itself continuously measures and detects load imbalances and tries to correct them by redistributing the data, or changing the granularity of the problem through domain repartitioning. Although such approaches have proved beneficial, they suffer from several limitations. First they are not transparent to application programmers. They require complex programming and are domain specific. Second, they require applications to be amenable to data partitioning, and therefore will not be applicable in areas that require rigid data partitioning. Lastly, when these applications are run on a dynamic grid, application-level techniques which have been applied successfully to heterogeneous clusters [8, 10] may fall short in coping with the high fluctuations in resource availability and usage. Our research targets middleware-level DLB which allows a separation of concerns: load balancing and resource management are transparently dealt with by the middleware, while application programmers deal with higher level domain specific issues.

Several recent efforts have focused on enhancing MPI run-time systems to adapt applications to dynamic environments. Adaptive MPI (AMPI) [11, 12] is an implementation of MPI on top of light-weight threads that balances the load transparently based on a parallel object-oriented language with object migration support. Load balancing in AMPI is done through migrating user-level threads that MPI processes are executed on. This approach limits the portability of process migration across different architectures since it relies on thread migration. Process swapping [13] is an enhancement to MPI that uses over-allocation of resources and improves performance of MPI applications by allowing them to execute on the best performing nodes. MPI process swapping has been also used for the class of iterative applications. Our approach is different in the sense that we do not need to over-allocate resources initially. Such a strategy, though potentially very useful, may be impractical in large-scale dynamic environments such as grids where resources join and leave and where an initial over-allocation may not be possible. We allow new nodes that become available to join the computational grid to improve the performance of running applications during their execution.

Other efforts have focused on process checkpointing and restart as a mechanism to allow applications to adapt to changing environments. Examples include CoCheck [14], starFish [15], MPICH-V [16], and the SRS library [17]. CoCheck, starFish, and MPICH-V support checkpointing for fault-tolerance, while we provide this feature to allow

process migration and hence load balancing. Our framework could be integrated with the sophisticated checkpointing techniques used in these projects to be able to support also non-iterative applications. SRS supports checkpointing to allow application stop and restart. Our work differs in the sense that we support migration at a finer granularity. Application-transparent process checkpointing is not a trivial task, could be very expensive, and is architecture-dependent as it requires saving the entire process state. Semi-transparent checkpointing provides a simpler solution and a more portable approach. It has been been proved useful for the important class of iterative applications [13, 17]. API calls are inserted in the MPI program that informs the middleware of the important data structures to save. This is an attractive solution that can benefit a wide range of applications and does not incur significant overhead since only relevant state is saved. The instrumentation of applications could be easily automated since iterative application have a common structure.

Several projects have actively investigated the issue of application adaptivity in grid environments. Examples include GrADS [18], AppLeS [19], Cactus [20], and GridWay [21]. We share several performance and scheduling ideas with these projects. Most of the strategies they have adopted rely on the application 's stop and restart mechanism; i.e., the entire application is stopped, checkpointed, migrated, and restarted in another hardware configuration. Although this strategy can result in improved performance in some scenarios, a more effective adaptivity could be achieved if migration is supported at a finer granularity. We address reconfigurability at the process-level of the application to increase flexibility.

3 Overview of the IOS Framework

The goal of IOS middleware is to provide effective decentralized middleware-triggered dynamic reconfiguration strategies that enable application adaptation to the constantly changing behavior of large scale shared networks. To distributed application developers who often lack the time and expertise to handle complex performance tunings, IOS is a promising approach that combines both ease of use and high performance.

Applications wishing to interact with IOS need to have a flexible structure that synergizes easily with the dynamic nature of shared networks. They should exhibit a large degree of processing and/or data parallelism for efficient use of the system and scalability to a large number of resources. We assume that every application consists of distributed and migratable entities. In the case of MPI application, such entities refer to MPI processes.

IOS reconfiguration mechanisms allow 1) analyzing profiled application communication patterns, 2) capturing the dynamics of the underlying physical resources, 3) and utilizing the profiled information to reconfigure application entities by changing their mappings to physical resources through migration. A key characteristic of IOS is that it adopts a decentralized strategy that avoids the use of any global knowledge to allow scalable reconfiguration.

The IOS architecture consists of distributed middleware agents that are capable of interconnecting themselves in various virtual topologies. We support both hierarchical and peer-to-peer (P2P) topologies. The first is more suitable in grid environments that

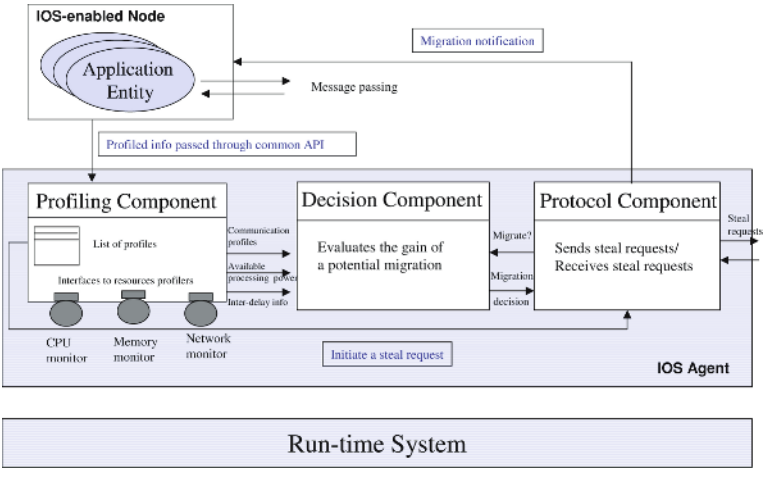


Fig. 1. Interactions between a reconfigurable application and the local IOS agents

usually consist of a hierarchy of homogeneous clusters while the second is more suitable for Internet environments that usually lack a well defined structure.

Figure 1 shows the architecture of an IOS agent and how it interacts with the application entities that are hosted in a given node. Every IOS agent consists of a profiling component, a decision component, and a protocol component:

- Dynamic Profiling Component
  Resource-level and application-level profiling is used to gather dynamic performance profiles about physical resources and application entities. Profiling generates performance profiles that are used in the reconfiguration decisions. Every application entity profiles its processing, communication, data accesses and memory usage. Every resource has also a profiling monitor that monitors periodically its utilization. Information about the resource's available CPU power, memory, and disk storage are measured and recorded periodically. The IOS architecture defines well-defined interfaces for profiling monitors which allow using several profiling technologies as long as they implement the appropriate interfaces. Examples include the Network Weather Service (NWS) [22] and MDS [23].
- Protocol Component
  The protocol component is responsible for inter-agent communication and virtual topology creation. The middleware agents form a virtual network. When new nodes join the network or existing nodes become idle, their corresponding protocol component contact peers randomly to steal work [24]. This strategy aids in advertising new or existing resources as they become available. Every work stealing request carries with it performance-related information about the node that originated the request.
- Reconfiguration Module
  Upon receiving a work stealing request, the reconfiguration module tries to evaluate whether there are any potential candidate entities that will benefit from migration

to the originator of the request. The decision is done by examining the performance of the application entities in the local node and the predicted performance in the remote node. If a gain is predicted, the local protocol component notifies the selected entities to migrate.

Applications communicate with the IOS middleware through clearly defined interfaces that permit the exchange of profiled information and reconfiguration requests. Figure 2 shows the profiling interface that allows applications to notify the middleware about all communication exchanges. The profiling component maintains a list of all application entities hosted in its local run-time system. For every application entity, the list maintains information about all other entities that have been exchanging messages with it and their frequency of communication. In other words, this list represents a weighted communication subgraph of every application entity where the nodes represent application entities, the edges represent communication links, and the weights represent communication rates. Every entity has a unique name (UAN) associated with it and a universal locator (UAL) that keeps track of the current location where the entity is currently hosted. A UAN stands for Universal Actor Name while a UAL stands for Universal Actor Locator. We adopt here the naming conventions of the SALSA [25] language. SALSA is a language for developing actor oriented application. It is a dialect of Java with high-level constructs for universal naming, remote message sending, and coordination. IOS has been prototyped using both SALSA and Java.

```
//The following methods notify the profiling agent of entities
//entering and exiting the local run-time system due
//to migration or initial entity startup.

public void addProfile(UAN uan);
public void removeProfile(UAN uan);
Public void migrateProfile(UAN uan, UAL target)

//The profiling agent updates its entity profiles based
//on message sending with these methods

public void msgSend(UAN uan, UAL targetUAL, UAL sourceUAL,
                    int msg_size);

//The profiling agent updates its entity profiles based
//on message reception with this method

public void msgReceive(UAN uan, UAL targetUAL, UAL sourceUAL,
                       int msg_size);

//The following methods notify the profiling agent of the start
//of a message being processed and the end of a message being processed,
//with a UAN or UAL to identify the sending entity

public void beginProcessing(UAN uan, UAL targetUAL, UAL sourceUAL,
                            int msg_size);
public void endProcessing(UAN uan, UAL targetUAL, UAL sourceUAL,
                          int msg_size);
```

Fig. 2. IOS Profiling API

4 Reconfiguring MPI Applications with IOS

4.1 Process Migration Support

In MPI, any communication between processes needs to be done as part of a *communicator*. An MPI communicator is an opaque object with a number of attributes, together with simple functions that govern its creation, use and destruction. An *intracommunicator* delineates a communication domain which can be used for point-to-point communications as well as collective communication among the members of the domain. On the other hand, an *intercommunicator* allows communication between processes belonging to disjoint intracommunicators.

We achieve MPI process migration by rearranging MPI communicators. Migration is performed by a collaboration of all the participating MPI processes. It has to be done at a point where there are no pending communications. Process migration requires careful update of any communicator that involves the migrating process. The class of iterative application have natural barrier points. When necessary, we perform all reconfiguration at the beginning of each iteration. A migration request forces all running MPI processes to enter a reconfiguration phase where they all cooperate to update their shared communicators. The migrating process spawns a new process in the target location and sends it its local checkpointed data.

Process migration and checkpointing support have been implemented as part of a user-level library. This approach allows portability across several vendor MPI implementations that support the MPI-2 process spawning feature since the library is implemented entirely in the user space and does not require any infrastructural changes. The library is called PCM (Process Checkpointing and Migration).

4.2 Profiling MPI Application

MPI processes need to send periodically their communication patterns to their corresponding IOS profiling agents. To achieve this, we have built a profiling library that is based on the MPI profiling interface (PMPI). The MPI specification provides a general mechanism for intercepting calls to MPI functions. This allows the development

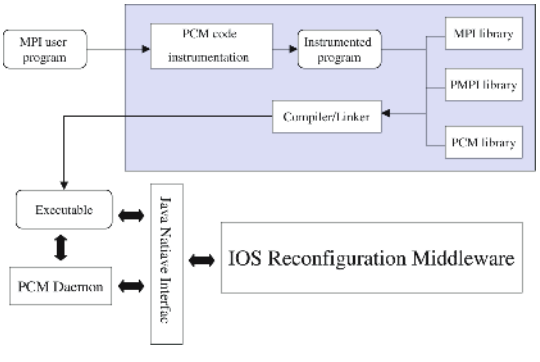


Fig. 3. Library and executable structure of an MPI/IOS application

of portable performance analyzers and other tools without access to the MPI implementation source code. The only requirement is that every MPI function be callable by an alternate name (`PMPI_Xxxx` instead of the usual `MPI_Xxxx`.). The built profiling library intercepts all communication methods of MPI and sends any communication event to the profiling agent.

All profiled MPI routines call their corresponding `PMPI_Xxxx` and, if necessary, PCM routines. Figure 3 shows the library structure of the MPI/IOS programs. The instrumented code is linked with the profiling library PMPI, the PCM library, and a vendor MPI implementation's library. The generated executable passes all profiled information to the IOS run-time system through Java Native Interface (JNI) and also communicates with a local PCM Daemon (PCMD) that is started in every node. The PCMD is responsible for storing local checkpoints and passing reconfiguration decisions across a socket API from the IOS agent to the MPI processes. For more details about the PCMD, the reader is referred to [7].

4.3 A Simple Scenario for Adaptation

MPI applications interact with each other, with the checkpointing and migration services provided by the PCM library, and with the profiling and reconfiguration services provided by IOS agents. Walking through the simple scenario of an application adaptation that is shown in Figure 4 further explains these interactions. The example illustrates the execution of a parallel MPI application that is composed of n processes running on n different processors.

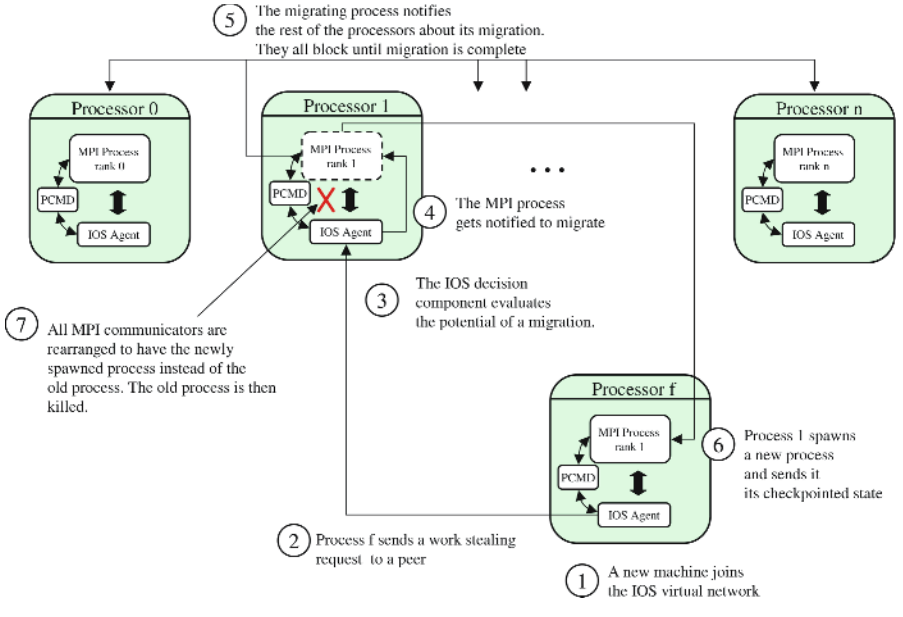


Fig. 4. A reconfiguration scenario of an MPI/IOS application

1. An available processor joins the IOS virtual network.
2. The new processor starts requesting work from its peers.
3. Processor 1 receives the work stealing request. The decision component in its local IOS agent predicts that there will be a gain migrating process 1 to the remote processor f.
4. MPI process 1 gets notified of a migration event.
5. At the beginning of the next iteration, the migrating process broadcasts a message to the rest of the processors so that they enter a reconfiguration phase.
6. The migrating process checkpoints its local state, and spawns an image of itself in the remote processor f. The local PCMD takes care of transferring the checkpointed state to the newly created process and notifying the rest of the processes.
7. As a part of completing the migration process, the PCM library takes care of rearranging the `MPI_COM_WORLD` communicator by removing the old process and including the new one. The newly created process gets assigned rank 1.

5 Reconfiguration Policies

The reconfiguration policies use the application's characteristics and the underlying resources' characteristics to evaluate whether there will be any performance gain through migration. Resources such as storage, CPU processing power, network bandwidth, and network latencies are ranked according to their importance to the running application. For instance, if the application is computationally intensive, more weight is given to the CPU processing power. If the application is communication intensive and the messages exchanged have large sizes, more weight is given to the network bandwidth. Whereas, if it is communication intensive with small exchanged message sizes, the network latency is very important.

Let P be the set of m processes running on a local node n.

$$P = \{p_0, p_1, ..., p_m\}$$

Let R be a set of resources available in the local node n.

$$R = \{r_0, r_1, ..., r_l\}$$

Let ω be the weight assigned to a given resource r_i based on its importance to the performance of the set P.

$$0 \leq \omega(r_i, P) \leq 1 \; and \; \sum_{i=0}^{l} \omega(r_i, P) = 1$$

Let $a(r, f)$ be the amount of resource r available at foreign node f, $u(r, l, A)$ be the amount of resource r used by the processes P at local node l, $M(P, l, f)$ be the estimated cost of migration of the set P from l to f, and $L(P)$ be the average life expectancy of the set of processes P. The predicted increase in overall performance Γ gained by migrating P from l to f, where $\Gamma \leq 1$ is:

$$\Delta_{r,l,f,P} = \frac{a(r, f) - u(r, l, P)}{a(r, f) + u(r, l, P)} \tag{1}$$

$$\Gamma = (\sum_{r} \omega(r, P) * \Delta_{r,l,f,P}) - (\frac{M(P,l,f)}{(10 + log(L(P))}) \quad (2)$$

When a node l receives a work stealing request from a node f, a process or a group of processes will be migrated if the evaulation of the gain Γ is positive. The larger this value is, the more beneficial the migration is.

It is important to factor in Equation 2 the remaining life expectancy of the group of processes P. The intuition behind this is that processes who are expected to have a short remaining life are not migrated because the cost of the reconfiguration might exceed the benefit over their remaining life expectancy. For the case of iterative applications, we estimate the life expectancy by looking at the number of remaining iterations multiplied by the average time each iteration takes in the current node.

6 Performance Evaluation

6.1 Application Case Study

We have used a fluid dynamic problem that solves heat diffusion in a solid for testing purposes. This applications is representative of the large class of highly synchronized iterative mesh-based applications. The application has been implemented using C and MPI and has been instrumented with PCM library calls. We have used a simplified version of this problem to evaluate our reconfiguration strategies. A two-dimensional mesh of cells is used to represent the problem data space. The mesh initially contains the initial values of the mesh with the boundary values. The cells are uniformly distributed among the parallel processors. At the beginning, a master process takes care of distributing the data among processors. For each iteration, the value of each cell is

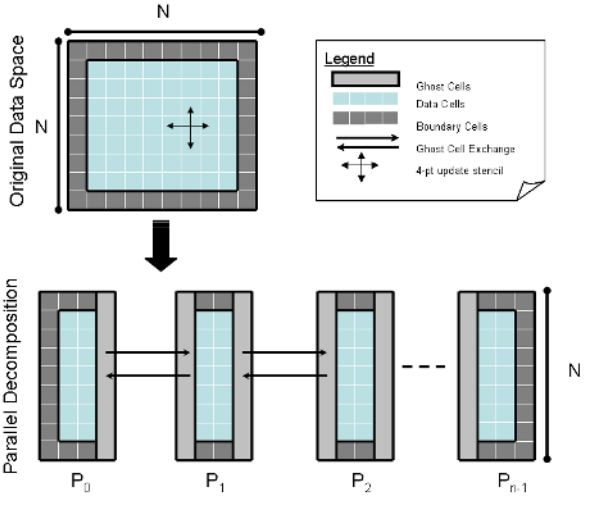


Fig. 5. Parallel decomposition of the 2D heat diffusion problem

calculated with the values of its neighbor cells. Each cell needs to maintain a current version of the values of its neighboring cells. To achieve this, processors exchange values of the neighboring cells, also referred to as ghost cells. To sum up, every iteration consists of doing computation and exchange of ghost cells from the neighboring processors. Figure 5 shows the structure of the parallel decomposition of the heat diffusion problem.

6.2 Adaptation Experiments

We have used the iterative 2-dimensional heat diffusion application to evaluate the reconfiguration capabilities of the MPI/IOS framework. The original MPI code was instrumented with PCM calls to enable checkpointing and migration. For the experimental testbed we used a 4-dual node cluster of SUN Blade 1000 machines. Each node has a processing speed of 750M cycles per second and 2 GB of memory. For comparative purposes, we used MPICH2 [26], a free implementation of the MPI-2 standard. We emulated a shared and dynamic environment with varying load conditions by introducing artificial load in some of the cluster nodes and varying it periodically.

We conducted two experiments using the heat application using the MPI/IOS framework and MPICH2 under similar load conditions. For both experiments, we started running the application, then we kept increasing the load in some of the cluster nodes and watched how the application's performance was affected in each case.

The first experiment was conducted with MPI/IOS. Figure 6 shows the performance of the application. We started running the application with 8 processes on 4 processors. the remaining 4 processors joined the virtual IOS network gradually. We started increasing gradually the load on one of the cluster nodes (two processors) participating in the computation. One node joined the virtual IOS network around iteration 1000 and started sending work stealing requests. This caused one process to migrate to the new machine and to reduce the load on its original hosting node. We notice an increase in the application throughput. The load in the slow machine increased even further around iteration 2500. Around iteration 3000, a fourth node joined the virtual network

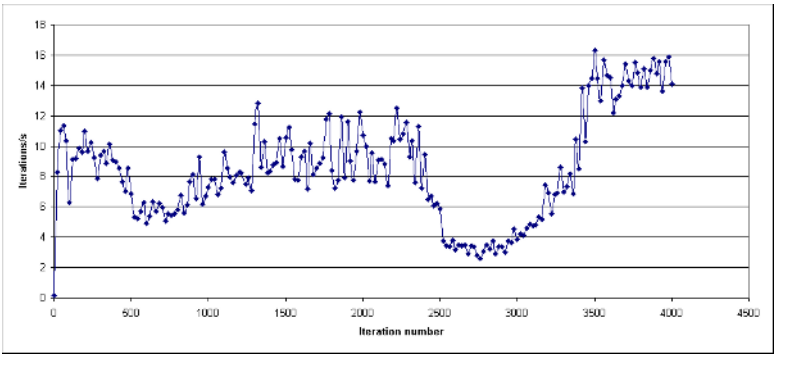


Fig. 6. Performance of the the two-dimensional heat simulation application using the reconfiguration mechanism of MPI/IOS. The experiment was conducted on a 4 dual-processor SUN blade 1000 cluster.

and started sending work stealing packets. At this point, two processes migrated to this machine. This caused the slow processors to be eliminated from the computation. The application ended up using a total of the 6 best available processors, which caused a substantial increase in its performance. The total execution time of the application was 645.67s.

Figure 7 shows the performance of the application using MPICH2. We emulated the same load conditions as in the first experiment. With no ability to adapt, the application was stuck with the first hardware configuration and experienced a constant slowdown in its performance. The highly synchronized nature of this application causes it to run as fast as the slowest processor. The application took 1173.79s to finish, about an 81.8% decrease in performance compared to the adaptive execution.

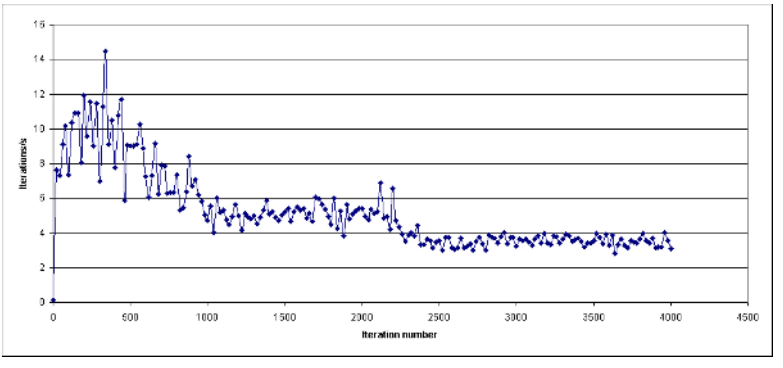


Fig. 7. Performance of the the two-dimensional heat simulation application using MPICH2. The experiment was conducted on a 4 dual-processor SUN blade 1000 cluster.

7 Discussion and Future Work

We presented in this paper an architecture that enhances the performance of iterative applications under dynamically changing conditions. We implemented a user-level library that adds checkpointing and process migration features to existing iterative MPI applications. We also integrated our library with a middleware for reconfigurable computing. The experimental evaluation has demonstrated the importance of augmenting long-running MPI application with reconfiguration capabilities to achieve high performance.

This work opens up several interesting future directions. One direction is the evaluation of different reconfiguration policies and how effective they are in dynamic execution environments. The sensitivity of these policies to application's characteristics needs also to be investigated. Another important direction is extending our framework beyond the class of iterative applications. This will require developing more sophisticated checkpointing and migration policies. We plan also to evaluate our framework on larger networks with interesting and realistic load characteristics and network latencies.

Acknowledgments

The authors would like to acknowledge the members of the Worldwide Computing Laboratory at Rensselaer Polytechnic Institute. In particular, our special thanks go to Travis Desell for his contributions to the IOS middleware. This work has been supported by the following grants: IBM SUR Award 2003, IBM SUR Award 2004, NSF CAREER Award No. CNS-0448407, and NSF-INT 0334667.

References

1. Message Passing Interface Forum: MPI: A message-passing interface standard. The International Journal of Supercomputer Applications and High Performance Computing **8**(3/4) (1994) 159–416
2. Gropp, W., Lusk, E.: Dynamic process management in an MPI setting. In: Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing, IEEE Computer Society (1995) 530
3. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface (1996)
4. Desell, T., Maghraoui, K.E., Varela, C.: Load balancing of autonomous actors over dynamic networks. In: Hawaii International Conference on System Sciences, HICSS-37 Software Technology Track, Hawaii (2004)
5. Maghraoui, K.E., Desell, T., Varela, C.: Network sensitive reconfiguration of distributed applications. Technical Report CS-05-03, Department of Computer Science, Rensselaer Polytechnic Institute (2005)
6. Fox, G.C., Williams, R.D., Messina, P.C. In: Parallel Computing Works. Morgan Kaufmann Publishers, San Fransisco, CA (1994) Available at http://www.npac.syr.edu/pcw/.
7. Maghraoui, K.E., Desell, T., Szymanski, B.K., Teresco, J.D., Varela, C.A.: Towards a middleware framework for dynamically reconfigurable scientific computing. In Grandinetti, L., ed.: Grid Computing and New Frontiers of High Performance Processing. Elsevier (2005) to appear.
8. Elsasser, R., Monien, B., Preis, R.: Diffusive load balancing schemes on heterogeneous networks. In: Proceedings of the Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM Press (2000) 30–38
9. Flaherty, J.E., Loy, R.M., Özturan, C., Shephard, M.S., Szymanski, B.K., Teresco, J.D., Ziantz, L.H.: Parallel structures and dynamic load balancing for adaptive finite element computation. Applied Numerical Mathematics **26** (1998) 241–263
10. Teresco, J.D., Faik, J., Flaherty, J.E.: Resource-aware scientific computation on a heterogeneous cluster. Technical Report CS-04-10, Williams College Department of Computer Science (2005) To appear, *Computing in Science & Engineering*.
11. Bhandarkar, M.A., Kale;, L.V., de Sturler, E., Hoeflinger, J.: Adaptive load balancing for MPI programs. In: Proceedings of the International Conference on Computational Science-Part II, Springer-Verlag (2001) 108–117
12. Huang, C., Lawlor, O., Kalé, L.V.: Adaptive MPI. In: Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 03), College Station, Texas (2003)
13. Sievert, O., Casanova, H.: A simple MPI process swapping architecture for iterative applications. International Journal of High Performance Computing Applications **18**(3) (2004) 341–352

14. Stellner, G.: Cocheck: Checkpointing and process migration for MPI. In: Proceedings of the 10th International Parallel Processing Symposium, IEEE Computer Society (1996) 526–531
15. Agbaria, A., Friedman, R.: Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations. In: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society (1999) 31
16. Bosilca, G., Bouteiller, A., Cappello, F., Djilali, S., Fedak, G., Germain, C., Herault, T., Lemarinier, P., Lodygensky, O., Magniette, F., Neri, V., Selikhov, A.: MPICH-V: toward a scalable fault tolerant mpi for volatile nodes. In: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, IEEE Computer Society Press (2002) 1–18
17. Vadhiyar, S.S., Dongarra, J.J.: SRS - a framework for developing malleable and migratable parallel applications for distributed systems. In: Parallel Processing Letters. Volume 13. (2003) 291–312
18. Vadhiyar, S., Dongarra: Self adaptivity in grid computing. Concurrency and Computation: Practice and Experience **17**(2-4) (2005) 235–257
19. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive Computing on the Grid Using AppLeS. IEEE Trans. Parallel Distrib. Syst. **14**(4) (2003) 369–382
20. Allen, G., Dramlitsch, T., Foster, I., Karonis, N.T., Ripeanu, M., Seidel, E., Toonen, B.: Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In: Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, ACM Press (2001) 52–52
21. Huedo, E., Montero, R.S., Llorente, I.M.: A framework for adaptive execution in grids. Softw. Pract. Exper. **34**(7) (2004) 631–651
22. Wolski, R.: Dynamically forecasting network performance using the network weather service. Cluster Computing **1**(1) (1998) 119–132
23. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE Symposium On High Performance Distributed Computing. (2001)
24. Blumofe, R.D., Leiserson, C.E.: Scheduling Multithreaded Computations by Work Stealing. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94), Santa Fe, New Mexico (1994) 356–368
25. Varela, C., Agha, G.: Programming dynamically reconfigurable open systems with SALSA. ACM SIGPLAN Notices. OOPSLA'2001 Intriguing Technology Track Proceedings **36**(12) (2001) 20–34 `http://www.cs.rpi.edu/~cvarela/oopsla2001.pdf`.
26. Argone National Laboratory: (MPICH2, `http://www-unix.mcs.anl.gov/mpi/mpich2`)

Parallel Computing in Java: Looking for the Most Effective RMI Implementation for Clusters*

Rafał Metkowski and Piotr Bała

Faculty of Mathematics and Computer Science,
N. Copernicus University, Chopina 12/18, 87-100 Toruń, Poland
{rafmet, bala}@mat.uni.torun.pl

Abstract. The importance of Java as a language for high performance computing is significant. The latest Java virtual machine technology is closing the gap between computational speed of object oriented and procedural languages like C or Fortran. Because current computers are based on the parallel execution, the Java communication capabilities become more important. The main high level mechanism for Java distributed computing is RMI (Remote Method Invocation) technology which gains significant critique because of low performance. In this paper we describe transport layer implementation for KaRMI based on PMv2 communication library which is part of the SCore Cluster System Software. Presented implementation allows to use efficiently RMI technology with the GigabitEthernet cards. The performance of our solution is compared with the standard RMI and original KaRMI.

1 Introduction

The importance of Java as a language for high performance computing is continuously increasing. The main reason is increasing popularity of Java and spreading of knowledge, not only amongst computer scientists but also among physicists, chemists and even biologists who are developing software to solve problems in the specific application areas. It has been common believe that Java will be efficient tool for interface design while computational kernel will be still written in traditional languages like Fortran or C. This was caused by the well known difference between procedural and object oriented programming models.

Resent comparisons made with respect to computational performance show that C++ generally outperforms Java. However Java is easier to use, leads to more robust code and shorter development times. With the advent of newer just-in-time compilers, Java performance is now comparable to C++ and the latest Java virtual machine technology is closing the gap. Eventually Java should be a good compromise between efficient algorithm performance and effective application development [1].

The single node performance is important, but nowadays most of the CPU cycles is delivered by the parallel and massively parallel computers, in particular

* This work has been partially supported by the State Committee for Scientific Research (4 T11F 009 25).

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 272–277, 2006.

large PC clusters. Therefore the user is interested in tools which allow for efficient parallel programming. The parallel code development for scientific computing is dominated by the message passage approach implemented in the form of the MPI library. Up to now, number of MPI bindings has been developed for Java [2, 3] ontop of available transport layers such as plain sockets or RMI. Unfortunately, most of the solutions has not been widely adopted by the programmers, most probably due to the fact that they does not fit well into Java objects paradigm. Low performance was additional reason.

The important advantage of Java is its networking capability which allows for easy development of distributed applications. Java supports different levels of network communication, staring form plain sockets up to application level tools. The main high level mechanism for Java distributed computing is RMI (Remote Method Invocation). RMI, introduced in the Java JDK 1.1, is designed as object oriented continuation of idea of Remote Procedure Calls (RPC). Unfortunately it has two main disadvantages: usage for the code parallelization is quite complicated and the performance of standard RMI is not satisfactory.

These disadvantages of the RMI gave motivation for development of new tools which aim is to simplify parallel programming in Java and to increase performance. In particular, parallelization can be greatly simplified using JavaParty [4] or JavaSymphony [5]. It extends Java language only by adding new keyword *remote*. New keyword can be used in class definition and it indicates that objects of this class are not limited to only one Java Virtual Machine (JVM).

Since raw socket programming require protocol design and implementation, most of the high level approaches to the parallel programming in Java choose RMI as the communication mechanism. As mentioned before, standard RMI implementation has limited performance which limits usability of tools developed ontop of it. Poor performance of standard RMI is caused by a slow serialization process and by the time required for TCP/IP based communication. Alternative RMI implementations with improved performance were developed by the number of authors. The most promising one is KaRMI [6]. Beside faster serialization it also separated transport layer which is not limited to TCP/IP. KaRMI comes with the transport layer implementation for the TCP/IP sockets, Myrinet/GM and Myrinet/ParaStation. Nowadays Gigabit Ethernet is quite common interconnect for PC clusters but in this case KaRMI has to use TCP/IP protocol since none of the low latency communication systems like M-VIA [7], Gamma [8] or SCore [9] is supported.

In this paper we describe transport layer implementation for the KaRMI based on PMv2 communication library (part of the SCore Cluster System Software) which allows to use software with the GigabitEthernet cards. The performance of our solution is compared with the standard RMI and original KaRMI.

2 KaRMI

KaRMI is a drop-in replacement for Java RMI. It is written almost completely in Java with some transport part which uses native calls to the communication

libraries. In the RMI arguments to remote methods are passed "by-value". Because of this, the arguments which are usually quite complicated object, must be serialized. After serialization they are sent to the remote JVM where deserialization takes place. Object serialization process has big influence on the RMI performance. According to the estimates it can take between 25 and 50% of the time needed for the remote method invocation. In result, any communication has large latency which prohibits efficient parallel computations. In order to improve performance the KaRMI authors introduced new efficient serialization mechanism called *UKA-serialization* [10].

Standard Java RMI uses JDK-serialization implemented in JVM. It uses type introspection mechanizm to convert object and all of its primitive fields to the byte array representation. A complete graph of objects that refer to non-primitive fields is also converted. UKA-serialization approach is different. Programmer is required to provide explicit marshaling and unmarshaling procedures. In this case one can avoid time consuming creation of the graph dependency and retrospection paths and the serialization can be performed faster. The another advantage is smaller size of the object transferred to the remote JVM ("Slim Encoding of Type Information").

The main advantage of the JDK-serialization is that it allows to recreate object from the byte representation in a newer Java release even when byte code of the class has changed. UKA-serialization is lacking such capability, however it is not important when computations are performed on the clusters since we can easily guarantee that code of the class and JDK versions are the same on all nodes.

Another important feature of KaRMI is separation of the transport layer. It allows to add interfaces for the different types of network hardware. Such interface in KaRMI is called *technology*. Configuration file describes available technologies and KaRMI chooses the best available technology to connect to the remote host. In contrast, Java RMI is limited to the TCP/IP transport layer only.

3 PMv2 Communication Library

SCore is a complete parallel programming environment developed for the Linux clusters. It provides many tools to manage the whole system as well as to run parallel jobs. One of them is MPICH-SCore - modified MPICH implementation that uses PMv2 communication library and modified startup procedure that follows SCore single system image model. Other important tools are PVM-SCore and SCASH - Software Distributed Shared Memory. In all cases the communication is based on the PMv2 which provides efficient low-latency transport.

The PMv2 Library supports many types of hardware. Drivers for shared memory, Myrinet, Ethernet and UDP are available. Important advantage of the PMv2 library is that it works with any ethernet card with properly written driver. Moreover, the card driver has not to be modified. The PMv2 library is not visible to the user since he is using higher level tools such as SCASH or MPICH-Score. However, the PMv2 API is available and communication functions can be called directly.

The library API contains functions for message passing and remote memory access. The message passing ones are very primitive and allow to send/receive packet smaller than MTU. It is possible to use system function *select* for polling but special functions from PMv2 API need to be called before and after select. There are no message types or any other means for selective receive available.

4 Implementation

PMv2 library can be used as the fast and low latency communication layer, we have used it to develop new *technology* for KaRMI. The Java part of our RMI implementation is based on *PSPTechnology* included in the KaRMI. It was expected that this will improve RMI speed and reduce latency. To follow KaRMI nomenclature our implementation has been called *SCoreTechnology*.

The simplest way to develop a new KaRMI technology is to create a class that extends *uka.karmi.stream.StreamTechnology*. In this case two main methods that have to be implemented are *initExportPoint* and *createConnection*. Former one is responsible for the creation of new export point while the later one creates connection to the remote export point. Since we have decided that our class is subclass of the *StreamTechnology* we also had to create our implementations of the *InputStream* and the *OutputStream*.

Because there is no Java API for the PMv2 library we had to use Java Native Interface (JNI) to provide cooperation between Java classes and communication library. When new export point is formed, a new thread is created to service incoming connections to this export point. Because of lack of support for the selective message receiving we have developed system of a virtual ports. Each port has its own cyclic buffer. If received message is addressed to the different export point it is stored in the cyclic buffer and then can be read by another thread. Posix mutexes and condition variables are used to synchronize access to the cyclic buffers. Since functions from PMv2 API are not thread safe the other mutexes are used to synchronize send and receive operations.

5 Performance

As it has been shown previously[11] the main advantage of SCore and PMv2 library is latency reduction which results in the speedup of the parallel codes. To check performance of our solution we have used simple code to create remote object and call its `ping` and `ping(byte b[])` methods. Size of the array `b` was changed to test *SCoreTechnology* with the different amounts of data. The results have been compared to the situations where original KaRMI and standard Java MPI have been used.

Tests have been performed on a cluster with 8 SMP nodes. Each node contains two pentium-III 450 MHz processors. Operating system on all nodes was Redhat 9 and SCore Cluster Software version was 5.8.2.

Obtained results for the ping test are shown in the Table 1. As it was expected *SCoreTechnology* allows to execute remote method much faster than the

Table 1. The invocation time of the remote method depends on the size of the array passed as argument. All times are given in the miliseconds. The last column contains speedup of the *SCoreTechnology* implementation vs. KaRMI *SocketTechnology*.

Size of the b[] array	Java RMI	KaRMI *SocketTechnology*	KaRMI *SCoreTechnology*	ScoreTechnology Speedup
w/o argument	470.5	293.8	224.2	1.31
500	688.7	396.0	325.1	1.22
1000	721.3	459.9	386.9	1.19
1500	755.6	423.5	360.4	1.18
2000	771.0	431.1	374.6	1.15
2500	856.1	446.7	390.5	1.14
3000	892.8	485.4	454.4	1.07
6000	1053.7	599.1	603.9	0.99
9000	1283.1	726.3	967.7	0.75
12000	1477.5	890.0	878.1	1.01
15000	1719.2	1011.3	1008.2	1.00

Table 2. The synchronization time for the different number of nodes. All times are given in the miliseconds

Number of nodes	Java RMI	KaRMI *SocketTechnology*	KaRMI *SCoreTechnology*	*ScoreTechnology* Speedup
2	505.4	269.9	310.2	0.87
3	1027.0	596.9	631.6	0.94
4	1552.4	922.0	953.0	0.97
5	2159.0	1241.1	1271.8	0.98
6	2763.4	1566.6	1592.8	0.98

original KaRMI. The invocation time, which in this case corresponds to the communication latency is reduced by 30%. Compare to the Java RMI reduction is even larger up to 50%. This results confirm that our implementation reduces significantly communication latency even compare to the KaRMI.

The ping test involves only two nodes of the cluster and do not reflect behavior of the multiprocessors systems. To test collective operations we have tested cluster synchronization performed with the exchange of messages. Short message has been send sequentially to the nearby node. The total time to travel through all nodes has been measured and is presented in the Table 2. The results show that both KaRMI with *SocketTechnoloy* and *SCoreTechnology* have similar performance, significantly better than standard RMI.

6 Conclusions

We have developed new communication mechanism for the Java RMI implementation dedicated for Linux clusters. This technology allows for fast and low

latency RMI communication on the Linux clusters equipped with the SCore Cluster Tools. We have measured performance and latency of the RMI calls. The results show that our implementation performs better than original Java RMI from SUN or KaRMI. Since the developed communication technology is based on the PMv2 library it does not use TCP/IP stack. Its usability is limited to the single clusters and cannot be used for the resources connected over WAN. Presented results show that despite common critique of the RMI performance, the latency can be reduced and RMI can be efficiently used as communication mechanism both directly or as low level communication layer for tools such as JavaParty or JavaSymphony.

References

1. R. A. Vivanco, N. J. Pizzi Scientific computing with Java and C++: a case study using functional magnetic resonance *Software: Practice and Experience* vol. 35 no. 3, pp. 237 – 254, 2004
2. S. Mintchev V. Getov Towards Portable Message Passing in Java: Binding MPI In: M. Bubak, J. Dongarra, J. Wasniewski editor, *Proceedings of the 4th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface Lecture Notes In Computer Science 1332* pp. 135 – 142, 1997
3. B. Carpenter, V. Getov, G. Judd, A. Skjellum, G. Fox MPJ: MPI-like message passing for Java *Concurrency: Practice and Experience* vol. 12, no. 11, pp. 1019 – 1038, 2000.
4. M. Philippsen, M. Zenger. JavaParty — Transparent Remote Objects in Java *Concurrency: Practice and Experience*, vol. 9, no. 11, pages 1225–1242, 1997.
5. T. Fahringer, A. Jugravu JavaSymphony: A new programming paradigm to control and to synchronize locality, parallelism, and load balancing for parallel and distributed computing. *Concurrency and Computation: Practice and Experience* vol. 17, no. 7-8, pp. 1005 – 1025, 2005.
6. M. Philippsen, B. Haumacher, C. Nester. More Efficient Serialization and RMI for Java. *Concurrency: Practice and Experience*, volume 12, no 7, pages 495–518, 2000.
7. A. Bertozzi, M. Panella, and M. Reggiani. Design of a via based communication protocol for LAM/MPI suite. In K. Klockner, editor, *Procs. 9th Euromicro Workshop on Parallel and Distributed Processing*, pp. 27–33, 2001.
8. L. Schneidenbach, B. Schnor, S. Petri Architecture and Implementation of the Socket Interface on Top of GAMMA Proceedings of the 28th Annual IEEE Conference on Local Computer Network, Bonn/Knigswinter, Germany, October 21-23, 2003.
9. Y. Ishikawa, A. Hori, H. Tezuka, F. O'Carroll, S. Sumimoto, H. Harada, and T. Takahashi. RWC PC cluster ii and Score cluster system software. In R. Buyya, editor, *High performance cluster computing: Architectures and systems*, volume 1, pages 646–660. Prentice Hall, 1999.
10. M. Philippsen, B. Haumacher. More Efficient Object Serialization *IPPS/SPDP Workshops*, pages 718–732, 1999.
11. R. Metkowski, P. Bała, and T. Clark. The performance of different communication mechanisms and algorithms used for parallelization of molecular dynamics code. In R. Wyrzykowski, J. Dongara, M. Paprzycki, and J. Waśniewski, editors, *PPAM01*, Lecture Notes in Computer Science 2328, pages 151–161. Springer-Verlag Berlin, 2001.

Practical Experience in Building an Agent System for Semantics-Based Provision and Selection of Grid Services

Gustaf Nimar[1,⋆], Vladimir Vlassov[2], and Konstantin Popov[3]

[1] Amadeus e-Travel, Sophia Antipolis, France
[2] Royal Institute of Technology (KTH), Stockholm, Sweden
[3] Swedish Institute of Computer Science (SICS), Stockholm, Sweden

Abstract. We present our practical experience in implementing an agent-based system for provision and selection of Grid services. The agents form a marketplace where services are offered and searched. Agents communicate semantic information about services using OWL-S. We describe our implementation that is built on Globus Toolkit 3, the JADE agent framework and an OWL-S toolkit. This combination of technologies can be used for more sophisticated agent-based services, such as automatic composition of services. We illustrate and evaluate our framework using a simple example, yet without loosing generality. Our preliminary evaluation captures the relative costs of different stages during service provision and selection, and detects potential bottlenecks.

1 Introduction and Related Work

The Grid is envisioned as a global ubiquitous infrastructure that allows to treat all kinds of computer-related services as commodity - that can be described, located, purchased or leased, used, shared, etc. Services can be composed together forming "virtual organizations" that deliver non-trivial qualities of service [9, 10]. The Grid is to be become large, decentralized and heterogeneous.

The scale and decentralization of the Grid implies that Grid clients and services can possess only partial information about the Grid. Moreover, access to some information can be restricted due to security. The Grid is also volatile, thus any information accumulated about it is inherently imprecise. The agent-based approach [19, 20, 16] is generally considered to facilitate system development for such environments [13], and its virtue for the Grid is well recognized [8, 6]. An agent in a multi-agent system (MAS) serves a specific role; it is situated in a particular environment; it has complete control over its own state and behaviour; it can communicate with other agents, and its internal and external behaviours are flexible depending on its state and environment. Agent-based software engineering can be used for enhancing the Grid infrastructure itself, providing e.g. knowledge-based information services and semantic service description [6] that will utilize and complement the present day Grid infrastructure [8].

⋆ The work was done when the author was with the KTH, Stockholm, Sweden.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 278–287, 2006.

Grid services, similar to agents in MAS, need flexible communication [8, 6]: its form and content changes following the evolution of the agent's state and environment, as well as the system structure can change over time. Communication using semantic, self-explanatory information addresses the problem [6], as agents can exchange both syntactic data and the knowledge domain in which the data is to be interpreted.

The use of semantic information for agent communication is already being standardized by FIPA [7]. In particular, in the Abstract Service Architecture the agent's "service description" contains semantic information [5]. W3C coined the notion of "semantic web" [1] that addresses the issues of knowledge representation and usage. W3C contributes in particular the Resource Description Framework (RDF), Web Ontology Language (OWL) and Web service ontology (OWL-S). RDF is a data model for entities and relations between them. It provides a simple semantics for this model and a representation schema in XML syntax. OWL extends RDF and can be used to explicitly represent the meaning of entities in vocabularies and the relationships between those entities. OWL-S defines a standard ontology shared by Web services.

In this paper we present our practical experience in integrating the state-of-the-art agent and semantic-web technologies in a Grid service based on the current release of the Globus Toolkit 3 [11]. We believe our results will remain valid for the GT4, as we do not crucially depend on OGSI [18].

We focus our study on use of agents and semantic information for service provision and selection. Clients need to locate suitable services, whereas providers can impose constraints on how their services can be used. Clients are represented by *service selection* agents that work on behalf of clients and guard their interests. Service providers are represented by *service provision* agents that provision services according to providers' interests. Both agents interact to achieve a mutual agreement. Each client or provider should be represented by its own agent because the agent can contain confidential information.

Communication with semantic information provides flexibility for provision and selection agents. First of all, a system for provision and selection of Grid services has to handle arbitrary types of services. Next, there should be an expressive language for specification of client's needs that might be difficult to translate into a rigid communication language. Finally, negotiation may take several steps that can be difficult to express in a rigid communication language.

Our prototype illustrates the issues in building such a system. Our limitations are that we have used a simple query language at the client's end (a WSDL document - the same type of description used at provider's end), and we have implemented a simple matching algorithm. These limitations, however, are encapsulated in particular components in our implementation, and the performance analysis factorizes away the performance properties of those components.

Surveys on MAS [16, 13] show, in particular, how agents can be used for resource brokering, workflow management and planning. Research is active in particular in the fields of negotiation, communication languages, ontologies and scalability issues. In the context of Grid, agents are in particular used for resource

brokering [4] and for scientific computing and workflow management [14, 15, 3]. Following the semantic web, the notion of "Semantic Grid" was coined [8, 6]. Concerning service provision and discovery, work is done on semantic matching - e.g. [2, 17] which however does not consider the use of agents and system design and implementation issues. Ontology-driven resource discovery for Grids is also considered in [12], which adopts the peer-to-peer approach.

2 Protype System Architecture

The system is structured as a marketplace where agents handle both provision and selection of Grid services (see Figure 1). Services to be provisioned are assigned to an SPA in the system. SPAs store service descriptions. An SPA can be assigned a single or several services registered at a VORegistry which is a simple registry service included in Globus Toolkit 3. SPAs advertise themselves at a Directory Facilitator (DF) which is a predefined agent with a directory of SPAs. DFs are used by Service Selection Agents (SSA). A user requesting a service specifies the requirements to an SSA as a WSDL service description document with additional semantic data. These are transformed into a service description in a supported ontology, and the SSA negotiates with the SPA.

In our prototype a service to be provisioned are defined in GWSDL of GT3. The precision of service selection is improved by semantic data such as service properties, capabilities, and taxonomy. This information is stored in GWSDL Service Data Elements (SDEs). A deployed Grid service can be assigned to an SPA which compiles the service definition into an OWL-S ontology object.

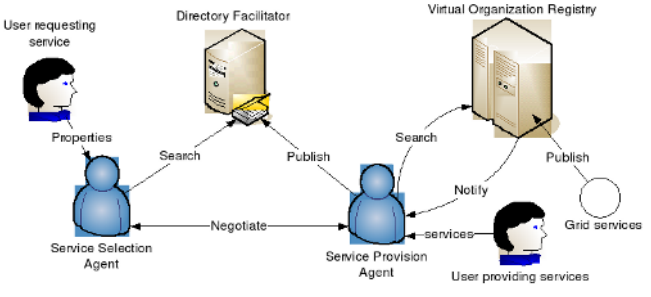


Fig. 1. System Architecture

When a client needs to locate a Grid service using an SSA, it specifies the service requirements. The specification is a GWSDL definition with SDEs that contain further requirements to be matched with properties and capabilities of offered services. The definition is translated into an OWL-S ontology object similarly to service provisioning. The definition does not need to be complete as it is used as "a pattern" during service selection.

A sequence diagram of assigning VO resources to an SPA is shown in Figure 2. First, SPA fetches descriptions of services from the VORegistry. Next, SPA constructs OWL-S descriptions using the WSDL documents and SDEs with semantic information. Finally, the SPA publishes itself at the Directory Facilitator.

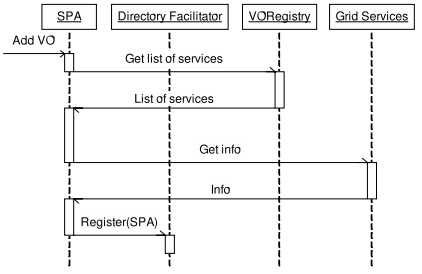


Fig. 2. Assigning a VO to an SPA

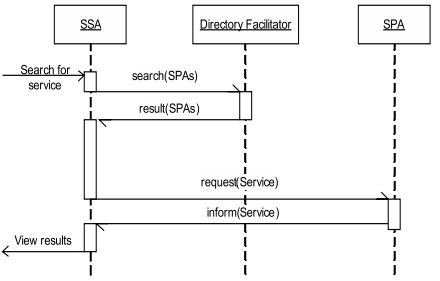


Fig. 3. Selecting a service

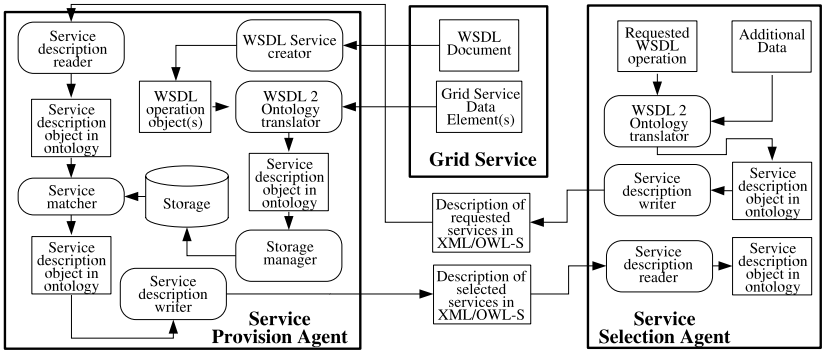


Fig. 4. The information flow in the system

The goal of an SSA is to identify a service best matching the user's requirements. The SSA sequence diagram is shown in Figure 3. When a client initiates a search for a service, the agent first downloads a list of available SPAs from the DF and selects the SPAs to interact with. Then the SSA sends the formalized user's query in FIPA ACL request messages. Each SPA searches its storages and responds with an FIPA ACL inform message. Upon receiving the descriptions from SPAs, the SSA searches the received descriptions for a best match.

A system that provides service selection should support different matching algorithms and different search strategies used by agents. It should encourage the requestors and the providers of services to be detailed in their descriptions of services; include semantic matching of inputs and outputs; and allow for prioritizing the search categories. In the example presented in Section 4 we use a simple algorithm; the study of matching algorithms is out of scope of this paper.

The information flow in the system is shown in Figure 4. GWSDL services descriptions and SDEs containing semantic data are converted by both SPAs and SSAs into OWL-S ontology objects by means of the *WSDL-2-ontology translator*. Internal representation of ontology objects is converted to and from the network representation by means of *service description writer* and *reader*, respectively.

3 Prototype System Implementation

The system prototype has been implemented in Java using the following software platforms and development tools: JADE 3.1 [22], GT 3.2 [23], Java SDK 1.4.2. In addition, we used OWL-S-1.0.1 [21] that is a Java API for reading, executing and writing OWL-S. The OWL-S-1.0.1 API has been revised in order to support the serviceCategory (including taxonomy) located in the ServiceProfile of a service description. The revision affected the reader and the writer of XML documents as well as the representation of ontology objects (in OWL-S 1.0). We also used Jdom 1.0 [24] that is a Java API for reading, manipulation, and writing Java representation of XML. As the system prototype was implemented before Java SDK 1.5 was released, we did not use Web Service support in Java SDK 1.5. When developing experimental Grid services we also used Eclipse 3.0 [25] with the additional Globus Toolkit Plug-in for Eclipse 0.2.0 [26].

The classes and interfaces of the system prototype have been organized in the following five packages: *agents*, *content*, *grid*, *matcher* and *storage*. The *agents* package holds implementations of agents based on the JADE platform. The *content* package includes classes for managing the content carried in the ACL messages together with classes used when working with WSDL documents. The package has been divided into three sub packages: *lang* (classes to work with the content languages included in the ACL messages), *owls* (classes to manage the OWL-S ontology), and *wsdl* (interfaces and classes of the rewritten OWL-S API mentioned above). The *grid* package includes classes that communicate directly with Grid services. The package, in particular, contains methods for extracting Service Data elements given a GSH. The *matcher* package includes classes for matching the requested service description expressed in OWL-S against the ones being advertised in the same ontology. In the matching algorithm described earlier, matching of input and output parameters types is based on the URI of the types. This means that providers and requestors of services can define their own types. The classes responsible for storing descriptions of the services at a Service Provision Agent are grouped in the *storage* package. The system prototype allows the storage to be implemented using various technologies, e.g. databases or classes of the Java collection framework.

Service Provision Agent is implemented by the class ServiceProvisionAgent which extends the JADE Agent class. The SPA includes functionality for assigning services and VOs. The ServiceProvisionAgent also includes functionality for performing service matching. The actions of each agent are based on JADE behaviours. The SPA behaviours are listed in Table 1.

Service Selection Agent is implemented by the class ServiceSelectionAgent. Like the ServiceProvisionAgent, invocation of the selection agent is supported by the platform. The SSA behaviours are listed in Table 2. One interesting method of the ServiceSelectionAgent class is the search method that initializes a search for a service. Every search is identified with a unique identification number. The first behaviour executed in the search is the GetSPAs, which fetches a list of available SPAs. Then a parallel behaviour is invoked, executing one or more SearchSPA behaviours in parallel. Each of the SearchSPA behaviours sends

Table 1. Behaviours of the Service Provision Agent

Behaviour	Description
addService	Assigns a service to an SPA given a GSH. Using the WSDL document representing the service, all non-standard Grid service operations are translated into service descriptions. The descriptions are then handed to the local storage.
addVO	Fetches the GSHs of all the services located in the registry. Each of the services is added to the SPA's storage using the addService behaviour.
ListenForReq	A cyclic behaviour that listens for incoming requests. If a message is received it will be parsed and the right action will be taken.
RegisterSPA	Registers itself at the Directory Facilitator.
SearchAndResponse	Searches the local storage for the requested service and sends the result back.

Table 2. Behaviours of the Service Selection Agent

Behaviour	Description
GetSPAs	Searches the Directory Facilitator for available SPAs.
Receive	A cyclic behaviour that listens for incoming result messages. A received message is parsed and if it contains search results it's stored in a result vector. The behaviour can be terminated by calling the *setDone* method.
SearchSPA	Searches a given SPA for the requested services, i.e. sends an ACL request with a findService Action.
Timeout	Behaviour that sleeps for a while, wakes up after a given timeout, and terminates the collection of search results and starts evaluating the results.

a request to one of the SPAs. Finally a second parallel behaviour is executed including both a receive behaviour (collecting results) and a Timeout behaviour (terminating the search after the given timeout). The Timeout behaviour will also sort the collected results finding the best suited service.

Grid Service Extension. Properties that cannot be expressed in WSDL are defined in Service Data Elements. Some of the OWL-S properties are used in our matching algorithm, e.g. the data fields of the Service Category. In the proposed solution we created an SDE called *OwlsDataType* holding the necessary properties (Fig. 5). The Service Data type can be imported into any Grid service description using the *import* element.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="OwlsData" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://globus.org/master/thesis/service/OwlsService/OwlsSDE"
    xmlns:tns="http://globus.org/master/thesis/service/OwlsService/OwlsSDE">
<wsdl:types>
<schema targetNamespace="http://globus.org/master/thesis/service/OwlsService/OwlsSDE"
    attributeFormDefault="qualified" elementFormDefault="qualified"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <complexType name="OwlsDataType">
        <sequence> <element name="categoryName" type="string"/>
                   <element name="taxonomy" type="string"/>
                   <element name="value" type="string"/>
                   <element name="code" type="string"/>
        </sequence>
    </complexType>
</schema> </wsdl:types> </wsdl:definitions>
```

Fig. 5. The Service Data Type OwlsDataType

4 An Example: Sending Messages to Mobile Phones

Our example considers Grid services sending SMS messages, MMS messages, ring tones, and images to mobile phones. The services are provided by different operators. Grid services, compared to stateless Web services, allow to define additional service data which in our case carries semantic data. Furthermore, it might be convenient to charge a client for all messages sent within some period of time by associating an instance of a service with a client instead of charging each message sent by means of a stateless Web service.

Assume four mobile operators each represented by an SPA. A Grid service offered by an operator is described in WSDL with additional Service Data (based on taxonomy). The service is assigned to the operator's SPA which registers at the Directory Facilitator. The services provisioned by SPAs are listed in Table 3. Assume that a SSA is to find a service sending SMSs that has an operation similar to SMS(msg:string, number:long) → (status: boolean) that costs less than 1.5 SEK. The SSA will transform a WSDL definition of the required service to an OWL-S description object that is sent to SPAs in a FIPA ACL request message.

Table 3. The operator's SPAs and their services

Agent	Operations (WSDL pseudo code)	Service Category
SPA1	**sendSMS**(msg:string, num:long)→(status:bool) **sendMMS**(msg:MMS, num:long)→(status:bool) **sendRingTone**(tone:int, num:long)→(status:bool) **sendPicture**(picture:int, num:long)→(status:bool)	**categoryName:** currency **code:** LESSTHAN **taxonomy:** sek **value:** 1.3
SPA2	**SMSSender**(msg:string, num:int)→void **MMSSender**(msg:MMS, num:int)→void **RingToneSender**(tone:int, num:int)→void **PictureSender**(pic:int, num:int)→void	**categoryName:** currency **code:** LESSTHAN **taxonomy:** sek **value:** 1.4
SPA3	**sendSMS**(msg:string, num:long)→(status:bool) **sendMMS**(msg:MMS, num:long)→(status:bool) **sendRingTone**(tone:int, num:long)→(status:bool) **sendPicture**(picture:int, num:long)→(status:bool)	**categoryName:** currency **code:** LESSTHAN **taxonomy:** sek **value:** 1.6
SPA4	**birthdaySMS**(msg:string, num:long, time:Time)→(status) **birthdayMMS**(msg:MMS, num:long, time:Time)→(status) **birthdayRingTone**(tone:int, num:long, time)→(status) **birthdayPicture**(pic:int, num:long, time:Time)→(status)	**categoryName:** currency **code:** LESSTHAN **taxonomy:** sek **value:** 1.3

```
Service Match(requestedService, providedServices) {
  int highestScore; Service hasHighestScore;
  for all providedService in providedServices do {
    int score = 0;
    score += weightInput  * matchInput(requestedService, providedService);
    score += weightOutput * matchOutput(requestedService, providedService);
    score += weightName   * matchName(requestedService, providedService);
    score += weightTaxonomy * matchTaxonomy(requestedService, providedService);
    if (Score > highestScore) {
       highestScore   = Score;
       hasHighestScore = providedService;
    }
  }
  return hasHighestScore;
}
```

Fig. 6. The Service Matching Algorithm

For this prototype we use a simple matching algorithm shown in Figure 6. Each provisioned service is assigned an integer score – the higher the score the better the service match. The score is based on a weighted addition of the results of four different comparison methods. The weights make it possible for prioritizing between the methods. With the algorithm, the best match for the requested service will be the service offered by the SPA1 for 1.3 SEK.

5 Preliminary Evaluation of the Framework

In this paper, we present results of preliminary evaluation of the ontology-enabled agent-based framework leaving more detailed evaluation for future work. In order to determine potential bottlenecks in the framework we measured the time consumed by different parts of the system prototype. We ran the system on an AMD Athlon 1800+ under Windows XP.

In the first series of experiments, we measured the time consumed by different parts of the system when an SPA executes the service provision behaviour, i.e. creates an OWL-S service description object given the Grid service's WSDL document and additional Service Data. The overall time of provisioning of four services described in the above example was 1209 ms. Our experiments showed that when provisioning service, the part of the system interacting with the Grid container (in our case an Apache server) consumed about 90% of the total time. In particular, measurements showed that the Apache WSDL parser was the most time consuming element. Experiments also showed that fetching service data consumed 4,5% of the total provision time; storing of four services (i.e. merging WSDL Service objects and corresponding Service Data into service descriptions in OWL-S) consumed from 0,2% to 6% per service.

In the second series of experiments, we measured time spent in different parts of the system when an SSA selects services provided by several SPAs. The total wall clock time for selection among four services was 813 ms. The SSA timeout value was set to 500 ms. When the SSA timeouts, it sorts the SPA responses and selects the best one. Measurements showed that the JADE agent platform itself consumed most (about 60%) of the overall selection time. Apart the platform, the conversion of ontology objects to and from XML documents was also time-consuming (18% consumed for writing and 22% for reading XML documents).

We also evaluated the memory usage in the prototype. The results show that a single service description consumes 80% of the total memory usage, when kept in the main memory as a collection object (e.g. Vector). Databases or text files can be used for storing the large amount of service descriptions, whereas object collections can be used for caching of descriptions in the memory.

6 Conclusions and Future Work

We have presented our practical experience in creating an agent-based system for service provision and selection in Grids using semantic Grid service descriptions.

A system is based on using GWSDL and OWL-S languages to define service operations and to provide additional semantic data (properties and capabilities, or constraints and requirements) on Grid services to be provisioned by Service Provision Agents (SPA) and/or to be searched by Service Selection Agents (SSA). Initially, semantic information is provided in service data elements of WSDL service descriptions, which are then translated into OWL-S descriptions. SPAs may use different algorithms to match its service properties and capabilities against requirements and constraints given by an SSA. A system prototype was implemented using available programming platforms and environments: the JADE agent platform, the Globus Toolkit (GT3), the OWL-S-1.0.1 API [11] for working with OWL-S, and the Jdom 1.0 API [24] for working with XML.

Our future work includes porting the system prototype to GT4; stronger evaluation of the prototype; considering different matching algorithms; experimenting with semantic information defined in different ontologies; considering security aspects; providing support for service composition.

Acknowledgments. This work was supported by Vinnova, Swedish Agency for Innovation Systems (GES3 project 2003-00931).

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci.Am., May 2001.
2. Brooke, J., Fellows, D., Garwood, K., Goble,C.A.: Semantic matching of grid resource descriptions. In Proc. of the 2nd Eur. Across Grids Conf., Cyprus, 2004.
3. Cao, J., Jarvis, S.A., Saini, S.: ARMS: An agent-based resource management system for grid computing. Scientific Programming, 10(2):135-148, 2002.
4. Czajkowski, K., Foster, I., Kesselman, C.: Resource and service management. [10].
5. Dale, J., Lyell, M.: Towards an abstract service architecture for multi-agent systems. In Workshop on Challenges in Open Agent Systems'03, 2003.
6. de Roure, D., Jennings, N.R., Shadbolt, N.: The semantic Grid: Past, present and future. Proceedings of the IEEE, 93, 2005.
7. Foundation for Intelligent Physical Agents (FIPA). www.fipa.org.
8. Foster, I., Jennings, N.R., Kesselman, C.: Brain meets brawn: Why grid and agents need each other. In AAMAS'04, New York, USA, July 2004. IEEE.
9. Foster, I., Kesselman, C., Tuecke,S.: The anatomy of the Grid: Enabling scalable virtual organizations. International J. of Supercomputer Applications, 15(3), 2001.
10. Foster, I., Kesselman, C., editors.: The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2nd edition, 2003.
11. The Globus Alliance. http://www.globus.org.
12. Heine, F., Hovestadt, M.: Towards ontology-driven P2P Grid resource discovery. In 5th IEEE/ACM International Workshop on Grid Computing, November 2004.
13. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM, 44(4):35-41, 2001.
14. Kumar, K.: An agent-based Grid flow management framework. GlobusWorld 2004.
15. Moreau, L. et. al.: On the use of agents in bioinformatics Grid. In CCGRID'03: 3rd International Symposium on Cluster Computing and the Grid. IEEE, 2003.
16. Sycara, K.P.: Multiagent systems. AI Magazine, 19(2), summer 1998.

17. Tangmunarunkit, H., Decker, S., Kesselman, C.: Ontology-based resource matching in the Grid - the Grid meets the semantic web. In ISWC'03, pages 706-721, 2003.
18. Tuecke, S. et. al.: Global Grid Forum OGSI Working Group, June 27 2003.
19. Wooldridge, M.: Agent-based software engineering. IEEE Proceedings of Software Engineering, 144:26-37, 1997.
20. Wooldridge, M.: Introduction to MultiAgent Systems. John Wiley & Sons, 2002.
21. Sirin, E.: OWL-S API. http://www.mindswap.org/2004/owl-s/api/
22. Telecom Italia Lab. Jade 3.1. http://jade.tilab.com/
23. The Globus Alliance. Globus Toolkit 3.2. http://www-unix.globus.org/toolkit/
24. The JDOM ^{TM}Project. Jdom 1.0. http://jdom.org/
25. Eclipse Foundation. Eclipse 3.0. http://www.eclipse.org/
26. A Globus Toolkit Plug-in for Eclipse. http://gsbt.sourceforge.net/

A Framework for Managing Large Scale Computing Fabrics and Its Computational Complexity

Piotr Poznański[1,2] and Jacek Kitowski[2,3]

[1] CERN, CH-1211 Geneve 23, Switzerland
[2] Institute of Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, Poland
[3] ACK CYFRONET AGH, ul. Nawojki 11, 30-950 Krakow, Poland

Abstract. In this paper, we discuss the computational complexity of the problem of configuring a computing site. The theoretical considerations are based on the Quattor framework configuration model. We also present some experimental results.

Quattor is a framework for managing large scale computing fabrics. It is a result of developments of the Fabric Management Work Package of the European DataGrid project.

Our experience on using the Quattor framework in the DataGrid testbed and the CERN Computing Centre shows that the ability of proper configuration and management of a computing fabric is essential for having a working grid.

1 Introduction

The Quattor framework [1] presented more thoroughly in [2] provides automated and scalable configuration and installation of very large heterogeneous computing fabrics. It has a modular architecture with a central configuration database and autonomous agents running on fabric nodes. Configuration information is expressed in a high level description language called Pan [3].

With its novel and innovative approach, featuring two main concepts of nodes autonomy and central control over configuration of a fabric, Quattor addresses the requirements of managing large scale grid enabled sites [4, 5] allowing constructing large scale computing fabrics built of thousands of computing nodes. Such fabrics support evolutionary model that allow introduction of new technologies and components while maintaining service.

Other available automatic fabric management tools such as: Cfengine [6], OSCAR [7], Rocks [8] or LCFGng [9] lack required functionality such as scalability or open architecture.

Quattor is a result of developments of the Fabric Management Work Package of the European DataGrid project [10]. It has been used for more than 3 years in production in the CERN [11] Computer Centre to manage more than 2600 machines of multiple functionality, such as batch nodes, disk, tape, database and web

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 288–295, 2006.

servers; and of heterogeneous hardware. Outside of CERN, Quattor is used by the LHC Computing Grid Project ([12]) and many European institutes such as Dapnia, LAL/CNRS, Lyon/IN2P3, CINES, NIKHEF, CCLRC-RAL, UAM etc.

In this paper we present a theoretical estimation of computation complexity of configuring a site with Quattor. It is an important factor since it contributes to scalability characteristics especially when dealing with very large computing fabrics.

2 Configuration Trees and Reference Structures of Pan Templates

In the Quattor model, configuration information is expressed in the Pan language [3, 13] and structured using a construct called the Pan template that groups language statements. A Pan template is a single unit of compilation. Amongst different types of templates featuring in the language, object templates are special. The Pan compiler generates configuration information for every object template it compiles. The compiler can only be invoked on object templates and the other types of templates cannot be compiled directly. A configuration information has a form of a tree of configuration elements described by an XML document. Exactly one configuration tree is associated with exactly one object template, as a result of compilation of this object template. Since, templates may reference other templates by include statement, create function or external path mechanism, the compiler processes a tree of referenced templates. A given object template is a root for such a tree of compilation from which the compilation process begins. Object template references directly or indirectly the nodes of this tree.

Although the external path references an element in a configuration tree, it can be seen as a reference to an object template. A configuration tree can be only obtained by compiling an object template, and there is one to one mapping between object templates and configuration trees.

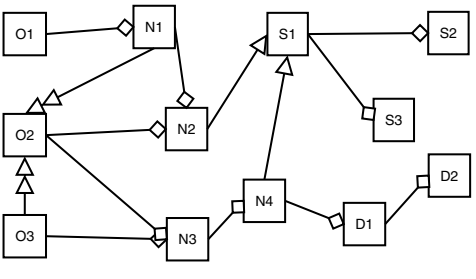


Fig. 1. Templates reference graph. Squares represent templates. Diamond shaped arrows represent the include, single triangle shaped the create and double triangle shaped arrows the external type of dependencies. **O** denotes the object, **N, S, D** the other types templates

External path can be treated by the compiler differently than other dependencies. For every type of dependency different from external path, the compiler processes included or created template, every time appropriate referencing statement is encountered, since the compilation happens in the compilation space of compiled object template. For an external path, the compiler accesses a configuration tree, which is a result of a compilation of an object template in its private space. Therefore, the compiler can compile such an object template every time it is referenced, or can keep resulting configuration tree in its memory after the first compilation. To optimise the process of compilation, the compiler keeps all templates during its compilation session.

Templates create a Directed Acyclic Graph (DAG). Templates are the nodes, and reference relations created by include, create and external path statements are the vertexes of such a graph. An example of a graph is presented in Figure 1. For example for the O3 object template, the compiler will compile templates: O3, N3, N4, S1, S2, S3, D1, D2, O2, N2. Templates S1, S2 and S3 will be compiled twice since they are in trees of both the O2 and O3 object templates.

3 Complexity of Full Compilation

The maximum time of compilation, expressed in templates being compiled, is shown below. This is achieved by finding worse case scenario and estimating the number of compilations expressed in templates. It is assumed that the compiler is invoked on all object templates at once, therefore they are compiled in one compiler run.

To perform the calculation one important and realistic assumption is made. It is assumed that the time to compile non object templates increases linearly with their number [14]. To simplify calculations, it is assumed that for every object template, every non object template is recompiled exactly once. The assumed structure for calculating the complexity is shown in Figure 2. In practice the structure will look more like it is presented in Figure 3.

Thanks to the behavior of the compiler, the references to object templates do not change the number of templates compiled. Object templates are compiled

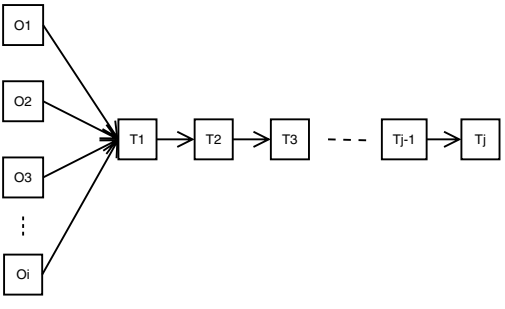


Fig. 2. The structure of templates assumed for complexity calculation. **O** denotes object **T** non-object templates. Arrows represent the dependencies

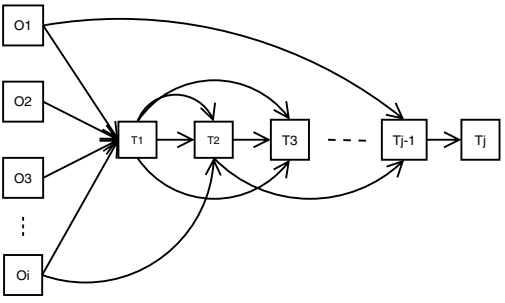


Fig. 3. Structure of templates that happens in reality. **O** denotes object **T** non-object templates. Arrows represent the dependencies

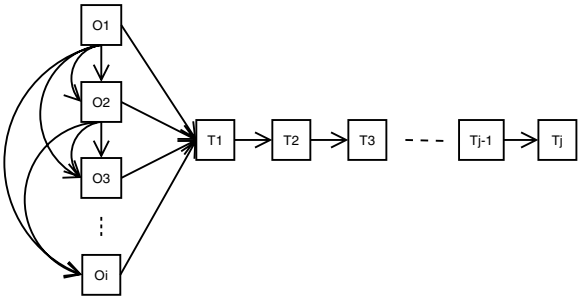


Fig. 4. Structure of templates equivalent in time of compilation to the structure in Figure 2. **O** denotes object **T** non-object templates. Arrows represent the dependencies

only once during a session of the compiler. Therefore the structures shown in Figure 4 are equivalent to the one in Figure 2.

T is a set of all templates and $|T| = n$

N is a set of non-object templates, $N \subset T$ and $|N| = i$

O is a set of object templates, $O \subset T$ and $|O| = j$

$$O \cup N = T$$
$$i + j = n$$

For every object template, the object template and all templates from the set N will be compiled. Time of compilation t will be a function of j - number of object templates and i - number of non-object templates:

$$t(i,j) = j(i+1) \qquad \text{where } i,j \in \mathcal{N} \tag{1}$$

since $i = n - j$

$$t(j) = -j^2 + nj + j \quad \text{and} \quad \frac{dt(j)}{dj} = -2j + n + 1$$

to find an extrema of the function $t(j)$, the equation is to be solved:

$$\frac{dt(j)}{dj} = 0$$

it gives:

$$j = \frac{n+1}{2}$$

This is a maximum of the function, since $\frac{dt(j)}{dj}$ is monotonically decreasing because:

$$\frac{d^2t(j)}{dj^2} = -2 < 0$$

For $n = 2k + 1$ where $k \in \mathcal{N}$
and then:

$$t_{max2k+1} = t(\frac{n+1}{2}) = \frac{(n+1)^2}{4}$$

For $n = 2k$ where $k \in \mathcal{N}$ the maximum of the discrete function $t(j)$ is in both points:

$$j_1 = \frac{n}{2} \quad \text{and} \quad j_2 = \frac{n+2}{2}$$

since for both:

$$t_{max2k} = \frac{n(n+2)}{4}$$

Overall complexity of compilation of a set of n templates is:

$$Complexity(n) = O(t_{max2k}) = O(t_{max2k+1}) = O(n^2)$$

In case of a stable situation, when the global configuration of a computing site is constant and only configuration of computing nodes changes, therefore the set of non-object templates is invariable, the time of compilation depends on the number of object templates that are compiled. According to Eq.(1), the time is linear to the number of object templates compiled, with a constant factor corresponding to the number of non-object templates.

In practice, a set of non-object templates is much smaller than a set of object templates. Therefore compilation time of a set of templates is linear to the number of object templates and much shorter than in worst case scenario. For example, for a set of 5000 templates describing configuration of 4800 machines, therefore containing 4800 object templates, there will be 964800 compilation. Assuming worst case scenario, a set of 5000 templates containing 2500 object templates would require 12505000 compilation. An order of magnitude is the difference.

4 Experimental Results

Experiments were performed on a set of 8799 templates coming from the Configuration Database (CDB) [15] of the CERN Computer Centre. The set contained 3203 object templates. Every object template was accompanied by a template with network information. Therefore, there was a set of 2393 shared templates. The set

of shared templates is rather constant. During experiments, number of compiled object templates varied and the set of shared templates was kept constant.

The templates were compiled on a Pentium-M 1.5 GHz machine with 756 Mbytes of RAM running Fedora Core 3 Linux. Compilation was made with the Pan 'panc' compiler version 1.0.9, and the CDB 'cake' dependency machine version 1.1 [15]. The compilation times were measured using the 'time' standard Unix utility.

First, a random set of 100 object templates was compiled 1000 times. A compilation set was randomly chosen for every compilation. The histogram of the durations of the compilation is presented in Figure 5. The distribution of the results is similar to the normal distribution. In this case the obtained results

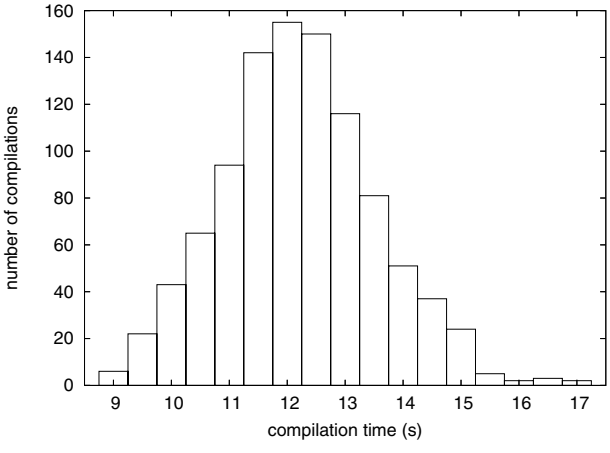


Fig. 5. Histogram of the durations of the compilation of 100 templates compiled 1000 times

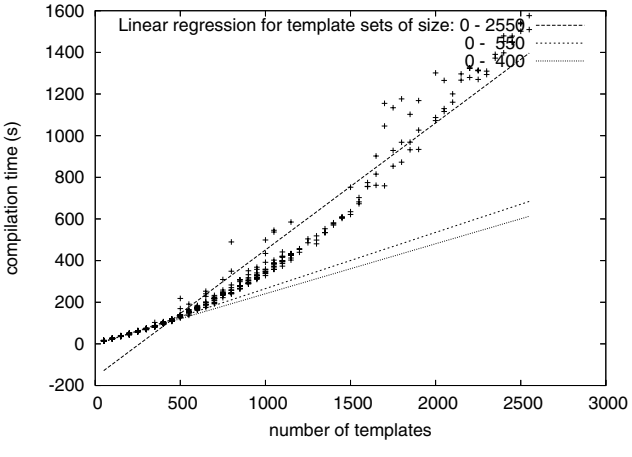


Fig. 6. The compilation durations of the sets of 0 - 2550 object templates

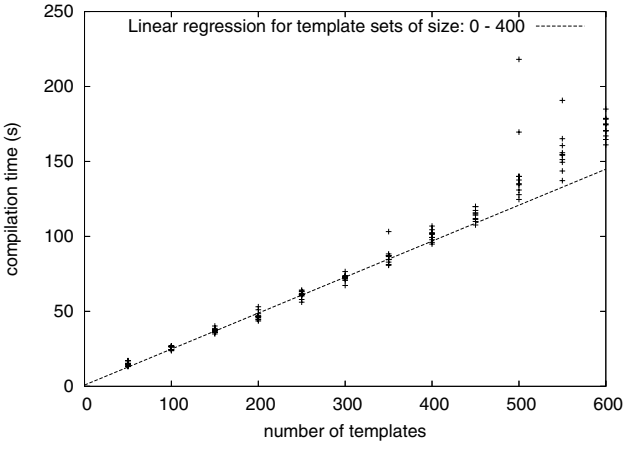


Fig. 7. The compilation durations of sets of 0 - 600 object templates

have the statistical parameters equal to: $\bar{t} = 12.47$[s] and $s_t = 1.45$[s]. This shows that the time of compilation of object templates is statistically equal.

To verify that the duration of compilation of a set of object templates is linear to the number of these templates, the following experiment was performed. Sets of object templates of the size starting with 50 up to 2550 with a step of 50 templates were compiled. The sets of the sizes 50 - 1050 were compiled 10 times and bigger sets 3 times.

The results for the sets of size 0 - 2550 templates are presented in Figure 6 and 0 - 600 in Figure 7.

It can be observed that up to around 400 object templates, the compilation duration is linear to the quantity of compiled templates, that confirms linear complexity obtained theoretically (cf. Eq.(1) and the following discussion). This is caused by the fact that the compiler allocates all the machine physical memory, and operating system starts swapping. On machines with more memory, the compilation stays linear for bigger sets of templates. Optimising the compiler to occupy less memory is another possibility of achieving linear time for big compilation sets.

5 Conclusions

The obtained theoretical results show that computational complexity of compiling configuration is quadratic to a number of all templates. Furthermore, the results show that it is linear to a number of object templates. The results has been confirmed experimentally.

The theoretical and practical considerations of computational complexity of configuration demonstrate that this aspect of managing computing fabrics is scalable.

Acknowledgments

The authors of this article would like to thank all colleagues of the Work Package 4 of the DataGrid project, working on the Quattor framework. They also would like to thank the CERN Computer Centre managers, especially German Cancio, the Quattor project coordinator for supplying real production data to perform the experiments. AGH-UST grant is also acknowledged.

References

1. Quattor, http://quattor.org
2. Poznański P. et al: "Quattor - a Framework for Managing Grid-enabled Large Scale Computing Fabrics", *Proc. of the Cracow Grid Workshop*, Cracow, Poland, Dec.12-15, ACK Cyfronet-AGH, Cracow, 2004, in press.
3. Cons L., Poznański P.: "Pan: A High-Level Configuration Language", *USENIX, LISA Conf. Proc.*, Philadelphia, Nov.3-8, 2002, USENIX, pp.83-98.
4. "WP4 Report on Current Technology", 2001.
5. "WP4 Final Evaluation Report", 2004.
6. Cfengine, http://www.cfengine.org
7. OSCAR, http://oscar.openclustergroup.org/
8. NPACI Rocks, http://www.rockscluster.org
9. LCFGng, http://www.lcfg.org
10. European Union DataGrid Project (EDG), http://www.eu-datagrid.org
11. European Organization for Nuclear Research (CERN), http://www.cern.ch
12. LHC Computing Grid, http://lcg.web.cern.ch/LCG/
13. Cons L., Poznański P.: "High Level Configuration Description Language Specification", http://cern.ch/hep-proj-grid-fabric-config
14. Poznański P.: "Framework for Managing Grid-enabled Large Scale Computing Fabrics", PhD Thesis, AGH-UST, Cracow, 2005.
15. Cons L. and Poznański P.: "Configuration Database Global Design", http://cern.ch/hep-proj-grid-fabric-config/documents/cdb-design.pdf

Domus – An Architecture for Cluster-Oriented Distributed Hash Tables

José Rufino[1,⋆], António Pina[2], Albano Alves[1], and José Exposto[1]

[1] Polytechnic Institute of Bragança, 5300-854 Bragança, Portugal
{rufino, albano, exp}@ipb.pt
[2] University of Minho, 4710-057 Braga, Portugal
pina@di.uminho.pt

Abstract. This paper presents a high level description of Domus, an architecture for cluster-oriented Distributed Hash Tables.

As a data management layer, Domus supports the concurrent execution of multiple and heterogeneous DHTs, that may be simultaneously accessed by different distributed/parallel client applications. At system level, a load balancement mechanism allows for the (re)distribution of each DHT over cluster nodes, based on the monitoring of their resources, including CPUs, memory, storage and network. Two basic units of balancement are supported: *vnodes*, a coarse-grain unit, and *partitions*, a fine-grain unit. The design also takes advantage of the strict separation of object *lookup* and *storage*, at each cluster node, and for each DHT. Lookup follows a distributed strategy that benefits from the joint analysis of multiple partition-specific routing information, to shorten routing paths. Storage is accomplished through different kinds of data repositories, according to the specificity and requirements of each DHT.

1 Introduction

Certain classes of applications require large *dictionaries*, which are data repositories that store <key, data> records, accessed by its unique <key>.

Distributed Hash Tables (DHTs) are one of the most popular approaches to distributed dictionaries. Research in this domain has been prolific, ranging from 1st generation models, oriented to the cluster environment [1, 2, 3, 4], to numerous recent contributions, most exclusively focused in the area of P2P systems [5, 6].

Despite its abundance, most DHT designs have concentrated on the issues related to a single DHT deployment. Applications, however, may require the simultaneous availability of several DHTs, whether exclusively accessed by a single application, or shared by multiple applications.

Moreover, depending on certain basic attributes (*e.g.*, the type of hash function, the persistence level required for data records, etc.), DHTs may exhibit different properties that are suited to different application scenarios. The lack

⋆ Supported by the portuguese grant PRODEP III - 5.3/N/199.006/00.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 296–303, 2006.

of support for the above desired features, both at the design and deployment levels, lengthens the application development cycle and prevents the resources of the distributed/parallel environment to be more efficiently exploited.

In this paper we present a general description of Domus, an architecture aimed to support the creation, usage and management of *multiple DHTs*, to be *simultaneously* deployed at a cluster environment.

The design is compatible with the possibility to specify, for each DHT, the value of a basic set of attributes, thus supporting *heterogeneity of DHTs*.

At the execution environment, heterogeneity is also conveniently exploited through a load balancement mechanism that builds on the dissociation of the *addressing/lookup* and *storage* functions of the data records, once these functions may impose different requisites on the resources of the cluster nodes.

The discovery of a data record is based on a *distributed lookup* approach, that benefits from the combined analysis of multiple sources of routing information, available at each cluster node, to shorten routing paths.

The remaining of the paper is organized as follows: section 2 revisits previous work on specific issues of the architecture, section 3 presents a general view, sections 4 and 5 elaborate on the main components, and section 6 concludes.

2 Previous Work

Domus architecture gives continuity to our previous work.

In [7, 8] we have presented and evaluated a model for the balancement of the range (*address space*) R_h of an hash function h, over a set of heterogeneous cluster nodes. The model provides for the definition of two basic units of balancement: a) the *vnode*, a coarse-grain unit, and b) the *partition*, a fine-grain unit.

Whenever a cluster node n requires $\#V.n$ vnodes of a DHT, for a global number of $\#V$ vnodes then, ideally, n should be responsible for the fraction $Q^i.n = \#V.n/\#V$ of R_h, which defines the node's *ideal quota*. However, in reality, nodes are given a certain number of *partitions* of R_h, which may be viewed as slices of R_h, all of the same size, and mutually exclusive. Thus, if a node n is bound to $\#P.n$ partitions, for a global number of $\#P$ partitions, the node's *real quota* of R_h is given by $Q^r.n = \#P.n/\#P$.

Vnodes and partitions relate as follows: for each one of its vnodes, a node will be given a certain number of partitions; this number is dynamically adjustable so that, for every cluster node n enrolled in the DHT, $Q^i.n$ and $Q^r.n$ are maintained as close as possible, at every moment, even when the number of nodes that support the DHT and/or their ideal quotas change.

Depending on the number of nodes that support a DHT, and on the maximum deviation allowed between real and ideal quotas, the overall number of partitions bound to each node may be considerable, preventing any cluster node to maintain the full <partition, node> assignment/addressing table.

A well known solution to the previous problem is to interconnect the partitions of the DHT by an *application level topology* that supports a distributed lookup mechanism. Under such mechanism: a) each node needs to keep only a

small (logarithmic bound) number of <partition, node> associations, deterministically defined and b) looking up for any partition involves the participation of a limited (logarithmic bound) number of nodes.

However, in the context of our work, the direct application of this solution to our models is not recommended. Because a cluster node may be responsible for multiple partitions of the same DHT, it may be visited many times if a *routing chain* is exclusively based on the *conventional* distributed lookup algorithms, where lookup requests "hop between partitions".

We have thus studied algorithms for *aggregated routing* [9], suitable to *Chord* [6] and *de Bruijn* [10] graphs, which fit naturally in our models for the weighted distribution of the hash function range. By using *aggregated routing*, the combined routing information of various partitions is investigated to ensure that lookup requests "hop between cluster nodes", leading to shorter routing chains, when compared to conventional routing.

3 General View

A typical Domus deployment, as shown in figure 1, builds on four major components: i) *client applications* (a_i); ii) *DHTs* (d_j); iii) *services* (s_k); iv) *cluster nodes* (n_l). External applications and services, not represented, may also coexist. A Domus deployment or instantiations is named hereafter as *Domus cluster.*

In the same figure, *base services* allow for: a) the interaction between applications and/or services, through high performance message passing (as provided by RoCL [11]), b) the global monitoring of resources (as provided by Ganglia [12]) and c) the remote execution of services (*e.g.*, via RoCL, Ganglia or `rexec`).

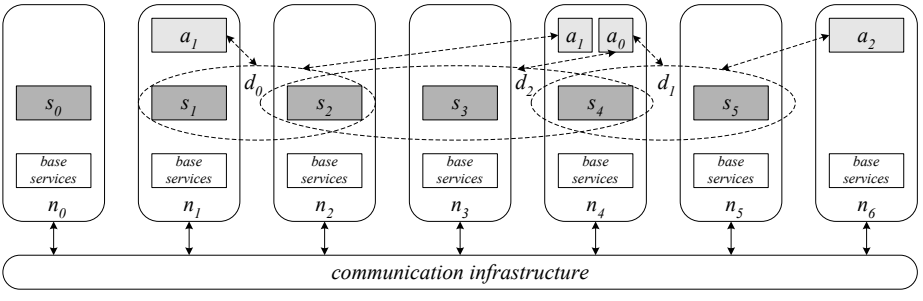


Fig. 1. A typical Domus cluster

Figure 1 also illustrates some basic architectural possibilities and constraints:

- one client application may access multiple DHTs;
- one DHT may be accessed by multiple client applications;
- one DHT may be implemented by multiple Domus services;
- one Domus service may support multiple DHTs;
- one cluster node runs no more than one Domus service, per Domus cluster.

Client applications interact with Domus services to explore DHT abstractions or for administration purposes. Interactions are conducted through the facilities offered by a user-level library which, among others, includes methods to: a) create/destroy, open/close and shutdown/restart a Domus cluster; b) add/remove Domus services; c) create/destroy, open/close and shutdown/restart DHTs; d) insert, modify, read, browse or remove data records from DHTs.

Domus also supports the user-level definition of a basic set of attributes for each DHT: a) the hash function; b) maximum deviation between ideal and real quotas of the DHT, for any node; c) initial distribution of the DHT; d) redistribution constraints; e) balancement thresholds; f) routing overlay and algorithms; g) storage media and block size; h) data access pattern (read-only/read-write).

4 Domus Services

Domus services are the fundamental building blocks of the Domus architecture: it's their cooperation that enables to deploy, access and manage multiple DHTs.

At each node, Domus services comprise an *addressing subsystem* (AS), a *storage subsystem* (SS) and a *balancement subsystem* (BS).

The AS and SS subsystems support the *address space* and *storage space*, respectively, of one or more vnodes, eventually from different DHTs. The BS subsystem monitors the utilization of certain node resources and, if necessary, triggers the migration of local vnodes to another Domus services.

The address/storage space of a vnode is the union/sum of the address/storage space of its partitions. The address space of a partition is its own slice of the hash function range. The storage space of a partition refers to the storage resources consumed to hold the data records whose keys map onto its address space.

The separate management of the address/storage space of vnodes and partitions allows increased flexibility: such spaces may be independently managed, by different services[1], which has proved to be useful for balancement purposes.

The possible enrollment of a service in more than one DHT, and at several levels, is illustrated by figure 2. The figure shows how the AS and SS subsystems of services $s_1, ..., s_5$ support the DHTs $d_0, ..., d_2$, for the scenario of figure 1.

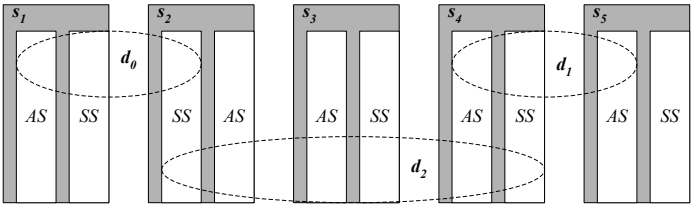


Fig. 2. A possible association of the AS/SS subsystems of $s_1, ..., s_5$ to $d_0, ..., d_2$

[1] Named *addressing/storage services* of the vnodes or partitions (the services whose AS/SS subsystem manage the address/storage space of the vnodes or partitions).

4.1 Addressing Subsystem

Lookup of Partitions. For a large number of partitions, a distributed approach to partition lookup is advisable. In our case, that approach must also be compatible with the decoupling of the addressing and storage functions of partitions; such implies that the lookup of a partition may involve two distinct lookups: i) the finding of its addressing service and ii) the finding of its storage service.

Domus supports the interconnection of the overall partition set of any DHT by a routing overlay, such as a *Chord* or *de Bruijn* graph, thus requiring the definition of a *routing table* (RT) per partition, residing at the partition's addressing service (in the AS subsystem). Each RT maintains *addressing references* (AR) to the addressing services of other partitions of the same DHT. Along with its RT, a *storage reference* (SR) to the storage service of the partition is also preserved.

Thus, finding the storage service of a partition starts by locating its addressing service, in order to obtain its SR reference. Relying in addressing services for the maintenance of SR references ensures that finding a storage service benefits from the same distributed lookup mechanisms used to find an addressing service.

Aggregated Routing. A Domus service may be the addressing service for several partitions of the same DHT. In such case, there will be many RTs locally available for that DHT. The combination of the totality or part of the RTs of a DHT, available at each service, allows routing decisions to be performed under enhanced algorithms (*aggregated routing*), that usually lead to shorter routing chains than *conventional routing*, which bases routing decisions in a single RT.

To make aggregated routing faster, all the addressing information available at each Domus service is brought into *addressing indexes* (AIs), one per each DHT. For a partition uniquely identified by its PID, the AI index maintains the RT table of the partition, its SR reference and an *access counter* (AC).

Figure 3 shows the AI index maintained by services s_3 and s_4, at their AS subsystems, for DHTs d_2 and d_1, accordingly to the roles defined in figure 2 for those services. SR references are explicitly represented, by dashed arrows.

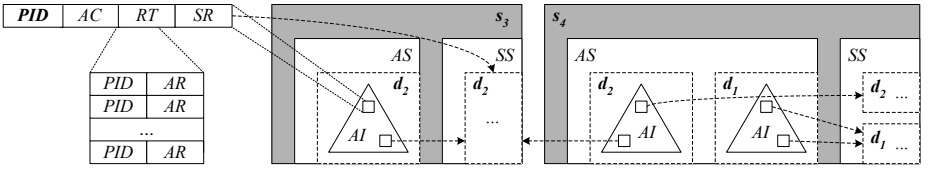


Fig. 3. A representation of the AI indexes of s_3 and s_4

Load Balancing. The AS subsystem measures the access rate to the local RTs of a DHT and, if this statistic surpasses a certain threshold on the average access rate per vnode, it will trigger the creation of a new vnode. The new vnode will "steal" some partitions from the others, thus the average number of partitions per vnode decreases which, in turn, lowers the average routing load per vnode.

The AS subsystem may also switch routing algorithms, if needed. Routing algorithms supported in Domus, for *Chord* and *de Bruijn* graphs, range from conventional routing methods, to more complex algorithms for aggregated routing; the first class of algorithms is lighter, but routing chains are longer; the second class is heavier, but routing chains are smaller. Because all these algorithms converge to the same final answer of a lookup request (though some converge faster than others), different algorithms may be used along the same routing chain.

4.2 Storage Subsystem

Repositories. The data records whose keys map onto the address space of a partition are saved in a per DHT *repository*, at the storage service of the partition. In this context, a repository is any data store capable of holding *dictionaries*.

The SS subsystem supports any kind of repository (and multiple instances), if provided proper *midleware* to interact with it. Each kind has specific properties, relevant to different application scenarios. Any Domus deployment should support at least a RAM-based (volatile) and a disk-based (persistent) repository.

The local repository of a DHT is identified by a *repository reference* (RR) – see figure 4. This is used to access the repository trough a *repository abstraction layer* (RAL) that presents a generic/unified interface to repositories, irregardless of their base storage platform. The RAL layer maps a high-level RR reference to a low-level handler, used to access the repository via appropriate middleware.

A specific *storage index* (SI) per DHT is also preserved. For each partition locally stored, it contains its PID, the *addressing reference* (AR) to its addressing service and a *storage counter* (SC). Conveniently, if the AS subsystem is allowed to inspect the SI indexes of the SS subsystem, a routing chain may end sooner.

Figure 4 magnifies the SS subsystem of s_4, already visible in figure 3; it shows the SI index for d_2 and d_1, including some AR back-references (also to s_4).

Load Balancing. Domus provides several mechanisms to balance the consumption of storage resources. Firstly, during the creation of a DHT, a correspondence between a vnode and a certain amount of storage is established; thus, if vnodes

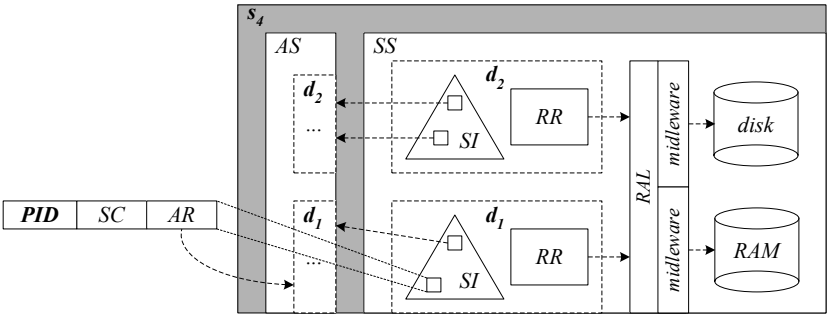


Fig. 4. A representation of the inner structure of the SS subsystem of s_4

become "full", some options are: a) to delete data records (the DHT may behave like a cache), or b) to create more vnodes for the data records in excess. Secondly, Domus may enforce fragmentation of user-level data records into DHT-level data blocks; these are uniformly spread over the partition set of the DHT, which has a twofold effect: it ensures that both the storage consumption and the access load are also uniformly spread over the Domus services that support the DHT.

4.3 Balancement Subsystem

The BS subsystem monitors the load on local resources (like CPU, main memory, disk and network), with the help of specialized tasks, provided by the base services (*e.g.*, Ganglia [12] and Domus-specific plugins). Such tasks collect raw information about the load/availability of node resources and maintain utilization statistics (*e.g.*, exponential moving averages). If the utilization of a resource surpasses a certain threshold, the BS subsystem triggers a *balancement event*.

The processing of a balancement event involves the selection of the set of RTs or data records, of a certain local vnode, that should be moved to another Domus service, in order to diminish the pressure on the overloaded resources. The selection is based on the analysis of the access rate and storage utilization statistics, maintained by the AS and SS subsystems, for RTs and data records. A *supervisor* service choses the service where RTs or data records should be moved.

5 Supervisor Service

Some tasks in a Domus cluster may require global coordination, such as: a) the creation/destruction, shutdown/restart of the Domus cluster; b) the adding/removal, shutdown/restart of specific Domus services; c) the creation/destruction, shutdown/restart and redistribution of specific DHTs. These tasks are conducted under the supervision of a well known Supervisor service).

The expansion of the set of Domus services, of a Domus cluster, may be *administrative*, as required by an administrative application, or *automatic*, as the result of the processing of balancement events. Contraction is administrative, typically motivated by the need to detach some cluster nodes from a Domus deployment; in such scenario, the addressing and/or storage responsibilities of the affected services must be transfered to other services, in other cluster nodes.

Shutting down a DHT requires saving all its RTs and data records in persistent repositories, at the cluster nodes that host the Domus services enrolled in the DHT. Afterwards, the DHT will enter an *off-line* state. The restart of a DHT brings it back to the *on-line* state. Shutting down a Domus service involves the local shutdown of its supported DHTs. Finally, shutting down an entire Domus cluster involves the shutdown of the Domus services and of the Supervisor.

Balancement events are serialized by the supervisor and comprise the following two phases: 1) the discovery (or instantiation) of Domus services with the necessary available resources, and 2) the coordination of the transfer of RTs or data records, from the overloaded service to the selected recipients.

6 Conclusions

The main contribution of this paper is the definition of an architecture for cluster-oriented DHTs, aimed to support multiple, heterogeneous and balanced DHTs.

The design allows the user-level specification, separately for each DHT, of a basic set of attributes, providing for heterogeneous DHTs.

It also supports the decoupling of the routing and storage functions of DHTs, as a strategy to achieve global load and storage balancement, in the highly dynamic execution environment of the cluster.

To our knowledge, Domus is novel in the combined support for all these features, and also in the distributed lookup enhancements that allow to accommodate aggregated routing algorithms.

A prototype version of Domus is being developed in the context of SIRe[2], as a simple implementation of the work described here.

References

1. Litwin, W., Neimat, M.A., Schneider, D.: LH*: Linear Hashing for Distributed Files. In: Procs. of ACM SIGMOD - Int. Conf. on Management of Data. (1993)
2. Devine, R.: Design and implementation of DDH: a distributed dynamic hashing algorithm. In: Proceedings of the 4th Int. Conference on Foundations of Data Organization and Algorithms. (1993)
3. Hilford, V., Bastani, F., Cukic, B.: EH* – Extendible Hashing in a Distributed Environment. In: Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference. (1997)
4. Gribble, S., Brewer, E., Hellerstein, J., Culler, D.: Scalable, Distributed Data Structures for Internet Service Construction. In: Proceedings of the Fourth Symposium on Operating Systems Design and Implementation. (2000)
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content–Addressable Network. In: Proceedings of the ACM SIGCOMM'01. (2001)
6. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balkrishnan, H.: Chord: A Scalable P2P Lookup Service for Internet Applications. In: Proceedings of ACM SIGCOMM'01. (2001)
7. Rufino, J., Pina, A., Alves, A., Exposto, J.: Toward a dynamically balanced cluster oriented DHT. In: Proceedings of the International Conference on Parallel and Distributed Computing and Networks (PDCN'04). (2004)
8. Rufino, J., Alves, A., Pina, A., Exposto, J.: A cluster oriented model for dynamically balanced DHTs. In: Proceedings of IPDPS '04. (2004)
9. Rufino, J., Pina, A., Alves, A., Exposto, J.: Agregated routing for a cluster oriented DHT. Technical report, Dep. of Informatics and Communications, Polytechnic Institute of Bragança, Portugal (2004)
10. Bermond, J.C., Liu, Z., Syska, M.: Mean Eccentricities of de Bruijn Networks. Technical report, Université de Nice-Sophia Antipolis (1993)
11. Alves, A., Pina, A., Rufino, J., Exposto, J.: RoCL: A Resource oriented Communication Library. In: Proceedings of Euro-Par 2003. (2003)
12. D. Sacerdoti, F., Katz, M.J., Massie, M.L., Culler, D.E.: Wide Area Cluster Monitoring with Ganglia. In: Proceedings of the IEEE Cluster 2003 Conference. (2003)

[2] Scalable Information Retrieval environment (grant FCT POSI/CHS/41739/2001).

Iterative Reconstruction of Tomographic Scans in Dynamic SMP Clusters with Communication on the Fly

Boguslaw Butrylo[1], Marek Tudruj[2,3], and Lukasz Masko[2]

[1] Białystok Technical University, Faculty of Electrical Engineering,
45D Wiejska str, 15–351 Bialystok, Poland
[2] Institute of Computer Science of the Polish Academy of Sciences,
01–237 Warsaw, ul. Ordona 21, Poland
[3] Polish–Japanese Institute of Information Technology,
02–008 Warsaw, ul. Koszykowa 86, Poland
butrylo@we.pb.bialystok.pl, {tudruj, masko}@ipipan.waw.pl

Abstract. This paper deals with iterative algorithm of reconstruction of 2D tomographic scans. The relative speedup of the reconstruction process as well as real–time and high fidelity imaging can be improved by implementation of distributed processing. The properties of the sequential algorithm are studied, and the scheme of coloring of projections is introduced in the concurrent version of the presented algorithm. The details of the elaborated distributed implementation of the ART (Algebraic Reconstruction Technique) are presented. Parallel implementation in a system based on dynamically configurable SMP clusters is proposed followed by performance analysis.

1 Introduction

Modern computer technology has a great impact on properties and development of some methods of medical non–invasive diagnosis. The medical imaging techniques, including different kinds of computer tomography (CT), have evolved to more accurate and real–time algorithms [1, 2]. Accuracy, fidelity, real-time processing, and efficiency of the reconstruction remain the most important factors in the wide application of the methods. Parallel implementation of the reconstruction algorithms should reduce the processing time and improve the efficiency of the CT algorithms.

The structure and properties of the iterative Algebraic Reconstruction Technique (ART) have been presented in [1]. The sequential structure of the method is analysed and evaluated with the graph theory. The elaborated algorithm is implemented on the distributed memory message passing multicomputer platform. Some benchmark two–dimensional problems are used to estimate performance and weight coefficients of the operations in the distributed environment. The massive data transfers between processing units decrease the performance of the method. A shared memory system seems to be more suitable to efficient application of the presented algorithm. The distributed scheme of the ART method is

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 304–311, 2006.

implemented in a system of dynamic shared memory processor (SMP) clusters with communication on the fly [5,6].

2 Problem Formulation

Since some different deterministic and parasitic physical phenomena shape the final output form of the Radon transforms $\mathbf{p} + \mathbf{p}_e$, the reconstructed two–dimensional image $\boldsymbol{\mu}$ is an approximate solution of the complex matrix equation

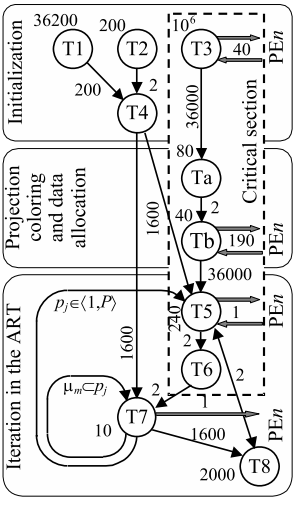
$$\mathbf{C} \cdot (\boldsymbol{\mu} + \mathbf{e}) = \mathbf{p} + \mathbf{p}_e \tag{1}$$

The resolution of the final image is determined by the medical requirements, and in general case $\dim(\boldsymbol{\mu}) = M_x \times M_y$ pixels. The precision and fidelity of the final results depend on the number of projections, $\dim(\mathbf{p}) = P$, as well as the relative values of noise signals and some disturbances during the time of exposure, $\dim(\mathbf{p}_e) = P$, $p_{e,i} =$ random. The $\mathbf{C}$ matrix is sparse, real and positive definite, $\dim(\mathbf{C}) = P \times (M_x \times M_y)$. The components of the matrix describe the contribution of the c_{ij} pixel in the i–th projection $\mathbf{p}_i$. The outcome of CT reconstruction is deformed by error $\mathbf{e}$ of the implemented either iterative or transform–based algorithm of image processing. The general, threshold level of this error $\max(\mathbf{e})$ can be estimated, but its local value in the created image is random. Among other things, it depends on the complexity and shape of the scanned object.

The coherent and effective formulation of the CT reconstruction algorithm should be based on the coupled and complex analysis of physical constrains of the measuring process, and properties of some selected mathematical methods. A high fidelity and real–time reconstruction of the CT image is a computationally and memory demanding task. Nowadays, the distributed implementation of the reconstruction algorithm is a promising and cost–effective way to increase the quality and performance of the CT technique. The correct, software and hardware driven formulation of the distributed algorithm enables to overcome some limitations and to reduce some errors of the reconstruction process.

In the first step the concurrent version of the elaborated algorithm was implemented and validated in a multicomputer, message passing environment. The distributed version of the ART algorithm was based on the MIMD (multiple instruction, multiple–data) and typical domain decomposition paradigms. Since $\mathbf{C}$ matrix is sparse, its full representation in the computer memory was squeezed with the CRS (Compressed Row Storage) algorithm [3]. This matrix remains the largest data structure in the algorithm, therefore it was homogeneously decomposed between processing units PEn. The vector of projections $\mathbf{p} + \mathbf{p}_e$ and vector of the image $\boldsymbol{\mu}$ were duplicated in the computing nodes.

The size of $\mathbf{C}$ matrix makes data transfers between computing units non–efficient or even impossible. The spatial decomposition of $\mathbf{C}$ matrix on either distributed or shared memory environment is the general constraint of the reconstruction algorithm. The critical section of the algorithm consists of tasks, where one of the operands is a part of $\mathbf{C}$ matrix (Fig. 1).



T1 Calculate the sum of projections.
T2 Set the number of projections.
T3 Assemble the C matrix.
T4 Create the introductory form of the image $\boldsymbol{\mu}^{(0)}$
Ta Decompose the vector of projections $\mathbf{p}$ by colors (create colors).
Tb Transfer components of the $\mathbf{C}$ matrix between processing units to obtain mutually–dependent subsets.
T5 Calculate approximate value of the j–th projection in the i–th iteration.
T6 Calculate value of the correction coefficient δ_j.
T7 Modify the vector of the image $\boldsymbol{\mu}^{(i)}$.
T8 Check the convergence of the iterative algorithm.

Fig. 1. Graph of the elaborated iterative CT reconstruction algorithm in the distributed message–passing environment. Weight coefficients of threads (nodes on the graph) and relations between threads (thin arrows) are estimated for low resolution image $\dim(\boldsymbol{\mu}) = 40 \times 40$ pixels. Wide, horizontal arrows indicate data transfers between processing units.

The general form of the leap–frog ART algorithm is shaped by the iterative processing of projection vector $\mathbf{p}$. Any new, approximate value of the projection $p_j^{(i)}$ must be taken into account in the next step of image reconstruction $\boldsymbol{\mu}^{(i+1)}$. Certainly, the modified form of the image $\boldsymbol{\mu}^{(i+1)}$ should be used immediately in the calculation of the next projection $p_j^{(i)}$

$$\forall_{j \in <1,P>} \quad \delta_j^{(i)} = \frac{1}{N_j} \left(p_j - \sum_{m=1}^{N_j} c_{jm} \mu_m^{(i)} \right) \tag{2}$$

$$\forall_{j \in <1,P>} \quad \forall_{\mu_m \subset p_j} \quad \mu_m^{(i+1)} = \max \left(\mu_m^{(i)} + \delta_i^j ; 0 \right) \tag{3}$$

where N_j is equal to the number of pixels connected with the j–th projection. These task and data dependencies are not suitable to simple implementation in the multi–computer/multi–processor platform. Therefore the relations between the coupled data are modified in the presented approach. The performance of the ART algorithm is improved, when the set of projections p is divided into mutually–independent sub–sets of projections (i.e. colors of projections)

$$\mathbf{p} = [p_1, \ldots, p_j, \ldots, p_P]^T = \left[\mathbf{p}_1^T, \ldots, \mathbf{p}_j^T, \ldots, \mathbf{p}_R^T \right]^T \tag{4}$$

where r is the index of the sub–set (i.e. color), $r = 1, \ldots, R$. The r–th sub–set consists of the independent projections, where the dot product of any two projections remains equal to zero

$$\forall_{p_i,p_j \in \mathbf{p}_r} \quad \sum_{m=1}^{M_x \times M_y} (c_{i,m} \cdot c_{j,m}) = 0 \tag{5}$$

The coloring task is made once, before the iterative reconstruction of the image (Fig. 1). The independent projections belonging to the same color are transferred between processing units and they are written down to a single processing nodes (or memory bank). In this way the scheme of the iterative loop is modified. The main iterative processing of projections is decomposed to two nested steps. The outer loop is controlled by the index of color r, while the inner loop refers to independent projections in the r–th sub–color.

The applied scheme of domain partitioning, and subsequent task decomposition make the algorithm suitable to efficient parallelization. Finally, the set of projections is divided into two compliment sets in the iterative loop: S–A – set of active color, S–NA – a number of mutually–dependent sets (colors). The processing of the active color consists of three subsequent steps T5 → T6 → T7 (Fig. 1). They are made for a single projection, and then the new locally modified form of the image $\boldsymbol{\mu}^{(i)}$ is broadcasted to other concurrently processing units. They start to calculate large number, partial and simple corrections of the approximate solution (a part of the T5 task)

$$\forall_{p_j,j \neq r} \quad p_j^{(i),n} = p_j^{(i),o} + \sum_{m=1}^{N_j} c_{jm} \cdot \left(\mu_m^{(i),r_n} - \mu_m^{(i),r_{n-1}} \right) \tag{6}$$

while the active processing node move to the next projection. The activation of the next color leads to changing of the S–A and the S–NA subsets. The new active unit begins from T6 → T7 tasks.

The presented formulation of the ART technique produced a I/O bound algorithm that was first executed in a distributed memory message–passing environment. The performance of the processing units does not create some perceptible constraints in the benchmark calculations. However, a large number of tiny–size data transfers must be made in the iterative loop which was the reason of low parallel execution speedup (Fig. 2). A shared memory multiprocessor platform seemed to be more appropriate to implement the presented formulation of the ART technique. In the next chapter, we will analyze of the algorithm efficiency

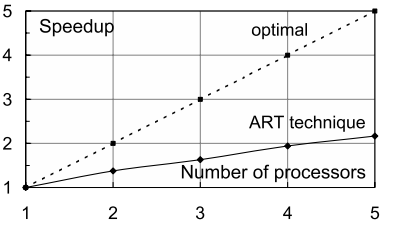


Fig. 2. Optimal speedup (dotted line) and the speedup of the ART algorithm (solid line) as a function of the number of processors in a message passing environment

in a system of dynamic shared memory processor clusters (SMP) with a new type of inter–processor communication called communication on the fly.

3 Implementation of the Method in Dynamic SMP Clusters

Dynamic SMP clusters are organized by connecting processors at program run-time to shared memory module busses. The essential concept of the new communication on the fly is based on elimination of shared data exchange through multiple transactions concerning memory modules and replacing it by switching processors (with data in their data caches) between SMP clusters which contain processors that are interested in using shared data [5, 6]. The shared data (brought by a switched processor) are made available for other processors in the cluster by means of so called reads on the fly. They consist in simultaneous parallel reading of data to processor's data caches by means of snooping a cluster memory bus while a processor writes shared data to the memory module from its data cache. In these way the multiple data reads (otherwise performed sequentially over the memory bus) are overlapped with the data writes, which speeds up program execution. The other feature is that such data exchange takes place directly between processor data caches, which is another source of communication speedup since it eliminates a series of memory–data cache transactions, otherwise necessary in the system.

The assumed system is built of dynamic shared memory clusters (A, B, ...) created by connecting processor nodes PEn ($n = 1, \ldots, N$) to busses of memory modules Mm ($m = 1, \ldots, M$) at program run–time (Fig. 3). Basic elements of a processor node (PEn) consists of a data processor (Pn) which co–operates

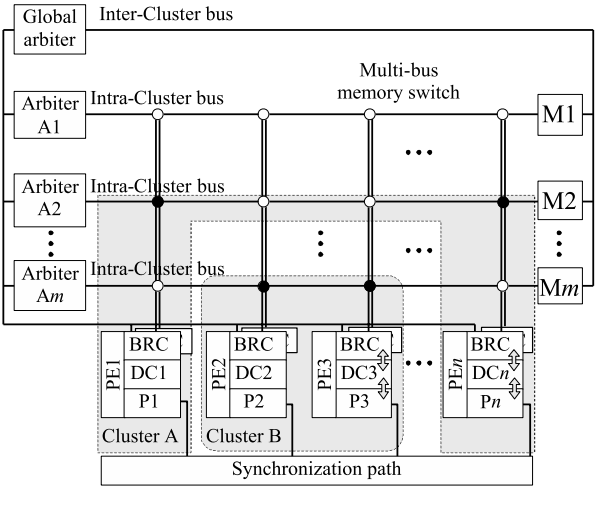


Fig. 3. System structure with dynamic SMP clusters and communication on the fly

with its data cache (DCn) and the bus request controllers (BRC). PEn can be connected to two memory modules at a time. BRCs control data communication of processor node with memory modules including initiation of switching the processor between SMP clusters and of accessing busses of memory modules.

Each cluster has its internal data bus (Intra–cluster Bus) connected to a local memory module and shared by all processors in the cluster. The bus is supervised by a local bus arbiter. The main task of a local arbiter is to attribute access rights to the memory bus on requests from processors. Arbiters fulfill also inter–cluster switching requests coming from processors. Many processors can read data from memory busses simultaneously based on snooping the address and data lines by the BRC units. Such reads on the fly are not controlled by arbiters, excluding involved write operations. All processors are also connected to the global bus, which enables for them reading data from all non–local memory modules under coordination of the global arbiter [5, 6].

The main advantage of the described system for image reconstruction algorithms is the possibility of parallel data exchange performed in synergy with dynamic processor switching between SMP clusters. Reconfiguration of system structure enables adjusting the interconnection structure to program needs. It also enables adjusting to current needs the computing power available in clusters.

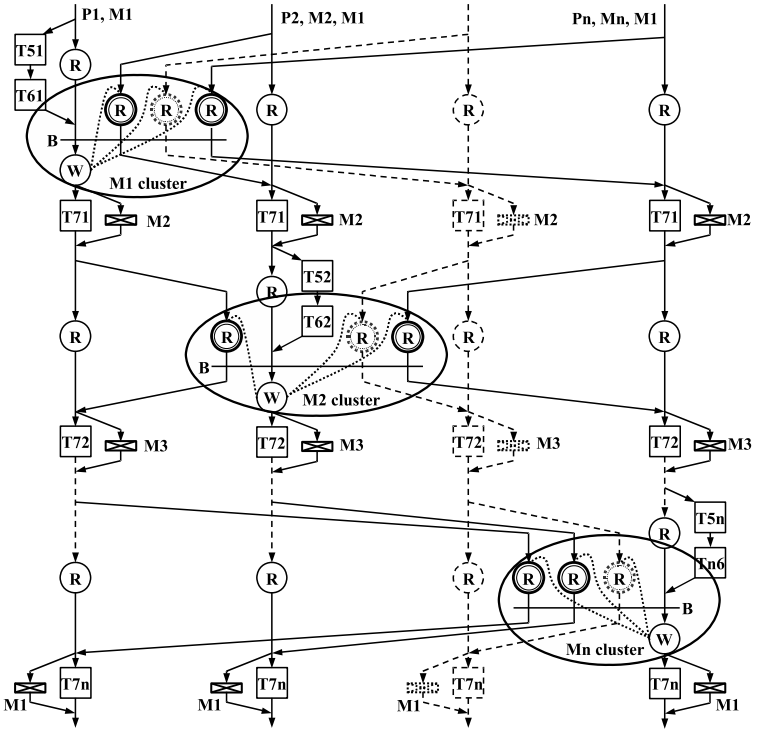


Fig. 4. Extended macro–data flow graph of the iteration loop of the ART algorithm

Efficiency of the ART algorithm executed in the system with the proposed architecture was evaluated by simulation experiments based on the use of program graph execution simulator written in C. Special extended macro–data flow graph representation was used [5, 6]. The idea of the execution of the iterative part of the ART algorithm shown in Fig. 1 in the assumed system is presented in a graph in Fig. 4. In this graph, rectangles mean computational sequential code nodes, circles with R (or W) inside denote data reads from a memory module to a processor data cache (or data write from a data cache to a memory module). A double circle with R inside denotes a data read on the fly. Such nodes are synchronized with a write node (the source of data for the reads) by a barrier (B) to be sure that all requests of data reads on the fly are deposed in BRCs before the write takes place. A small crossed rectangle means switching a processor from one SMP cluster (memory module bus) to another (indicated by a memory module — Mi, $i \in (1, \ldots, M)$). Ellipses denote dynamic processor clusters organized by processor switchings for data transmissions through reads on the fly. The graph activation paths (going top down) are assigned to processors P1, P2, ..., Pn. Each processor Pj, $j \in (1, \ldots, N)$, has 2–ported data cache at the memory modules side. One cache port of Pj is constantly connected to the memory module Mj. The second port of Pj can be dynamically switched to busses of consecutive memory modules (M1, M2, ..., Mm, see consecutive node levels at the graph in Fig. 4) to collect (via reads on the fly) the results of computations produced by nodes T7l, $l \in (1, \ldots, n)$, executed in consecutive processors.

Fig. 5 presents parallel program speedup evaluated by simulation of the ART method program graph execution. The efficiency is maximal in the case when all initial data are pre–fetched before execution of programs into data caches whose capacities are adjusted to program needs. In this case, there is no communication data cache–memory during computations and processors use initial data exclusively from caches and data supplied to data caches by other processors via reads on the fly. Such computing can be performed only if data caches are sufficiently large. However this method cannot be used when **C** matrix size is too big. In such cases, data have to be gradually pre–fetched from memory modules between computations. Introduction of multiple memory modules in such cases improves scalability and efficiency of the algorithm comparing that with a single module use.

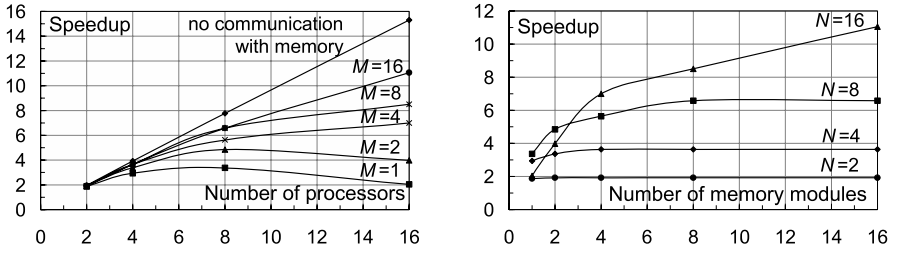


Fig. 5. ART computation speedup as a function of the number of processors (left) and the number of shared memory modules (right)

Increasing the number of memory modules enables a better distribution of data in the system and improves efficiency since processors can implement two data transmissions: data cache–memory module, at a time. However, for a given number of memory modules, increasing the number of processors above a certain limiting value can cause efficiency to fall down. It happens when too many processors compete for accessing memory modules. Program efficiency improvement can be obtained only by a coordinated increase the number of processors and memory modules. The simulation results have shown that increasing the number M of memory modules with a fixed number N of processors gives processors strong efficiency improvement if $M \leq N$. There exist boundary values of these parameters, for which the advantages from using many processors for parallel computations are reduced by contention in accessing shared memory modules.

4 Conclusions

Applying multiprocessor systems based on dynamic SMP clusters with communication on the fly brings strong shortening of the computation time in tomographic image reconstruction. The assumed system architecture strongly improves data communication in the system and enables adjusting system structure to program needs. An efficient implementation of the ART image reconstruction algorithm has to include the appropriate number of processors and shared memory modules in the system. Program graph analysis supported by introduced coloring scheme enables taking correct parallel program design decisions.

The paper has been partially sponsored by the ICS PAS and PJIIT research grants as well as the W/WE/2/03 grant of Bialystok Technical University.

References

1. Kak, A. C., Slaney, M.: Principles of Computerized Tomographic Imaging. IEEE Press, New York, USA (1988)
2. Hryniewicz, A. Z., Rokita, E.: Methods of Medical Diagnosis and Teraphy. PWN, Warsaw (in Polish) (2000)
3. Barrett, R. (ed.): Templates for the solution of linear systems: building blocks for iterative methods. SIAM, Philadelphia (1994)
4. Buyya, R.: High Performance Cluster Computing. Vol. 2. Prentice Hall PTR, New Jersey, USA (1999)
5. Tudruj, M., Masko, L.: Dynamic SMP Clusters with Communication on the Fly, Proceedings of the 2nd International Symposium on Parallel and Distributed Computing, ISPDC-2003, Lublana, IEEE Computer Society, Los Alamitos, CA (2003) 250–257
6. Tudruj, M., Masko, L.: Communication on the Fly in Dynamic SMP Clusters – Towards Efficient Fine Grain Numerical Computations, Lecture Notes in Computer Science, Vol. 3019, Springer–Verlag, Berlin Heidelberg New York (2003) 59–68

Parallel Tool for Solution of Multiphase Flow Problems

Raimondas Čiegis, Alexander Jakušev, and Vadimas Starikovičius

Vilnius Gediminas Technical University,
Saulėtekio Str. 11, LT-10223 Vilnius, Lithuania
{rc, Aleksandr.Jakushev}@fm.vtu.lt, vs@sc.vtu.lt

Abstract. *ParSol* is a library for semiautomatic parallelization of data parallel algorithms. It offers sequential and parallel arrays to be used in C/C++ algorithm implementations. The tool is developed as an extensible software package by using object–oriented techniques. *ParSol* uses MPI for interprocess communication. In this article, issues of application of parallel arrays for parallelization of `MfsolverC++` are discussed. This solver simulates two-phase immiscible flow in porous media. A short description of the mathematical model is given and a basic information on the finite-volume approximation scheme is presented. Results of computational experiments are presented.

1 Introduction

The multiphase flow in porous media has gained recently a lot of attention. This is due to the fact that problems involving the multiphase flow, heat transfer, and multicomponent mass transport in porous media arise in a broad spectrum of engineering disciplines. Important technological applications include the drying of porous solids and soils, subsurface contamination and remediation, thermally enhanced oil recovery, geothermal energy production, porous heat pipes, nuclear reactor safety analysis, high-level radioactive waste repositories, paper machines.

A standard modeling approach for multiphase flow in porous media are macroscopic models obtained by volume averaging or homogenization methods from microscopic equations. The transport phenomena are mathematically described by the basic principles of conservation for each phase separately and by appropriate interfacial conditions between various phases. Unfortunately, the resulting models are difficult to solve due to large number of strongly coupled nonlinear differential equations in the systems.

Mathematical models for multiphase flow and heat transfer in porous media were described in many papers and books (see [1, 4, 5] and references cited in these works). We have created an extensible software tool *MfsolverC++* for solution of multiphase flow problems. It is using an object-oriented approach and special numerical methods to build a solver for system of PDEs by merging together independent solvers for alone-standing equations that enter the system. Such strategy is used in Diffpack software library [6].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 312–319, 2006.

The rest of the paper is organized as follows. In Section 2, we formulate the mathematical model of two–phase immiscible flow in porous media and describe very briefly the finite-volume scheme, which approximates the system of nonlinear PDEs. The tool itself is described Section 3 and results of computational experiments are presented in Section 4. In Section 5, we give details about a parallel implementation of the tool and describe *ParSol* tool of parallel `C++`array objects. By using this tool a parallel version of the code follows semi-automatically from the serial one.

2 Model for Two-Phase Immiscible Flow in Porous Media

Isothermal two-phase immiscible flow in porous media is an example of a problem solved in our tool. For this problem we have chosen global pressure model [4]. Comparing to other models and formulations, equations in this model are less coupled and entering quantities are smoother, because most of them describe mixture properties: velocity, density, etc.

The problem is described by the system of two partial differential equations. The first equation is mass conservation equation for two-phase mixture:

$$\varepsilon \frac{\partial \rho}{\partial t} - \nabla \left(\frac{\mathbf{K}}{\nu} \Big(\nabla p - (\lambda_1 \rho_1 + \lambda_2 \rho_2)\, \mathbf{g} \Big) \right) = m, \tag{1}$$

where $m = m_1 + m_2$ is the mixture mass source or sink. In the absence of any external mass source or sink, $m = 0$. The obtained equation is sometimes called *pressure* equation and is used to find the global (mixture) pressure p. The mixture kinematic viscosity ν is positive. Thus if the absolute permeability tensor $\mathbf{K}$ of porous medium is positive-definite, so is $\mathbf{K}/\nu$. Consequently, it follows from (1) that the pressure equation is elliptic.

The second equation describes mass conservation of one of the phases:

$$\varepsilon \frac{\partial (\rho_1 s_1)}{\partial t} + \nabla (\rho \mathbf{u} \lambda_1) = -\nabla \left(\frac{\mathbf{K} \lambda_1 \lambda_2}{\nu} \Big(\nabla p_c + (\rho_1 - \rho_2)\, \mathbf{g} \Big) \right) + m_1, \tag{2}$$

where mixture velocity $\mathbf{u}$ is found from the Darcy law:

$$\rho \mathbf{u} = -\frac{\mathbf{K}}{\nu} \Big(\nabla p - (\lambda_1 \rho_1 + \lambda_2 \rho_2)\, \mathbf{g} \Big). \tag{3}$$

We call (2) *saturation* equation. It is used to find saturation s_1.

Recall that $dp_c/ds_1 < 0$ by the property of capillary pressure. Hence if $\mathbf{K}$ is positive-definite, then (2) is a degenerate parabolic equation. The degeneracy is caused by the fact that fractional mobilities λ_1 and λ_2 can become zero. When the capillary forces are small, saturation equation (2) is advection dominated. It is purely hyperbolic in the absence of these diffusive forces.

For numerical solution of the obtained system of PDEs we employ a special sequential procedure [2]. According to it, equations are first decoupled and then solved implicitly. This solution procedure is sometimes called semi-implicit. Differently from fully coupled and implicit schemes (see [1]), it is more flexible and

can be extended in the natural way, if the model is extended by additional PDEs for saturations, concentrations or temperature. Differently from IMPES methods (implicit approximation of the pressure equation and explicit approximation of saturation equations), all equations are solved implicitly, what can lead to the relaxations of the time step restrictions.

In our tool we are using a cell-centered non-uniform staggered Cartesian grid to partition the problem domain into the control volumes. PDEs are approximated by the finite–volume schemes. In this way, the mass is conserved locally (volume by volume).

The advection terms in (2) are approximated by using one of the following methods:

- Upwind approximation, which gives the monotone difference scheme. PDEs are approximated with the first order of accuracy even for smooth solutions.
- TVD schemes [5], which have the second order of accuracy for smooth part of the solution. We note that TVD schemes are nonlinear even for linear problems and the stencil of the grid used by such schemes is larger than for the upwind approximation.

The pressure equation is linearized by the Picard method. A combination of the Picard and Newton iterations is used to deal with strong nonlinearities in the saturation equation. Newton iterations generally give better convergence performance if the initial guess is good, but the Picard method is often more robust, i.e. less dependent on the initial guess.

We use harmonic or arithmetic means of coefficients to approximate diffusive fluxes on the interfaces of control volumes. Obtained matrices are 3-diagonal in 1D case, 5-diagonal in 2D, and 7-diagonal in 3D. Dirichlet, Neumann, and full flux boundary conditions are implemented.

3 Description of MfsolverC++

It becomes more and more popular in the scientific computing to exploit the object-oriented programming (OOP) techniques. We also use OOP in the design and implementation of our tool. This allows us to reduce the amount of time spent on the programming and debugging and makes all implementational aspects cleaner and simpler. According to the strategy proposed in [6] and used in Diffpack software library, we build our solvers for the systems of PDEs by merging together independent solvers for alone-standing equations that enter the system. Our solver for solution of isothermal two-phase immiscible flow problems using global pressure model is shown in Figure 1. Initially `MfsolverC++` was created as a sequential application.

Class Pressure and his children are independent solvers of pressure equation (1). Class Saturation and his children are independent solvers of saturation equation (2). Equations become coupled into the system through the coefficients. In our PDE solvers, these coefficients, including the source, initial and boundary conditions functions, are represented by virtual functions. Subclasses of PDE solvers override these functions and implement the physically relevant versions,

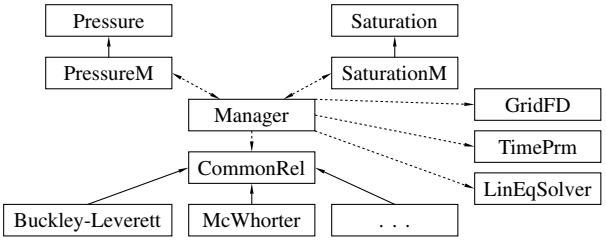


Fig. 1. Design of the PDE system solver with relations between solver classes, manager and pool of common relations. Solid arrows indicate inheritance ("is-a" relationship, with arrows pointing from subclass to base class). Dashed arrows indicate pointer ("has-a" relationship).

when the coefficients are coupled to other unknown fields in the PDE system. All these functions are often built of a common set of relations (constitutive relationships, model definitions, etc.). Therefore, they are collected in class hierarchies and accessed from PDE solvers through a base class CommonRel interface (pointer). Note that constitutive relationships can be easily changed without affecting the code in PDE solvers.

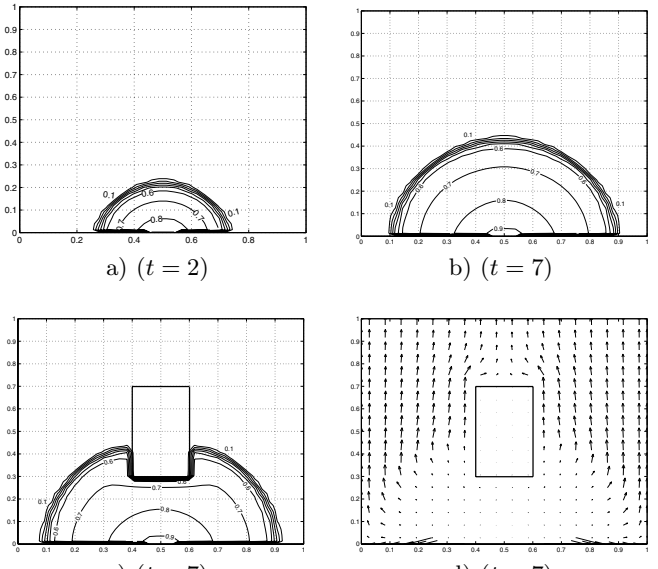
A manage class Manager acts as the solver class for the whole PDEs system. This class contains two way pointers to the subclasses for solving the pressure and saturation equations, PressureM and SaturationM, respectively, which enable the coupling by overriding the virtual functions of base classes with the functions from common relation hierarchy. The manager is also responsible for creating a space grid and time discretization. It allocates a common linear system and solver object, and distributes all these data to the PDE solver classes.

Finally, we note that such design of PDE system solver suits very well to the sequential solution procedure, which was chosen to decouple the systems of PDEs in our tool. It is naturally extensible if the differential model is extended by the additional PDEs.

4 Computational Experiments

In this paper we show how our tool solves a sample 2D problem. We consider unsteady displacement of oil by water in two-dimensional horizontal reservoir (water flooding problem). An assumption is made that external forces that are driving water into reservoir are so large that capillary forces can be neglected, i.e. saturation equation (2) is hyperbolic. We investigate the case of a homogeneous permeability field and the case with a simple heterogeneity. Results of numerical experiments are presented in Figure 2.

The reservoir is initially filled with the oil. The water is pumped in over the well at one of reservoir's boundary: $x \in (0,45; 0,55)$, $y = 0$. The oil can exit domain over the whole opposite boundary, i.e. $x \in (0; 1)$, $y = 1$. Figures 2(a) and 2(b) show saturation distribution of infiltrating water phase at time moments $t = 2$ and $t = 7$ in case of homogeneous permeability field. Brooks and Corey



a) $(t = 2)$ b) $(t = 7)$

c) $(t = 7)$ d) $(t = 7)$

Fig. 2. 2D water flooding simulation

expressions [1, 5] for relative permeabilities were used. Wells are implemented as flux-type boundary conditions.

As can be expected from hyperbolic-type problem, solution exhibits a shock wave, which is moving in all possible directions with the same speed. Figure 2(c) shows water saturation profiles in case of simple heterogeneity in reservoir, when absolute permeability inside rectangular domain $(0.4, 0.6) \times (0.3, 0.7)$ is 10^{-3} times smaller than outside. Comparing results at the same time moment $t = 7$, influence of heterogeneity can be clearly seen. In Figure 2(d) streamlines of oil flow are shown.

5 Parallel Implementation of MfsolverC++

The major difficulty in using parallel computers is that writing a parallel program (or parallelizing existing sequential codes), requires the knowledge of special methods and tools, which is not trivial to be mastered. Hence the main obstacle in the spreading parallel computing is the lack of specialists who may create parallel software. One of the ways to improve the situation is the creation of tools to simplify the parallelization of algorithms [7]. We have developed a new tool, which can be used for semi–automatic parallelization of data parallel algorithms, that are implemented in `C++`.

5.1 Parallel Array Objects

The aim of *ParSol* is to bring HPF parallelization simplicity to `C++`language, using popular parallelization standards. The current *ParSol* library features are: a)

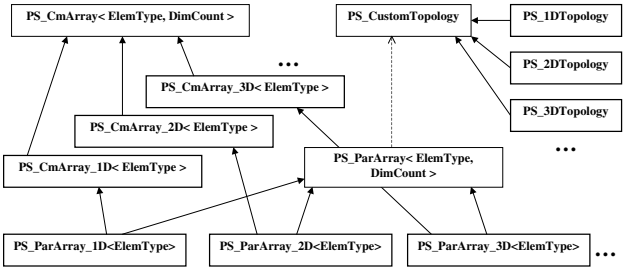


Fig. 3. ParSol library class diagram

created for `C++` programming language, b) based on HPF ideology, c) the library heavily uses such `C++` features as OOP and template and only standard `C/C++` features are used, d) MPI 1.1 standard is used to implement parallelization, e) *ParSol* is an open source library. At present, *ParSol* may be used for parallelization of data-parallel or domain decomposition algorithms. *ParSol* class diagram is shown in Fig. 3.

Next we describe the main elements of the library.

Sequential array classes. These are the classes to be used instead of native `C/C++` arrays. No other libraries, except *ParSol* itself, is necessary to use them. Comparing to native `C/C++` arrays, *ParSol* sequential arrays have a number of advantages for programming mathematical algorithms, such as virtual indexes, built-in array operations, automated management of dynamically allocated memory. The main functionality resides in template class `PS_CmArray`. However, general functionality requires interface complexity. So children are derived for special cases (i.e. $1D, 2D, 3D$ arrays), that provide a user-friendly interface. It is recommended for end-user to use those classes whenever possible.

Parallel array classes. Parallel arrays are the descendants of appropriate sequential arrays and parallelization methods are added to the sequential array functionality. We note that parallelization is similar for different kinds of arrays. So parallelization code is localized in class `PS_ParArray`, and is used in parallel array classes by multiple inheritance.

Topology classes. The purpose of these classes is to ensure that all processes are in proper order for parallel array functionality. In HPF, this functionality is performed by special directives. All the general code resides in `PS_CustomTopology` class. As with sequential array classes, there are also descendants classes, which provide the end–user with more friendly interface for one-, two- and three-dimensional cases.

Stencil classes. A stencil is determined depending on requirements of computational scheme. Based on the given stencil, different amount of information needs to be exchanged among neighbours. This part of data is required for parallel arrays to operate properly.

A user of *ParSol* tool should develop a sequential code and use array operations provided by *ParSol* wherever possible. This requirement can be called an advantage, because it frees programmer from implementing simple tasks, allowing to concentrate on problem solving, and makes code cleaner.

Parallelization of data parallel algorithms is done in few simple steps. The difference from HPF is that the user must specify explicitly the stencil of the grid used in the algorithm. Such information is required to implement additional data communication part of parallel algorithm. The other requirement is that all computations should not depend on the order in which array points are processed (for example the Jacobi iterative method satisfies this requirement, but the Seidel iterative method can not be parallelized with *ParSol*.)

The parallelization of such a sequential program takes the following steps: a) replace includes of sequential headers with parallel ones; b) replace sequential classes with their parallel analogy in variable declarations only; c) add MPI initialization code (one line at the beginning of the program); d) add topology initialization code (in its simplest case, one line at the beginning of the program); e) specify when array neighbours should exchange data.

5.2 Computational Experiments

We have tested *ParSol* on a set of benchmarks designed to measure the performance of several components and parts of `MfsolverC++` crucial for efficient performance of parallel version of the tool. A nonlinear 2D diffusion problem was approximated by the explicit Euler scheme. A detailed description of the problem is given in [3]. The problem was approximated on rectangular grids $N \times N$ with a different number of cells. Table 1 presents experimental speedup $S_p(N)$ and efficiency $E_p(N)$ values for solving discrete problems on IBM SP4 computer. A two–dimensional data decomposition $p_1 \times p_2$ was used with p_1, p_2 values as close to each other as possible.

In the second test we have solved by the Conjugate Gradient method a system of linear equations which was obtained after the discretization of the three dimensional Puasson problem by the finite–volume method. Table 2 presents experimental speedup $S_p(n)$ and efficiency $E_p(n)$ values for solving problems of different size. Computations were performed on PC cluster.

Application of *ParSol* for parallelization of `MfsolverC++` still requires some restructuring of solver's code. However we have shown that essential parts of

Table 1. The speedup and efficiency for the explicit Euler algorithm on SP4

p	$S_p(80)$	$E_p(80)$	$S_p(160)$	$E_p(160)$	$S_p(320)$	$E_p(320)$
2	1.975	0.988	1.984	0.992	2.004	1.002
4	3.741	0.935	3.928	0.982	3.986	0.996
8	6.766	0.846	7.293	0.911	7.831	0.979
12	8.701	0.725	10.19	0.849	11.216	0.934
24	14.18	0.591	18.24	0.760	21.961	0.915

Table 2. The speedup and efficiency of the CG algorithm on PC cluster

p	Iterations	n	T_p	S_p	E_p
1	188	100	24.10		
2	188	100	13.22	1.82	0.911
4	188	100	6.65	3.63	0.906
8	188	100	4.03	5.97	0.747
1	350	200	366.54		
2	350	200	185.51	1.98	0.988
4	350	200	94.99	3.86	0.965
8	350	200	51.58	7.11	0.888

the solver, i.e. solvers for saturation and pressure equations can be successfully parallelized by using *ParSol* tool.

References

1. P. Bastian. *Numerical Computation of Multiphase Flows in Porous Media.* Habilitation Dissertation, Kiel university, 1999.
2. Z. Chen, G. Qin, and R. E. Ewing. Analysis of compositional model for fluid flow in porous media. *SIAM J. Appl. Math.*, 60(3): 747–777, 2000.
3. R. Čiegis, A. Jakušev, A. Krylovas and O. Suboč. Parallel algorithms for solution of nonlinear diffusion problems in image smoothing. *Math. Modelling and Analysis*, **10**(2), 155–172, 2005.
4. R. Čiegis and V. Starikovičius. Mathematical modeling of wood drying process. *Mathematical modelling and analysis*, 7(2): 177–190, 2002.
5. R. Helmig. *Multiphase Flow and Transport Processes in the Subsurface—A Contribution to the Modelling of Hydrosystems.* Springer-Verlag, 1997.
6. H.P. Langtangen. *Computational Partial Differential Equations. Numerical Methods and Diffpack Programming.* Springer, Berlin, 2002.
7. H.P. Langtangen and A. Tveito. *Advanced Topics in Computational Partial Differential Equations. Numerical Methods and Diffpack Programming.* Springer, Berlin, 2003.

Grids for Real Time Data Applications

Geoffrey C. Fox[1,2], Mehmet S. Aktas[1,2], Galip Aydin[1,2], Hasan Bulut[1,2], Harshawardhan Gadgil[1,2], Sangyoon Oh[1,2], Shrideep Pallickara[1], Marlon E. Pierce[1], Ahmet Sayar[1,2], and Gang Zhai[1,2]

[1] Community Grids Laboratory, Indiana University, 501 North Morton Street, Suite 224, Bloomington, IN 47404
[2] Department of Computer Science, School of Informatics, Indiana University, 150 South Woodlawn Ave., Bloomington, IN 47405-7104
{gcf@grids.ucs, maktas@cs, gaydin@cs, hbulut@cs, hgadgil@cs, ohsangy@cs, spallick@grids.ucs, mpierce@cs, asayar@cs, gzhai@cs}.indiana.edu
http://grids.ucs.indiana.edu/ptliupages

Abstract. We describe our work in building support for streaming data services for Geographical Information System Grid services. We examine how streaming approaches may be used to increase data service performance for transporting XML messages. Similarly, streaming versions of traditional static map services may be combined with general audio/video session management capabilities to build collaborative, annotatable shared maps. Distributed services linked through messaging substrates require information and broker management capabilities, and we describe our research here. Finally, we discuss efficient XML representation techniques that can be used to increase performance of Web Services and support Web enabled devices.

1 Introduction

The implications of messaging and real-time data streaming in the core standards of Service Oriented Architectures [1] are just beginning to be investigated. SOAP intermediaries and distributed messaging systems may potentially greatly alter the nature of the Grid applications, creating an "Application Internet" on top of the core Internet.

As we have reviewed elsewhere [2], service oriented systems (i.e. Grids) are characterized by distinct, distributed services with well defined public interfaces. These services communicate through the exchange of messages. Both service definitions and message formats are expressible in XML using WSDL [3] and SOAP [4], respectively. Perhaps underappreciated so far is the importance of the message-based nature of service-oriented systems. Most applications have focused on the "remote procedure call" view of Web Services, and this approach has been successful in building file-system based scientific application Grids. However, remote procedure call implementations are only a convention. We may also exploit messaging approaches to handle streaming and real time data. One of the primary characteristics of this approach is the use of messaging substrates

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 320–332, 2006.

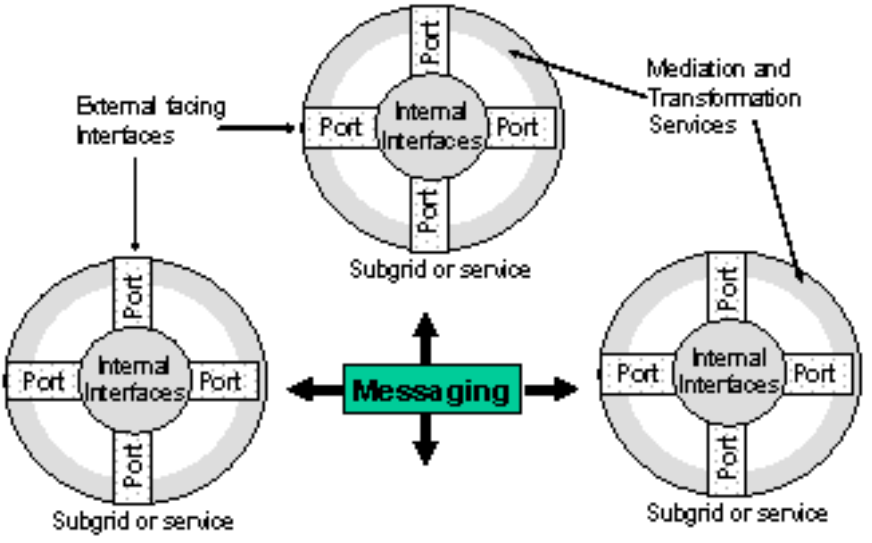


Fig. 1. Self-contained services, or collections of services, communicate through messages mediated by a messaging substrate. All communications (method invocation and responses, event notifications, data exchanges, and data streams) are messages.

[5] to support the routing of the messages, as well as provide various qualities of service such as reliability and security.

We illustrate the basic concepts in Fig. 1. We may consider the individual services as distinct structures on a messaging foundation (Fig. 1 represents a top-down view in our analogy). The individual services expose public interfaces to the rest of the Grid services. Resource-specific internal interfaces and bridges (i.e. access to a particular database or simulation application) are not public: the service mediates access to these resources. All communication between services uses distinct messages that are managed by the messaging substrate that abstract network transport protocols (TCP/IP, UPD, etc.). Services publish or subscribe to messaging channels, constructing or consuming messages as appropriate, but are not otherwise responsible for message routine or message quality of service. Similar approaches have been used to manage inter-service communication of state changes and other notifications [6], but as we advocate in this paper, the message substrate approach should apply to all messages. The system may support several modes, including client-server and peer-to-peer (see 2.3)

Fig.1 is self-similar in that we may build Grids hierarchically, where each collection of services that constitutes a particular subgrid may itself expose a single external interface. Such systems are made possible through workflow and lightweight information systems that together can be used to define these subgrids. This work is described in more detail in Section 3.

In this paper, we describe our efforts to build service oriented, real time systems based on the NaradaBrokering substrate [7]. NaradaBrokering is a

distributed, topic-based publish/subscribe system that may be used to route arbitrary message payloads. We begin with an examination of several applications: Geographical Information System (GIS)-based applications for accessing data archives, using video collaboration techniques for developing streaming map videos within collaborative sessions, and annotating video streams. We then examine two fundamental services needed to support messaging Grids: information management using WS-Context, stream and broker management using HPSearch. We conclude with an examination of techniques for improving Web Service performance by using more efficient XML representations.

2 Geographical Information System (GIS) Data Service Applications

2.1 Improving Web Feature Service Performance

The Open Geospatial Consortium (OGC) [8] standard specification Web Feature Service (WFS) [9] defines standard interfaces for web-based clients and servers to access geospatial feature data. WFS and other OGC based services use the Geography Markup Language (GML) [10] to encode geospatial data, and this provides a common language for both providers and consumers. The original WFS specification is based on HTTP Get/Post methods, but this type of service has several limitations such as the amount of the data that can be transported, the rate of the data transportation, and the difficulty of orchestrating multiple services for more complex tasks. Web Services help us overcome some of these problems by providing standard interfaces to the tools or applications we develop. We have developed a Web Service version of WFS and are testing in several scenarios where scientific data analysis tools such as Pattern Informatics [11] require fast access to large amount of data.

Our experience shows that although by using Web Services we can easily integrate several GIS and other services into complex tasks, providing high-rate transportation capabilities for large amounts of data remains a problem because the pure Web Services implementations rely on SOAP messages exchanged over HTTP. This conclusion has led us to an investigation of topic-based publish-subscribe messaging systems for exchanging SOAP messages and data payload between Web Services. We have used NaradaBrokering which provides several useful features besides streaming data transport such as reliable delivery, ability to choose alternate transport protocols, security and recovery from network failures.

Our streaming WFS uses standard SOAP messages for receiving queries from the clients; however, the query results are published (streamed) to a NaradaBrokering topic as they become available. Our initial implementation uses MySQL database for keeping geographic feature data, and we employ a capability in MySQL that streams the results row by row, allowing us to receive individual results and publish them to the messaging substrate instead of waiting for whole result set to be returned. The initial performance results show that (especially

Table 1. The performance of streaming and non-streaming versions of the Web Feature Service is compared. Timings are in milliseconds. Numbers in parentheses are standard deviations.

Event Magnitude Lower Bound	Data Size (KB)	Streaming WFS		Response Time Non-Streaming WFS
		Time for streaming the result	Total Response Time	
3	880	2414	4570 (360)	5663 (31)
3.5	287	827	3405 (48)	4414 (39)
4	106	320	2945 (50)	4099 (71)
4.5	36	100	2661 (27)	3917 (38)
5	11	31	2425 (38)	3913 (77)

for smaller data sets) streaming removes a lot of overhead introduced by object initializations. Table 1 gives a comparison of the streaming and non-streaming versions of our WFS implementations. The data requested is the Southern California seismic records for the eventful year of 1992, initially obtained from Southern California Earthquake data center [12] and converted into GML for our Web Feature Service. The first column is the minimum magnitude of the earthquake, the second column shows the data size of the query result. Timings for Streaming WFS contains two columns; the first column shows the time it takes to generate and stream out GML feature collection, the second column shows the total response time. The fourth column shows the total response time for non-streaming WFS. The difference between streaming and non-streaming WFS versions is that streaming version does not accumulate the query results and stream as soon as they become available. The timings are in milliseconds and include object initializations, query processing, database query and transport times.

We can deduce from the table that for larger data sets when using streaming our gain is about 25%. But for the smaller data sets this gain becomes about 40% which is mainly because in the traditional Web Services the SOAP message has to be created, transported and decoded the same way for all message sizes which introduces significant overhead. We are investigating new methods for reducing the overhead in the streaming WFS to further improve the performance.

2.2 Collaborative Map Videos Through the Streaming Web Map Services

The Web Map Service [13] is another HTTP GET/POST-based service that we may convert into a Web Service. Map servers are useful for building Web portal interfaces for geophysical Grid applications that integrate Web Feature Service-based data services with remote executables. Examples of this work may be found in [11]. We are also interested in going beyond static images to support the displays of time-dependent geographic data.

In our initial investigation of this problem, we generate the video image as a sequence of map images. After generating map images on the Web Map Server for each time slice for the same data layer, the WMS then converts the sequence into a video stream and publishes it to a Real Time Protocol (RTP) [14] session which is represented as <IP Address, Port Number> pair. The supported video stream formats are H.261 and H.263, which are widely used formats in videoconferencing systems. A client capable of playing those formats can connect to the RTP session and play the stream published. Map video stream can be played in collaborative environments such as AccessGrid [15] and GlobalMMCS [16] sessions.

The map video stream has several configurable parameters which affect the quality of the produced map video stream: frame rate and video format of the stream, update rate of the map images in the video stream. We use the H.261 codec and update map images every 0.5 seconds while we keep the video frame rate at 10 frames per second. This provides sufficient quality for the video stream displayed at the receiving side. The reason frame rate and image update rate are different is that some clients might not be capable of visualizing video streams with low frame rate or can visualize them with very low quality. Keeping frame rate high will improve the quality of the video shown on the player while the map image rate is kept at a different rate.

Map video streams can be published to unicast or multicast RTP sessions. AccessGrid venues are multicast sessions. A video client listening a multicast session can receive and play the stream as long as the underlying network lets client receive multicast packets. GlobalMMCS also provides this map video stream to its clients as unicast video stream.

2.3 Collaboration Tools for e-Annotation

Streaming map servers may be viewed by any compatible client, and by integrating with GlobalMMCS, we may deliver streaming map video to a range of systems (including Access Grid, Polycom, and RealPlayer); see [16]. We may also build our own custom clients and services with extended capabilities for replay and annotation.

The e-Annotation collaborative tools facilitate interactive collaboration and distance education. The e-Annotation collaborative tools work in a peer-to-peer fashion atop of the collaboration architecture based on publish-subscribe messaging middleware. All the participating peers collaboratively work with each other using the NaradaBrokering messaging overlay network to annotate a live or archived video stream. The e-Annotation player is composed of the following components, illustrated in Figure 2.

Stream List Panel: There are three stream lists in this panel: a list of real time live streams, an archived video stream list, and a composite annotation stream list. These three stream lists are dynamically updated from the RTSP server by subscribing to the streaming control info topic from one of the brokers. Each stream list is a topic.

Real time live video play panel: This panel contains a video player which can play the selected real time live video stream that user selects in the real time live

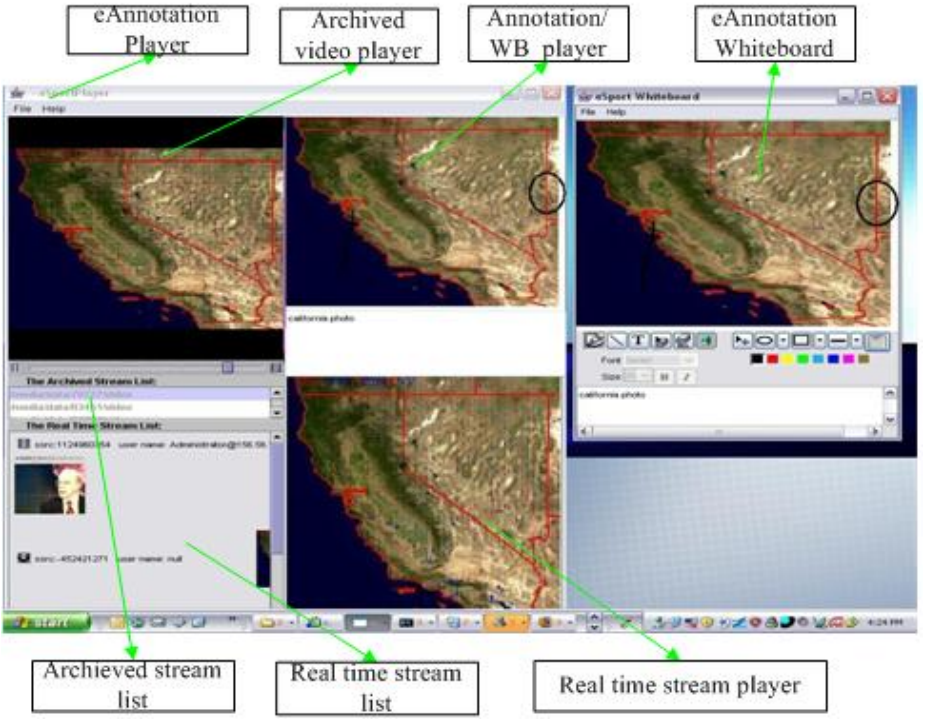


Fig. 2. The e-Annotation user interface allows video image display, capture, annotation, and playback. The image shown is LandSat imagery of the western United States, obtained from the NASA OnEarth project, http://onearth.jpl.nasa.gov/.

stream list. All the real time live video streams are published by GlobalMMCS through different NaradaBrokering topics.

Streaming player panel: This panel has a video player to play the buffered or archived video stream from the NaradaBrokering's buffers and storage nodes. This player supports pausing, forwarding and rewinding the video streams with dynamic length (the buffered live video) or fixed length (the archived video stream). It also supports taking snapshots from a playing video stream. When taking a snapshot, the timestamp is associated with that snapshot. These snapshots are loaded to whiteboard to be annotated collaboratively and saved with the original stream as a new composite stream.

Video annotation player panel: This panel contains a player to play the new created composite annotation video stream. When the annotation stream is played back, the original video stream is played in the streaming player panel, and the annotated snapshots (which are streamed by RTSP server) are played in this panel, synchronized with the original video stream by the timestamps associated with them.

3 Web Intermediary and Service Management

3.1 Service and Intermediary Management

We are developing HPSearch [17] as script-based management console for controlling services and NaradaBrokering intermediaries. HPSearch provides dual functionality: 1) it provides a high-level language suitable for application developers to program workflows in a Grid that utilizes the messaging middleware, and 2) it provides tools to manage the messaging middleware.

HPSearch enacts a simple distributed services model by leveraging the capabilities of messaging middleware for routing data in streams between services. WSProxy is a specialized component of the HPSearch system that wraps an existing service or a program while providing a Web Service interface for steering the service. It also provides an interface to the NaradaBrokering messaging system that transparently maps input / output streams to messages. Thus data is sent between services by publishing the event containing the data on a predetermined topic. Topics are automatically created by the HPSearch engine thus linking distributed services. WSProxy also presents a simple Web Service interface to help steer the service. The HPSearch runtime steers the service while the input / output is transparently managed by NaradaBrokering. NaradaBrokering has been recently augmented with topic creation and discovery [18] and message-level security. We plan on adding handlers in HPSearch to help automatically register secure streams and issue tokens to participating services. This will allow the application programmer to manage the security features of each stream that links distributed services.

HPSearch uses NaradaBrokering to route data streams between distributed services. Accessing a single broker over-and-over usually results in inefficient routing of data due to over loading. To assuage this problem we deploy a set of co-operating broker nodes that together form a virtual broker network. This broker network usually has multiple routes connecting peripheral brokers, thus providing alternate routes avoiding congestion at a single broker node. The distributed services connect to the peers at the periphery of this network. Thus, the brokering network can route data between peers much more efficiently, also providing different quality of service to each peer if required.

The scripting architecture also allows us to aggregate performance metrics in the system. These would allow us to determine congested paths and help decide alternate routes to create. The management architecture may be extended to manage remote brokers and provide means to try alternate configurations for creating links that span firewalls.

Management of such a set of multiple broker nodes and creating links between them poses a scalability issue. To address this issue, we are developing a specialized Web Service called the Broker Service Adapter. The Broker Service Adapter helps us deploy brokers on distributed nodes and setup links between them. Further broker nodes or the links between them may fail. HPSearch provides a scripting interface to instantiate new brokers at runtime and create links between brokers. The routing characteristics may be completely changed

by tearing down an existing broker network and instantiating a completely new network.

Recently, management of systems has gained much research interest within the Web Service community. WS-Management [19] and WS-Distributed Management [20] are two competing Web Service specifications that propose management of remote resources based on the Web Service architecture. We plan on extending the Broker Service Adapter architecture to incorporate a simple WS-Management based interaction while fault-tolerance and scalability is automatically handled by the underlying Broker Service Adapter architecture.

3.2 Managing Dynamic Information with WS-Context

Collections of services such as Geographic Information System and collaboration services, such as described in Section 2, may be thought of as an actively collaborating set of grid/web services where services are put together for a particular goal. Each interaction with a set of services (including both workflows described in [11] and video collaboration sessions described in 2.3) can be modeled as a session. Each session is associated with a life time and maintains rapidly updated information known as context. Simply restated, this means that context plays an important role in enabling services to correlate their activities. We use the term "gaggle" for dynamically assembled grid/web services for a particular functional collection [21]. Gaggles may be gathered at any one time and can be considered as very small part of the whole grid.

We have designed and implemented information services [22] that support dynamically generated context to meet with aforementioned requirements of rich interacting systems. We have extended existing WS-Context Specifications [23] and provided with an implementation of XML metadata services supported by a MySQL database as backend storage. The WS-Context Metadata Service keeps track of context information shared between multiple participants in grid/web service interactions. It maintains user profiles, application specific metadata, information regarding sessions and state of entities in these sessions. In order to provide fault tolerance and scalability, we have also designed distributed metadata management architecture to support dynamically assembled Grid applications where metadata is widely-scattered and dynamically generated [24]. Additional applications of WS-Context services are described in Section 4.

4 High Performance XML Transfer

In a conventional Web Service environment, XML is the presentation format, which provides interoperability to the heterogeneous participating nodes. But in some constrained computing environments, such as mobile computing and real-time computing, processing verbose XML-based messages becomes a performance bottleneck. These performance overheads consist of parsing to retrieve information from its structured representation, more transmission time with increased document size over a narrow-bandwidth mobile connection and conversion overheads from in-memory representation to textual format.

The high-performance XML encoding is an open research area [25] [26] [27]. Related work on solving these problems can be categorized as individual message optimization or message stream optimization. An individual message approach produces a simplified, efficient, and self-contained message that has different representation of XML content – XML Infoset information. XBIS [28] falls on to this category. The message stream approach optimizes a whole sequence of messages – a session. Participating nodes negotiate the characteristics of the session and the message format in the session. Fast Web Services [29] and Handheld Flexible Representation [30] [31] (described below) fall on this category.

To achieve high-performance SOAP message processing and exchange in constrained environments, we are designing Handheld Flexible Representation (HHFR) and have implemented a prototype. We focus initially on handheld applications but we see important extensions to both Web enabled sensor devices and to data-centric Web Services in general. The architecture provides the preferred representation for applications (target services, client services, or message receivers) by separating SOAP message contents from the XML syntax of the message. The architecture targets a message stream and negotiates characteristics of the stream that includes the structure and types of SOAP message, reliability and security issues, and a preferred representation. To achieve the goal, the architecture design should address several issues.

Replacement of XML Syntax with Optimized Representation: The HHFR architecture and its framework supports message exchanging in the preferred representations, which is an optimized (or binary) format in most cases. The architecture separates SOAP message contents from the XML syntax. It is responsible to build the message in the preferred format (or representation) using internal *DataStructure* object and the separated contents stored in the HHFR data model. The internal *DataStructure* is created by parsing the HHFR schema document, which is a surrogate of separated structure and types of the SOAP message. We restrict the XML Schema definition for the HHFR Schema definition. These restrictions, such as a single schema document, no facets like minInclusive and maxInclusive, no references, make parsing a HHFR Schema document produce a single structure.

Focus on Message Streams: The HHFR works best for the Web Services, where two participating nodes exchange *a stream of messages*. For applications using a specific service, messages in the stream share the same data structure and data type for information items in it. Most of message headers are not changing in the stream session. Therefore, the structure and type of SOAP message contents in HHFR schema format and unchanging-SOAP headers can be transmitted only once, and the rest of the messages in the stream has only payloads.

Context-store as a Repository: In the HHFR architecture, a *context-store* module holds a static data of the message stream including the unchanging-SOAP headers, HHFR schema as a data representation, and other stream characteristics. These characteristics are captured by a *negotiation stage*. WS-Context (see 3.2) is well suited for this purpose. Information is defined with a URI. For

example, we derived the HHFR scheme itself as *URI-S*. The current representation of the message in the stream is *URI-R* and the choice of transport protocol is *URI-T*.

Negotiation of Characteristics: to use the preferred representation and set up characteristics of the stream, the HHFR uses a conventional SOAP message in the beginning of the stream like the negotiation in the WS-SecureConversation [32] specification. The SOAP message, which has a *Negotiation* Schema defined with the HHFR Schema definition, is sent to request to start a HHFR session. It contains a HHFR Schema document of the SOAP message and any available quality of service issues.

To demonstrate the effectiveness of HHFR architecture, we have implemented a prototype mobile Web Service framework based on the HHFR architecture design. The prototype implements core design features of the design, such as the representation conversion, a fast transport option for a message streaming, the negotiation stage, and a simple HHFR data model. We choose Java as a language platform for both mobile and conventional computing and we limit the mobile node as the service client. Since the HHFR design doesn't cover the SOAP Engine feature, we utilize existing efforts – Apache AXIS and kSOAP.

We experiment benchmarks to compare the performance of the HHFR prototype with the performance of the conventional Web Service framework – AXIS. The details of the experiments and results can be found in [31] and we summarize them here. We develop two applications, a string array concatenation application and a floating number array addition application. Both applications use TCP transport for benchmarking. A string array concatenation service produces a single concatenated string of all string in a message. We measure a Round Trip Time (RTT) of the session, which includes multiple messages of given array length. The other application is a float number addition service that returns a summation of all float numbers of an array in a message. In this benchmark, RTT of the conventional SOAP application contains an OS level float-to-text conversion overhead, while RTT of the HHFR doesn't.

From the experiments, we observe a bigger performance savings on a longer session. The string concatenation benchmark and the float addition benchmark show that HHFR communication out-performs a conventional SOAP and the gap is fast-increasing as the number of messages in a session grows. These performance gaps are mainly caused by high latency of the cellular network, which uses the SprintPCS Vision service (speed up to 14.4kbps). The second observation we did is from the float number adding service and it is an efficient memory space usage of the HHFR prototype implementation by avoiding text conversions for building SOAP messages. During the benchmark, the runtime system of the prototype processes a larger size array in the message.

The high-performance XML can be achieved in many ways. For example, a binary data attachment in the SOAP message, such as MTOM [33] and XOP [34], is very popular solution to avoid a redundant encoding for already-encoded multimedia data or to preserve data integrity of the encrypted data. However the approach can be applied to only fixed data format and cannot cover user

defined data structure such as the array. Another example is compressing SOAP documents. Compressing is reducing the size of the document for narrow bandwidth connected Web Service nodes. The XML specific XMill [35] can reduce the document size in half or more. But because of another layer of processing – a compression and a decompression, the compression approach is not saving overall performance overhead in many data domains.

5 Summary and Conclusion

We have described several research efforts to investigate problems in managing streaming data, with an emphasis on applications to earth science data and collaboration through our SERVOGrid project. As we have discussed, streaming approaches are capable of increasing Web Service performance for data-centric services such as the Web Feature Service. Similarly, static Web Map Services may be transformed into video streaming services that can be further integrated with shared collaboration tools for annotation and playback. Underlying these systems is the need to manage both the services and the brokers (or, messaging intermediaries) that constitute the system. We described our efforts here in developing HPSearch and WS-Context. Performance is one of the key problems facing Web Service applications. As we discussed, efficient and flexible XML representation is one possible solution. We have initially investigated this for hand-held collaboration devices but see obvious extensions web enabled sensor devices and to more conventional data services (particularly the Web Feature Service). Several open areas for additional research are described within Sections 2, 3, and 4.

This work is supported in part by a grant from the NASA Advanced Information Systems Technology program.

References

1. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. “Web Service Architecture.” W3C Working Group Note, 11 February 2004. Available from http://www.w3c.org/TR/ws-arch
2. G. Fox, S. Pallickara and S. Parastatidis Towards Flexible Messaging for SOAP Based Services Proceedings of the IEEE/ACM Supercomputing Conference November 2004. Pittsburgh, PA.
3. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, “SOAP Version 1.2 Part 1: Messaging Framework.” W3C Recommendation 24 June 2003. Available from http://www.w3.org/TR/soap/.
4. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, “Web Service Descrption Language (WSDL) Version 2.0 Part 1: Core Language.” W3C Working Draft 3 August 2005.
5. G. Fox, S. Pallickara, M. Pierce, H. Gadgil, Building Messaging Substrates for Web and Grid Applications to be published in special Issue on *Scientific Applications of Grid Computing* in Philosophical Transactions of the Royal Society of London 2005.

6. I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems." IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.
7. S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/ USENIX International Middleware Conference Middleware-2003.
8. The Open Geospatial Consortium, Inc. web site: http://www.opengeospatial.org/.
9. Vretanos, P (ed.) (2002), Web Feature Service Implementation Specification, OpenGIS project document: OGC 02-058, version 1.0.0.
10. Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A. (eds) (2003), OpenGIS Geography Markup Language (GML) Implementation Specification. OpenGIS project document reference number OGC 02-023r4, Version 3.0.
11. G. Aydin, M. S. Aktas, G. C. Fox, .H. Gadgil, M. Pierce, and A. Sayar, SERVOGrid Complexity Computational Environments (CCE) Integrated Performance Analysis." Technical report June 2005 accepted as poster and short paper in Grid2005 Workshop, 2005.
12. Southern California Earthquake Data Center web site: http://www.data.scec.org.
13. de La Beaujardière, J. editor, 2002. Web Map Service Implementation Specification, Version 1.1.1, OGC 01-068r3. http://www.opengis.org/techno/specs/01-068r3.pdf.
14. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications. Internet Engineering Task Force Request for Comments 3550 (2003). http://www.ietf.org/rfc/rfc3550.txt
15. Access Grid, http://www.accessgrid.org
16. W. Wu, G. Fox, H. Bulut, A. Uyar, and H. Altay "Design and Implementation of A Collaboration Web-services system", Journal of Neural, Parallel & Scientific Computations (NPSC), Volume 12, 2004. See also http://www.globalmmcs.org.
17. H. Gadgil, G. Fox, S. Pallickara, M. Pierce, and R. Granat, "A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications." In proceedings of the IEEE/ACM Cluster Computing and Grid 2005 Conference, CCGrid 2005, Cardiff, UK
18. S. Pallickara, G. Fox, and H. Gadgil , "On the Discovery of Topics in Distributed Publish/Subscribe Systems." (To appear) Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing Grid 2005 Conference, Seattle, WA.
19. R. McCollum (ed), "Web Services for Management (WS-Management June 2005)." Available from http://msdn.microsoft.com/library/en-us/dnglobspec/html/ ws-management.pdf.
20. W. Vambenepe (ed), "Web Service Distributed Management: Management Using Web Services (MUWS 1.0) Part 1." OASIS Standard, 9 March 2005. Available from http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf
21. M. S. Aktas, G. Fox, M. Pierce, "Managing Dynamic Metadata as Context." In proceedings of Istanbul International Computational Science and Engineering Conference (ICCSE2005) June 2005 also please see: http://www.opengrids.org
22. B. Plale, P. Dinda, and G. Von Laszewski., Key Concepts and Services of a Grid Information Service. In Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002), 2002
23. Bunting, B., Chapman, M., Hurlery, O., Little M., Mischinkinky, J., Newcomer, E., Webber J., and Swenson, K., Web Services Context (WS-Context), available from http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf
24. M. S. Aktas, G. C. Fox, and M. Pierce, "An Architecture for Supporting Information in Dynamically Assembled Semantic Grids", 1st International Conference on Semantics, Knowledge and Grid, November 2005

25. O. Goldman, "XML Binary Characterization", W3C Working Group Note, Mar. 2005, http://www.w3.org/TR/xbc-characterization/
26. K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", Proc. Of 11^{th} IEEE Int. Symposium on High Performance Distributed Computing HPDC-11 2002, July 2002, pp. 256.
27. Fast Infoset, http://asn1.elibel.tm.fr/xml/finf.htm
28. XBIS XML Information Set Encoding, http://xbis.sourceforge.net/
29. P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, "Fast Web Services", Aug. 2003, http://java.sun.com/developer/technicalArticles/Web Services/FastWS/
30. S. Oh, H. Bulut, A. Uyar, W. Wu, and G. C. Fox, "Optimized Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications", Proc. Of the IEEE 2005 International Symposium on Collaborative Technologies and Systems (CTS 2005), St. Louis, Missouri, USA, May. 2005.
31. S. Oh and G. C. Fox, "HHFR: A new architecture for Mobile Web Services Principles and Implementations", Community Grids Technical Paper, Sep. 2005.
32. S. Anderson, at al, "Web Services Secure Conversation Language (WS-SecureConversation)", May. 2004, ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation052004.pdf
33. M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, "SOAP Message Transmission Optimization Mechanism", W3C Proposed Recommendation, Nov. 2004, http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/
34. M. Gudgin, N. Mendelsohn, M. Nottingham, H. Ruellan, "XML-binary Optimized Packaging", W3C Recommendation, Jan. 2005, http://www.w3.org/TR/2004/PR-soap12-mtom-20041116/
35. H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data", Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, May. 2000.

Modeling of People Flow in Public Transport Vehicles

Bartłomiej Gudowski and Jarosław Wąs

AGH University of Sciences and Technology, Institute of Automatics,
al. Mickiewicza 30, 30-059 Kraków, Poland
{bart, jarek}@agh.edu.pl

Abstract. Nowadays, there is a big necessity of modeling groups of people behavior. The knowledge of crowd dynamics is very useful in developing different facilities. The article contains a description of a model of passengers flow in public transport vehicles. The model is created on the basis of tram NGT-6, used by MPK SA in Krakow (Public Transport Company). The model proposed is based on Cellular Automata Technology combined with Multi-Agent theory. The interactions among passengers (agents) are included. The dimensions and the shape of an agent make it possible to take into account the real behavior of a passenger.

1 Introduction

Over the last years, an increasing interest in the movement patterns of groups of people could have been observed mainly, because the knowledge of people behavior is necessary for developers of all public facilities. In the article, the authors propose a simulation model of passengers movement in public transport vehicles, particularly for trams and buses.

There are two possible ways of modeling people movement: continuous and discrete. Generally, discrete models seem to be better for describing detailed passengers behavior, while most of the models are based on Cellular Automata (CA). Let us describe some of them. In 1992 Nagel and Schreckenberg proposed a 1D CA model of vehicles movement (on the road). Probably, the first work, which describes collision avoidance in movement of people was proposed by Fukui and Ishibashi in 1999 [1]. Another multi-directional pedestrian walkway models were presented by Blue and Adler in 1999 and 2000 [2]. In 2001 Burstedde at al., proposed a model which included two kinds of floor fields: a dynamic and a static one [3]. Dijkstra et al. proposed a multi-agent model of people movement in a shopping center (2001, 2002) [4, 5]. In 2003 Klüpfel proposed model of vessel evacuation based on static potential field [6]. In 2004 Gloor et al. proposed a simulation of hikers behavior in Alps [7]. Another interesting paper written by Narimatsu et al. (2004), presents patterns of collision avoidance [8]. In 2004, 2005 Wąs and Gudowski presented a model including strategical abilities in pedestrian movement [9, 10].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 333–339, 2006.

2 Some Assumptions of the Model

A simulation of the behavior of a crowd of passengers in a public transport vehicle requires a different set of assumptions, than the classical models based on Cellular Automata. The first important difference is space discretization. The majority of the known models use a quadratic lattice (or hexagonal), in which an agent is represented by a circle inscribed in a cell of the lattice. For models of smaller size, this is an excessive simplification. The authors propose a more precise representation of agents by the shape of ellipse. This could work better for imaging the interactions among agents. Each pedestrian is represented by an ellipse which occupies two cells (an agent is standing Fig. 1a), two cells plus some parts of other two cells (an agent is standing across Fig. 1b, 1c), or four cells (an agent is sitting on a seat). Geometry and possible movement directions are presented in Fig. 1.

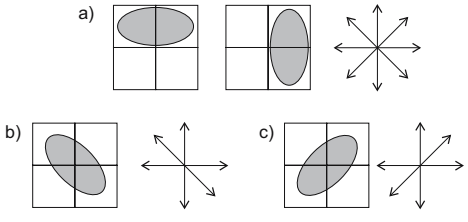


Fig. 1. Agent representations and their possible movement directions

A single cell has dimensions of 0.25×0.25m. The size of cell is based on average human dimensions, according to WHO data: shoulder breadth: $X = 0.45$m, body depth $Y = 0.27$m (Fig. 2).

The second assumption is connected with the movement in the model. The authors propose 1 cell/time-step-slice in all possible directions. A conclusion is that the movement could be realized in complex Moore neighborhood, composed of two simple Moore neighborhoods. Thus, space discretization is similar to Margolus neighborhood (Fig. 3).

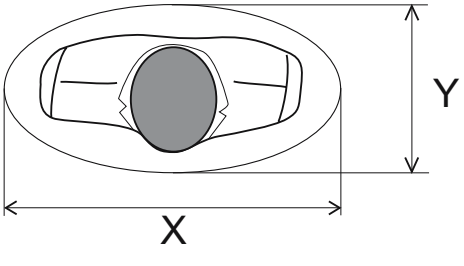


Fig. 2. Body dimensions of an agent, and ellipse of the body (according to WHO data)

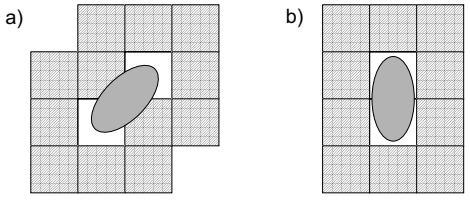


Fig. 3. Complex Moore Neighborhood, connection of two simple Moore neighborhoods

The description of the neighborhood proposed is shown below:

$$X(a, \lambda) = N(l, \lambda) \cup N(r, \lambda) \tag{1}$$

$X(a, \lambda)$ - neighborhood of an agent a,λ - radius of neighborhood X,
l - left side (arm) of a pedestrian,
r - right side (arm) of a pedestrian,
$N(l, \lambda)$ - Moore neighborhood of the left side of pedestrian,
$N(r, \lambda)$ - Moore neighborhood of the right side of pedestrian,
λ - radius of Moore neighborhood.

Value of parameter $\lambda = n$ describes the possibility of movement in n time steps. One of main reasons for using elliptic shape of agents is the problem of crowd compressibility, which could be better reflected with the application of the ellipse.

3 Model Description

The situation analyzed is passenger movement in public transport vehicles. Let us consider NGT-6, a low-floor tram used by MPK SA (Kraków Public Transport Company). The tram runs between two tram terminals. In each stop it opens the doors and some passengers exit it and some enter it. All the time, the passengers/agents have the possibility of moving inside the tram, for example validate tickets or to look for and take a seat etc. There are some places in the tram which could attract a passenger: validating machines, seats, places situated near doors etc.

3.1 Formalization

Let us apply in this work the following formal definition of Cellular Automata: (L, S, N, f), where: L - Set of cells of the lattice, S - Set of states, N - Set of neighbors, f - transition function (Weimar 1998). Although the model presented is an extension of classical Cellular Automata, the majority of mechanisms from CA are taken into account.

A configuration $C_t: L \rightarrow S$ is a function which associates each state with a grid cell. An equation of changing configuration is shown in term (2) with a supplement (3).

$$C_{t+1}(r) = f\left(\{C_t(i) | i \in N(r)\}\right) \tag{2}$$

where:
$N(r)$ - Set of neighbors of cells r,
r - Current cell number,
t - $t = t + 1$ discrete time step,
i - Single cell.

$$N(r) = \{i \in L | r - i \in N\} \tag{3}$$

3.2 Space Discretization

Presented model has 2D graphics. The surface in the model is divided into a quadratic lattice. There are distinctive cells in the model such as: walls, exits/entrances, inter-mediate aims (seats, ticket validating machines etc.). Marked edges, which indicate mean prohibited pass between two cells, represent various barriers. The figure below presents the layout of the tram NGT6 in the form of a model lattice (Fig. 4). Black cells represent walls, grey cells represent seats and blue cells represent open doors.

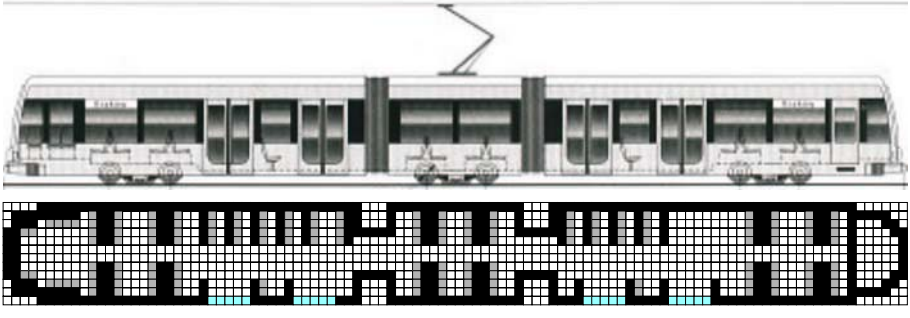


Fig. 4. Layout of tram NGT6 adapted to a model

3.3 Movement Algorithm

Tram runs between two terminals and it stops and opens doors at every tram stop. Agents are randomly generated with a random set of features such as destination (length of journey), the necessity of validate the ticket (single or periodical ticket), age. Agents can move in the model all the time (both when the tram moves and stops). Agents' features determine movement characteristics. For instance, if an agent is elderly, and their journey is of long distance, they want to take a seat. If an agent has a single ticket, they want to validate it immediately after getting to a vehicle. If an agent's journey is short, they stay in a short distance to the door. Thus, there are some defined cells in the lattice which represent intermediate aims (seats, validators, neighborhood of doors etc.). These cells are referred by authors as tarpits, because they stall agents. Every agent has a defined set of priorities and a timetable of tarpits.

Agents could move in the directions shown in Fig. 1. They can turn round, but the center of rotation could be situated only in the center of cell. General

movement algorithm is presented below in the Fig. 5. In this case we have a model with the maximum pedestrian speed $V_{\max}$ is larger than 1 simulation has to proceed within a time-step-slice given by formula $t/V_{\max}$, where t is a time step, and it usually equals 1 second [4]. Agents with the velocity V_{p} less than $V_{\max}$ do not move at every time-step-slice. The authors propose a simple way of creating automatically a movement structure for every agent with a given velocity. For every agent an integer part of ratio $V_{\max}/V_{\mathrm{p}}$ determine the time interval (measured in time-step-slices) between the movements. The fractions are

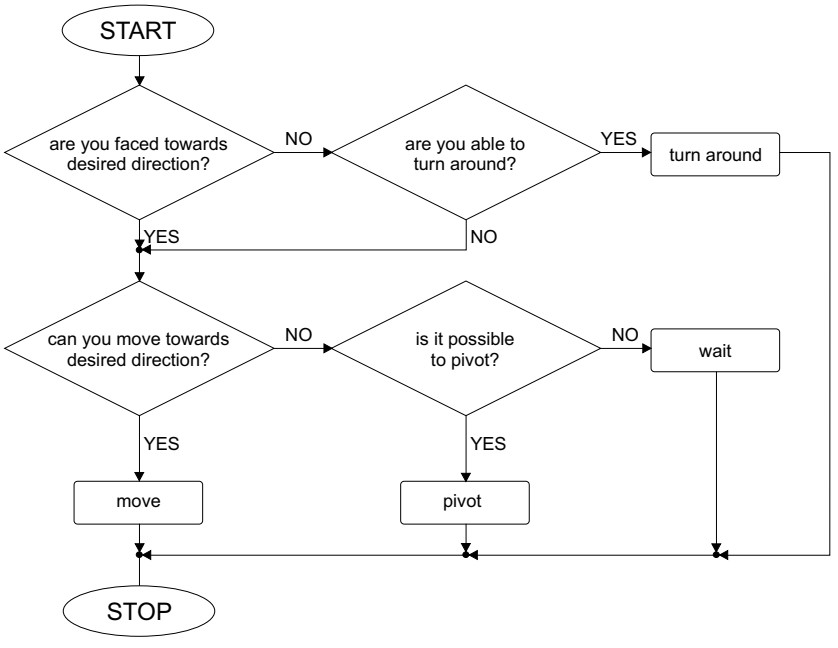


Fig. 5. General movement algorithm

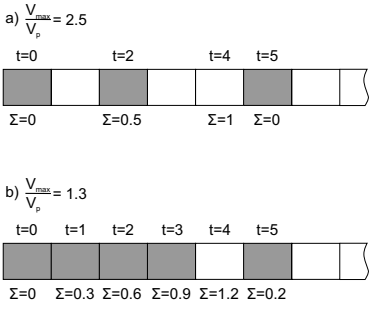


Fig. 6. Example of automatic movement structure creation

accumulated and when their total equals one or becomes larger, the agent waits one additional time-step-slice. Then, we subtract one from the total and proceed as described above. In Fig. 6, two cases of described method are presented. Gray fields indicate time-step-slices (t) in which movement occurs.

4 Application Description

The model described has been implemented as C++ application. The MFC library has been used for developing a user interface. The simulation model is represented by C++ class SModel.

The components of the model are: a grid with known length and heigth (class SGrid), a set of cells (class SCell), a set of agents (class SActor with its derivative SPassenger).

5 Simulation Results

In the current stage of research the authors want to formulate a set of rules which would reflect a real behaviour of passengers in public transportation vehicles. The model proposed seems to be accurate. The figures below present two cases. In the first we can observe passenger flow when vehicle's doors are open in the tram stop. And in the second we can see a situation inside a vehicle moving between the stops.

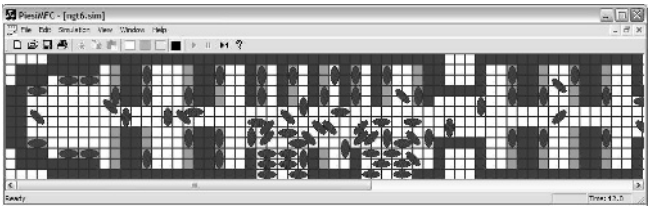


Fig. 7. Screenshot. Passengers flow during stopping at a tram stop.

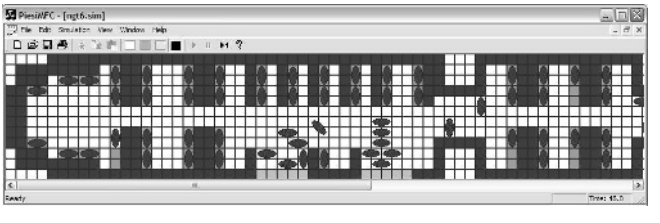


Fig. 8. Screenshot. Passengers flow between tram stops.

6 Conclusions

The model presented describes passenger movement in public transport vehicles. The simulation shows, that the agents behavior of the model is quite similar to the behavior of passengers in reality. But there are some limitations to the model which should to be developed and finally validation should be executed. One of the biggest difficulties in the model is crowd compressibility. The authors plan to apply described model in order to minimize the a tram needs to wait for the passengers to get on (average waiting time at a stop), minimize the evacuation time of a crowded vehicle, and make ticket punchers more available. These goals could be achieved by evaluating the quantity and the placement of doors and punchers and the arrangement of space inside a vehicle (seats, barriers, etc.).

The approach presented is based on Cellular Automata with substantial modification - not only local by also global relations are taken into consideration. Every agent occupies not one, but two cells or four cells. Therefore, besides progressive movement, rotation is also permitted. Thus, passenger can squeeze through the crowd. Tarpit cells are introduced which allows us represent intermediate aims in simulations.

References

1. Fukui M., Ishibashi Y.: Self-organized phase transitions in CA-models for pedestrians. J. Phys. Soc. Japan (1999) 2861–2863
2. Blue V., Adler J.: Bi-directional emergent fundamental flows from cellular automata microsimulation. Proceedings of ISTTT, Amsterdam (1999) 235–254
3. Burstedde C.K., Klauck K., Schadschneider A., Zittartz J.: Simulation of pedestrian dynamics using a 2-dimensional cellular automaton, Phys. Rev. A 295 (2001) 507–525.
4. Dijkstra J., Jessurun A.J., Timmermans H.: A multi-agent cellular automata model of pedestrian movement, Pedestrian and evacuation dynamics, Springer-Verlag Berlin (2000) 173–181.
5. Dijkstra J., Jessurun A.J., Timmermans H.: A multi-agent cellular automata system for visualising simulated pedestrian activity, Proceedings of TPICA 2000, (2001) 29–36.
6. Klüpfel H.: Cellular automaton model for crowd movement and egress simulation, Doc. thesis Duisburg-Essen (2003)
7. Gloor C., Stucki P., Nagel K.: Hybrid techniques for pedestrian simulations, Proceedings of 6th International Conference on Cellular Automata for Research and Industry, Amsterdam (2004) 581–590
8. Narimatsu K., Shiraishi T., Morishita S.: Acquisiting of local neighbour rules in the simulation of pedestrian flow by Cellular Automata, Proceedings of 6th International Conference on Cellular Automata for Research and Industry, Amsterdam (2004) 211–219
9. Wąs J., Gudowski B.: The application of cellular automata for pedestrian dynamic simulation., Automatyka Journal AGH-UST, Kraków (2004) 303–313
10. Wąs J., Gudowski B.: Simulation of strategical abilities in pedestrian movement using Cellular Automata, Proceedings of 24th IASTED MIC Conference, Innsbruck (2005) 549–553

Parallel Processing in Discrimination Between Models of Dynamic Systems

Bartosz Kuczewski[1], Przemysław Baranowski[2], and Dariusz Uciński[2]

[1] University of Zielona Góra, Institute of Control and Computation Engineering
[2] University of Zielona Góra, Computer Centre,
ul. Podgórna 50, 65-246, Zielona Góra, Poland
{B.Kuczewski, D.Ucinski}@issi.uz.zgora.pl, P.Baranowski@ck.uz.zgora.pl

Abstract. The paper considers the problem of determining an optimal observation schedule for discrimination between competing models of a dynamic process. To this end, an approach originating in optimum experimental design is applied. Its use necessitates solving some maximin problem. Unfortunately, a high computational cost is the main reason for limited practical applications, especially regarding distributed parameter systems. The paper constitutes an attempt to overcome such an impediment via a parallel implementation performed on a Linux cluster. The resulting numerical scheme is validated on a simulation example motivated by problems arising in chemical kinetics.

1 Introduction

Satisfactory simulation and control of complex phenomena usually involves knowledge about analytical models of the investigated processes. Issues regarding the quality of a model, its structure and parameter values are of great significance in all practical applications, e.g. technical diagnostics of industrial processes [7]. The existence of several plausible models, obtained as a result of performed experiments is rather a common situation in such structural identification tasks. Moreover, we have to bear in mind that for dynamic processes the quality of identification algorithms strongly depends on the manner of taking observations [17, 14]. The foregoing facts give rise to the question of how to design observation strategies for discrimination between alternative models.

Application of optimum experimental design methods to distinguish between candidate models constitutes an attractive approach due to the maximized discrimination certainty. For example, application of the T-optimality criterion is equivalent to maximization of the power of a test for the fit of a second model when the first one is assumed true [5]. In addition to that, an optimally designed experiment minimizes the experimental effort, since the observations are taken only at the most informative points (time instants). This economic aspect is very important in situations when measurements are expensive or difficult to perform.

The design of optimum experiments for the most accurate parameter identification originated in the late 1950's [6] and nowadays constitutes a well-known

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 340–348, 2006.

approach, rich in the literature and applicable for a wide variety of systems [1, 5, 16, 12]. Various measurement strategies have been considered, including determining the optimum trajectories of movable sensors for distributed parameter systems which is computationally most demanding. The recent comprehensive monographs [11, 15] provide good examples of this trend.

In contrast, the much higher level of complexity in the case of the experimental design for discrimination between competing models has been limiting its application to relatively simple models, usually static and linear [2, 1, 5]. In the past few years the approach based on the T-optimum criterion was generalized regarding multivariate models of dynamic systems [16]. Its use implies the necessity of solving a complex maximin problem. The same criterion was adopted for determining an optimum location of stationary sensors for discrimination between models of distributed parameter systems [9]. Nevertheless, serious computational efforts, indispensable to find optimum designs in acceptable time using a single low-cost computer, still limit practical applications in domains such as the prediction of atmospheric pollution or the diagnosis of complex industrial processes.

The present paper demonstrates a possibility of parallelizing computations for the classical iterative scheme of Wynn-Fedorov type, adopted to find T-optimum designs for multivariate models of dynamic systems described by systems of ODE's [2, 8]. Such an approach, never considered in the literature, significantly speeds up computations and, what is even more important, offers possibilities of further developments, which is especially attractive in the field of discrimination between models of distributed parameter systems using modern measurement techniques, e.g. moving sensors.

2 Motivating Example

Chemical kinetic models constitute a wide class of dynamic systems [3]. Methods of parameter estimation for that kind of equations, when their structural forms are fixed, are relatively well developed [14]. But when the mechanism of a reaction is not fully known, several alternative models are often proposed. As an example, we consider two chemical reactions describing conversion of substance A into substance B, which in turn changes into substance C:

– consecutive irreversible reactions $A \overset{\theta_1}{\to} B \overset{\theta_2}{\to} C$ described by the equations

$$\begin{aligned} \frac{\mathrm{d}[A]}{\mathrm{d}t} &= -\theta_1 [A]^{\lambda_1}, \\ \frac{\mathrm{d}[B]}{\mathrm{d}t} &= \theta_1 [A]^{\lambda_1} - \theta_2 [B]^{\lambda_2}, \\ \frac{\mathrm{d}[C]}{\mathrm{d}t} &= \theta_2 [B]^{\lambda_2}, \\ [A]_{t=0} &= a_0, \quad [B]_{t=0} = b_0, \quad [C]_{t=0} = c_0, \end{aligned} \tag{1}$$

where $[A]$, $[B]$ and $[C]$ denote the concentrations of reagents A, B and C, respectively, and a_0, b_0, c_0 are initial concentrations.

– a reversible first-order reaction followed by an irreversible reaction $A \underset{\theta_3}{\overset{\theta_1}{\rightleftharpoons}} B \overset{\theta_2}{\to} C$ described by the equations

$$\begin{aligned} \frac{\mathrm{d}[A]}{\mathrm{d}t} &= -\theta_1[A] + \theta_3[B], \\ \frac{\mathrm{d}[B]}{\mathrm{d}t} &= \theta_1[A] - \theta_2[B]^\lambda - \theta_3[B], \\ \frac{\mathrm{d}[C]}{\mathrm{d}t} &= \theta_2[B]^\lambda, \\ [A]_{t=0} = a_0, \quad [B]_{t=0} &= b_0, \quad [C]_{t=0} = c_0. \end{aligned} \tag{2}$$

The parameters θ_i and λ_i are unknown and, consequently, they are estimated from experimental data. In many chemical systems the reverse reaction is often neglected when it is very slow, which results in the simpler consecutive model (1). The validity of such an assumption may be examined using a model discrimination technique, when usually the time points to observe the concentrations are determined during an optimization process.

3 Optimum Experimental Design Problem

We consider a general non-linear system whose responses can be observed up to additive random errors. The model of that system has the state-space representation in the form

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t)), \quad x(0) = x_0, \quad t \in [0, t_f]. \tag{3}$$

Here t stands for time, $x : \mathbb{R} \to \mathbb{R}^s$ is the system state and t_f is the time horizon. Moreover, f is required to be continuously differentiable. Thus our basic assumption is that the observations $y_i \in \mathbb{R}^s$ of process *responses* are described by the following model:

$$y_{ij} = x(t_i) + \varepsilon_{ij}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, r_i, \tag{4}$$

where n is fixed and chosen prior to the experiment. In this description $t_i \in T$ stands for a time moment when the system state is measured, $t_i \neq t_\kappa$ whenever $i \neq \kappa$, T being some known compact set of allowable measurement points. The errors ε_{ij} are zero mean, uncorrelated and sampled from a normal distribution. The additional index j is necessary if the observations are to be repeated several times for some moments t_i, as in practice repeated experimental runs typically lead to different observed responses, even if the t_i's are exactly the same. Here the number of replications for a given time moment t_i is denoted by r_i, $\sum_{i=1}^n r_i = N$.

The basic assumption is that the model structure $f(x)$ coincides with either $f_1(x, \tilde{p}_1)$ or $f_2(x, \tilde{p}_2)$, where functions $f_1 : \mathbb{R}^{s+m_1} \to \mathbb{R}^s$ and $f_2 : \mathbb{R}^{s+m_2} \to \mathbb{R}^s$ are given *a priori*, with $\tilde{p}_1 \in P_1 \subset \mathbb{R}^{m_1}$ and $\tilde{p}_2 \in P_2 \subset \mathbb{R}^{m_2}$ being constant parameters which are fixed but unknown to the experimenter (P_1 and P_2 denote

some known compact sets). The purpose of the experiment is to determine which of the models f_1 and f_2 is true. There is no loss of generality in assuming that the first model is true, i.e., $f(x) = f_1(x, \tilde{p}_1)$, where $\tilde{p}_1$ is some given value regarded as known prior to the experiment (this value could be obtained based on some preliminary experiment or we can base on some nominal values if they are available). To make the discrimination between the models f_1 and f_2 as accurate as possible then amounts to selection of t_i's and w_i's so as to maximize the non-centrality parameter [5]:

$$T_{12}^0(\xi_N) = \min_{p_2 \in P_2} \sum_{i=1}^{n} w_i \|x(t_i) - x_2(t_i; p_2)\|^2, \tag{5}$$

where $x(t) = x_1(t, \tilde{p}_1)$, $w_i = r_i/N$, and the collection of variables

$$\xi_N \stackrel{\text{def}}{=} \begin{Bmatrix} t_1, \ \ldots, \ t_n \\ w_1, \ldots, w_n \end{Bmatrix} \tag{6}$$

is called the normalized N-observation exact design of the experiment. The t_i's and w_i's are said to be the support and weights, respectively.

Motivations behind the criterion (5) are intuitively clear, as it constitutes a measure of the discrepancy between the responses of both the models: a good design for discriminating between the models will then provide a large lack of fit in terms of the sum of squares for the second model.

The allowed replications of observations at support points t_i entail serious difficulties, as the resultant numerical analysis is not amenable to solution by standard optimization techniques, particularly when N is large. This is caused by the discrete nature of the N-observation exact designs, since the weights w_i are the multiples of the reciprocal of N. To alleviate this inconvenience, the notion of the design is relaxed to all probability measures ξ over T which are absolutely continuous with respect to the Lebesgue measure and satisfy by definition the condition

$$\int_T \xi(\mathrm{d}t) = 1. \tag{7}$$

The set of all such measures is denoted by $\Xi(T)$. They constitute the so-called *continuous* designs [5]. Thus, the relevant continuous generalization of the criterion (5) is as follows:

$$T_{12}(\xi) = \min_{p_2 \in P_2} \int_T \|x(t) - x_2(t, p_2)\|^2 \, \xi(\mathrm{d}t). \tag{8}$$

Any design satisfying

$$\xi^\star = \arg \max_{\xi \in \Xi(T)} T_{12}(\xi) \tag{9}$$

is then called the *local T_{12}-optimum design.*

4 Numerical Construction of Optimum Designs

In order to find numerical approximations of optimum designs, the generalized iterative Fedorov procedure [2, 8] was chosen for a parallel implementation. The algorithm in its basic, sequential version can be represented by the following steps (assuming that n stands for the number of support points at the current step):

Step 1. Choose an initial nonsingular design ξ_0. Set $k = 1$.
Step 2. In the k-th step find

$$\hat{p}_{2k} = \arg\min_{p_2 \in P_2} \sum_{i=1}^{n} w_i \|x(t_i) - x_2(t_i, p_2)\|^2, \quad t_k = \arg\max_{t \in T} \|x(t) - x_2(t, \hat{p}_{2k})\|^2.$$

Step 3. If $\phi(t_k) - \Delta(\xi_k) \leq \epsilon$, where

$$\phi(t_k) = \|x(t_k) - x_2(t_k, \hat{p}_{2k})\|^2, \quad \Delta(\xi_k) = \sum_{i=1}^{n} w_i \|x(t_i) - x_2(t_i, \hat{p}_{2k})\|^2,$$

then $\xi^\star = \xi_k$. STOP. Otherwise, go to Step 4
Step 4. Choose α_k with $0 \leq \alpha_k \leq 1$ and compute the convex combination of designs:

$$\xi_{k+1} = (1 - \alpha_k)\xi_k + \alpha_k \xi(t_k)$$

where $\xi(t_k)$ is the design concentrated only at one support point t_k.
Set $k = k + 1$ and go to Step 2

The choice of sequences α_k is not unambiguous [2]. In our case we set $\alpha_k = 1/(1+n_k)$, where n_k stands for the number of support points in the k-th iteration. A drawback of this kind of iterative algorithms is the clusterization phenomenon, which means the tendency to taking observations at adjacent (but different) time instants. This may result in an excessive number of support points. It is thus important to additionally use a postprocessing method in order to remove clusters and obtain a solution minimizing the size of the design (and thus the measurement effort) [13]. Computationally, Step 2 is the most demanding part of the algorithm, since the solution of the appropriate global optimization problems is crucial for the convergence of the entire scheme. It is also the most time-consuming part of the scheme. That is why main attention was concentrated on parallelizing that part of computations. Thus, in our approach a number of *slave* processors compute global minima in parallel and the reminder of the algorithm is executed by the *master* processor. As a global optimizer, the extremely simple adaptive random search (ARS) strategy was chosen [17] (a global minimizer for problems with box constraint). It belongs to the group of stochastic algorithms and is especially suited for the case when the admissible set of decision variables X is a hypercube. The procedure consists of two main phases – variance selection, when trial points are generated in order to fix an optimum range of search (a small variance of random increments in the decision variables enhances local

convergence, whereas a large one allows for an escape from a local minimum) and variance exploitation, when the best variance value found in the previous phase is used to generate a number of trial points by disturbing the initial guess with Gaussian noise.

A parallel implementation of the ARS scheme can be achieved by:

❶ using 'multistart', when each of the processors conducts a search in the entire admissible domain, but using different initial guesses; additionally, each processor can perform several search cycles, all with different initial guesses,
❷ using a partition of the search domain, when each processor makes a search only in a specific part of the entire domain; several search cycles per processor are also possible,
❸ using a combination of the foregoing two methods.

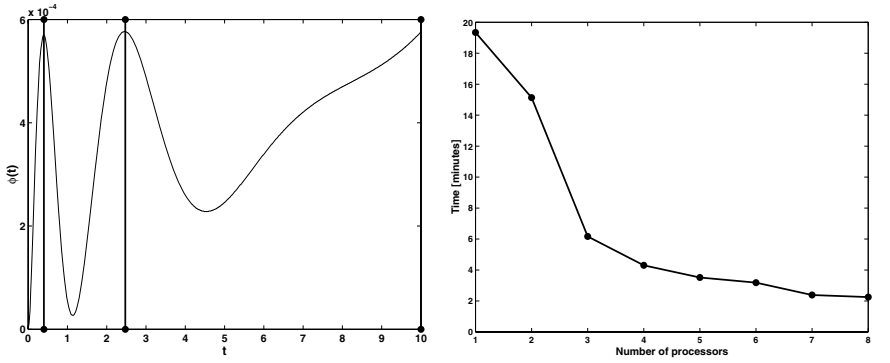
Again, bear in mind that the quality of the solutions obtained during Step 2 of Fedorov's scheme has strong influence on the convergence of the whole algorithm. Thus, it may be profitable to additionally perform several search cycles by each processor, even when the 'multistart' approach is used. Although this makes the time of computations on each processor larger, a better search of the space of decision variables may result with a reduction in the number of main loop iterations necessary to reach a desired numerical accuracy ϵ.

5 Experimental Results

In order to verify the efficiency of the presented approach, the discrimination problem formulated in Sec. 2 was considered. The model described by (2) is assumed to be true with nominal parameter values $(\theta_1, \theta_2, \theta_3, \lambda) = (0.7, 0.2, 0.1, 2.0)$. The feasible ranges of parameters for the alternative model (1) are set to $0.55 \leq \theta_1 \leq 0.85$, $0.05 \leq \theta_2 \leq 0.35$, $1.5 \leq \lambda_1 \leq 2.5$, $1.5 \leq \lambda_2 \leq 2.5$. The time horizon (design region) and initial concentrations are respectively set to $T = [0, 10]$ and $x_0 = (a_0, b_0, c_0) = (1, 0, 0)$ for both the models. Note, that the observation vector $x(t_i)$ contains three elements, i.e., $x(t_i) = ([A(t_i)], [B(t_i)], [C(t_i)])$.

The program used for parallel computing of optimum designs was written completely in Fortran 95 using ifort – Intel®Fortran Compiler v.8.1 for Linux 64-bit platforms and the mpich-1.2.6 implementation of MPI for message passing [10]. Computations were performed on a Linux cluster which was recently built at the University of Zielona Góra within the framework of the national CLUSTERIX project [18, 4]. This homogenous cluster is equipped with four SMP nodes with two 64-bit Intel Itanium II 1.4GHz processors each, running under the control of GNU/Linux Debian for ia64. The connection between the nodes is realized via Gigabit Ethernet.

The parallel implementation of the ARS procedure was realized using strategy ❶ ('multistart') in a *master-slave* model, when one of the processors plays a managing role and assigns subtasks to the remaining processors, gathers partial results from them and then determines the ultimate solution in the sequential



(a) (b)

Fig. 1. Plot of the $\phi(t)$ function for $\xi^\star$ vs. the support location (vertical lines) (a) and the plot of the obtained total computation time (b)

part of the program. In order to examine time speedup, we performed numerous experimental runs and obtained virtually the same solution (optimum design) each time – differences resulting from numerical inaccuracies appeared only from the second decimal place. The resulting optimum design contains three support points and has the form

$$\xi^\star = \left\{ \begin{array}{c} 0.40, \;\; 2.47, \;\; 10.00 \\ 0.229, 0.327, 0.444 \end{array} \right\}.$$

This means that almost half of the measurement effort should be concentrated on the time instant $t = 10s$. As can be seen from Fig. 1(a) containing a plot of the sensitivity function $\phi(t) = \|x(t) - x_2(t, p_2^\star)\|^2$, support points are located at the maxima of $\phi(t)$, which is consistent with T-optimality theory [15, 5]. The compiled program was executed using various numbers of processors. The number of search cycles performed on each *slave* processor depended on the total number of processors to achieve in total about 32 independent starts of the ARS algorithm per one iteration of the Fedorov procedure. Figure 1(b) presents the averaged total processing time as a function of the number of processors. When analyzing the obtained speedup, it is worth to noticing that despite the constant, deterministic computation time of a single ARS run (since the number of generated trial points is fixed *a priori* and remains constant), the ARS algorithm itself is probabilistic. This means, that the quality of the obtained solution may vary from run to run, which, in consequence, may influence the convergence of the Fedorov scheme (see also comments in Sec. 4).

6 Conclusions

The paper discussed the problem of parallel computations performed on a homogenous Linux cluster in order to determine T-optimum experimental designs

for discrimination between models of dynamic systems described by system of ODE's. The obtained results are promising, especially when we consider the fact that parallelization was applied only in the phase of global optimization. Satisfactory modelling of complex spatial processes, like atmospheric or groundwater pollution proliferation leads to problems of discriminating between models of distributed parameter systems described by systems of PDE's. In that case, the level of computational complexity is much higher owing to the necessity of multiply solving model equations on relatively dense, multidimensional grids. It seems to us that combining parallelization of solving PDE's describing models with parallelization of the global optimization process should allow us for a wider application of discrimination techniques based on optimum experimental design in fields such as pollution forecast or diagnosis of complex industrial processes.

Acknowledgement. This research was supported by the Polish State Committee for Scientific Research under Grant No. 3 T11C 035 27.

References

1. Atkinson A. C., Donev A. N.: Optimum Experimental Designs. Clarendon Press, Oxford, (1992).
2. Atkinson A. C., Fedorov V. V.: The designs of experiments for discriminating between two rival models. Biometrica, **62(1)**, (1975), 57–70.
3. Capellos Ch., Bielski B.H.J: Kinetic Systems, Wiley, New York (1972).
4. Clusterix project home page: http://clusterix.pcz.pl
5. Fedorov V. V., Hackl P.: Model-Oriented Design of Experiments. Springer-Verlag, New York, (1997).
6. Kiefer J., Wolfowitz J.: Optimum Designs in Regression Problems. Annals Math. Statist., **30**, (1959), 271–294.
7. Korbicz J., Kościelny J.M., Kowalczuk Z., Cholewa W. (Eds.): Fault Diagnosis: Models, Artificial intelligence, Applications. Springer-Verlag, Berlin Heidelberg, (2004).
8. Kuczewski B.: Discrimination between several multiresponse dynamic models using T-optimum experimental designs. Proc. of 10th Int. Conf. Metods and Models in Automation and Robotics, MMAR 2004, Szczecin, Poland, **1**, (2004), 619–624.
9. Kuczewski B., Patan M., Uciński D.: Discrimination between models of distributed parameter systems using T-optimum experimental design. In: Wyrzykowski R. et al. (Eds.): Parallel Processing and Applied Mathematics, LNCS 3019, Springer-Verlag, Berlin Heidelberg, (2004), 762 – 769.
10. Pacheco P. S.: Programming parallel with MPI. Morgan Kaufmann, San Francisco, (1997).
11. Patan M.: Optimal Observation Strategies for Parameter Estimation of Distributed Systems. University of Zielona Góra Press, Zielona Góra, (2004).
12. Patan M., Uciński D.: Robust activation strategy of scanning sensors via sequential design in parameter estimation of distributed systems. In: Wyrzykowski R. et al. (Eds.): Parallel Processing and Applied Mathematics, LNCS 3019, Springer-Verlag, Berlin Heidelberg, (2004), 770 – 778.
13. Rafajłowicz E.: Alghorithms of experimental design with implementations in MATHEMATICA. Academic Publishing Office PLJ, Warsaw, (1996) (in Polish).

14. Schittkowski K.: Numerical Data Fitting in Dynamical Systems - A Practical Introduction with Applications and Software. Kluwer Academic Publishers, Dordrecht, (2002).
15. Uciński D.: Optimal Measurement Methods for Distributed Parameter System Identification. CRC Press, Boca Raton, (2005).
16. Uciński D., Bogacka B.: T-optimum designs for discrimination between two multiresponse dynamic models. Journal of the Royal Statistical Society, B, **67(1)**, (2005), 3–18.
17. Walter, É., Pronzato L.: Identification of Parametric Models from Experimental Data. Springer-Verlag, Berlin, (1997).
18. Wyrzykowski R., Meyer N., Stroiński M., CLUSTERIX: National Cluster of Linux Systems. Proc. 2nd European Across Grids 2004 Conf., Nicosia, Cyprus, (2004) (available on-line: http://grid.ucy.ac.cy/axgrids04/AxGrids/index.html, http://clusterix.pcz.pl/project/publications/acrossGrids2004.pdf)

Real Terrain Visualisation with a Distributed PC-Cluster*

Jacek Lebiedź, Krzysztof Mieloszyk, and Bogdan Wiszniewski

Gdańsk University of Technology,
Faculty of Electronics, Telecommunications and Informatics,
ul. Narutowicza 11/12, 80-952, Gdańsk
{jacekl, krzymi, bowisz}@eti.pg.gda.pl

Abstract. Results of the ongoing project[1] , aimed at parallel interactive visualisation of spatial data coming from GIS system are presented. Application Vis3D being developed by the authors is a product of one of the demonstration tasks of the CLUSTERIX (National Cluster of Linux Systems) project, currently in its final stage. Major characteristics of the application is the potential for 3D immersion of emergency teams in a virtual scene for Crisis Information Management Software (CIMS) using real terrain data in a real-time.

1 Introduction

Virtual imagery using 3D is increasingly getting attention from various scientific communities, as the perspectives of getting out of Internet extremely high computational power offered by the grid technology, capable of exploring vast resources of scientific data, also in the Internet, are getting closer. High performance applications of particular interest are related to interactive visualisation, simulation steering and real-time collaboration. One example are flight simulators combining real cockpit and flight controls operated by human users with computer generated sky, weather, runways and terrain [4]. Natural interaction using 3D computer graphics involves three classes of systems: home entertainment with computer and video games, that alone were able to create a market worth of billions of dollars worldwide, experience based systems for training simulations, especially Distributed Mission Training (DMT) systems for the military and civil sectors [4], or the most recent Crisis Information Management Software (CIMS) for the emerging application area known as homeland security, and testbeds for novel technologies in a relatively rich environment before they are ready for the real world.

More often, such applications can offer *immersion* by superimposing virtual images on real objects and making users real *participants* of a dynamic scene, who

* This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098 "CLUSTERIX - National Cluster of Linux Systems".

[1] This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 349–356, 2006.

can look around, over and under the objects being displayed in three dimensions. The trend is evident in computer games [2], and will certainly boost expectations of users of other systems with an interactive 3D graphics capability. Although their visualised structures may be extremely complex, most games deal with artificial data. On the other hand, non-entertainment applications implementing interactive 3D graphics, may use the same engines, physics and architectures as the latter, but have to deal with real data. Two problems emerge:

- vast stores of distributed data must be shared, disseminated over some (quite often long distances) and processed on time to enable on-line collaboration of participants;
- data at different locations may be in incompatible formats, inconsistent, incomplete, mixed with irrelevant ones, and otherwise exhibiting low quality for high performance processing.

Interactive visualisation of urban terrain based on spatial data provided by GIS is one of several pilot utility applications within the CLUSTERIX (National Cluster of Linux Systems) project [5]. The Vis3D application is targeted at the GIS system currently in use by the City of Gdansk, as a step towards expanding the existing municipal CIMS system by a virtual training capability for emergency teams.

2 Clusterix Project Context

The basic concept behind the CLUSTERIX project is to develop a distributed meta-cluster based on the Polish Optical Network PIONIER [5]. Local clusters located at several geographically separated supercomputer centres across Poland, connected dynamically by a high-performance network provided by PIONIER, will be able to deliver advanced services integrated into one coherent system. While the major effort in the CLUSTERIX project is on development of a middleware capable of providing quality of service with regard to management and monitoring of resources and classic high-performance computing applications, some other types of applications are also attempted to explore the potential brought by this new grid architecture. One of these applications is Vis3D for

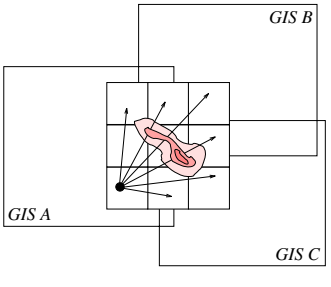


Fig. 1. Interactive visualisation of terrain data

interactive visualisation of real terrain data based on the content of a typical GIS system. The idea of this application is outlined in Figure 1.

An observer moving on or above a terrain indicated in Figure 1 as a grid may change its position and orientation to view various sectors of a dynamic scene from different distances and angles of inclination. Position changes are *dynamic* and *unpredictable* from the system point of view, as they reflect arbotrary decisions of an independent human operator, e.g. a pilot of a helicopter flight simulator, or a driver of a simulated car. The visualised terrain may consist of contingent geographic areas, described by spatial data coming from several GISes. This may be the case when flying over contengent regions of a country, with separate administrative units maintaining their own geographical databases, or driving through agglomerated municipal areas.

Owing to the physical distribution of local clusters across Poland, and the planned meta-cluster capability of the CLUSTERIX system, it is conceivable to develop a 3D visualisation system for the entire country. This ambitious goal, however, has to be approached in steps – of which the first one is to solve the problem of an effective cluster-based visualisation of spatial data coming from a single GIS. This is the case of the Vis3D application described in this paper.

3 Real Terrain Visualisation

Exact visualisation of a real terrain requires access to high quality spatial data. A common assumption is that users may want all visually recognisable objects and their details to be present in the scene. Unfortunately, data provided by the supporting GIS is often missing or incomplete. It can be seen in Figure 2 showing a real photo of some urban terrain close to GUT campus, and its view rendered by Vis3D based on the actual GIS content for this part of the city.

Note that the parking bay along the street has not been visualised, while a church-like object behind the line of buildings has been visualised as a block-shaped object. Such problems result from deliberate elimination of information by GIS administrators, who may consider some information redundant (chimneys, windows, roof types), or irrelevant (species of trees, road markings). In such

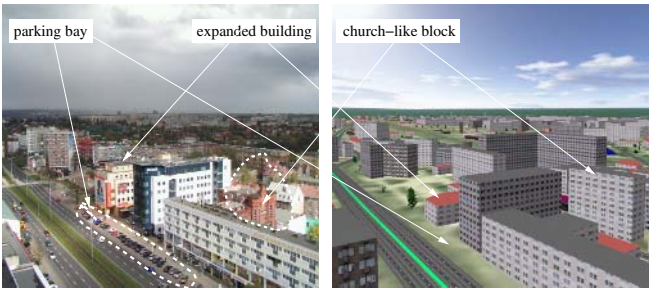


Fig. 2. Real and rendered views of the same terrain

cases all missing information must be generated automatically using the existing data. Special heuristics had to be developed in the project to overcome this problem [3]. Of course no heuristics can be 100% accurate, so manual correction (edition) is often necessary, particularly for unique objects like churches.

Another problem with GIS data may be their inconsistency. This is because GIS content is maintained and continuously updated to reflect the current state of the city structure. So far there are no mechanisms for automatic verification of consistency of updates introduced to the database by various agencies in a city, so in most cases municipalities must rely on their officers and bureaucratic procedures rather then specialised tools; note in Figure 2 the apparently obsolete data concerning the height of a building recently expanded, and incorrect dimensions of the church-like object, visualised (due to the lack of other specific information) as a block of a similar height. In order to overcome such problems we have decided not to implement dynamic acquisition of GIS data in parallel to the visualisation process and developed a separate terrain visualisation model, optimised additionally for better performance of visualisation algorithms [3]. Creation of the model is off-line with a special editing tool, on the basis of available GIS data, which are checked and corrected when necessary. One disadvantage of this approach is that further changes of GIS data (e.g. new buildings erected or streets reconstructed) require updates of the visualisation model. This however may be eliminated in the future, as GIS technology improves and develops more efficient mechanisms for internal model checking.

A need for photo-realistic image generation poses another problem for the scene complexity, related to the fact that human observers cannot accept identical oval trees, still river waters, simple box-shaped buildings, etc. Therefore it is important for the terrain model to balance properly complexity of the scene and reality of its view. Photo-realistic visualisation of a real terrain in an interactive 3D graphics application is further complicated by a dynamism of an observer moving freely along arbitrary trajectories in a cube, of which a visualised terrain is just a floor. An observer may shift on the terrain as well as fly over it, so scene complexity changes dynamically. The level of visual details of each object is selected in general on the basis of its distance from the observer, viewing direction, its size and form. The scene may also contain moving objects, which can add further to the complexity of a scene view. These may be "fixed" objects that exhibit motion, like waves, flags, or trees, as well as cars, planes, or trains that move across the scene. On top of that, some of the moving objects can be independent of the visualisation application, in the sense that they may have their autonomous operators forcing them to behave unpredictably (as in the case of distributed interactive simulation systems [4]).

4 Solution Model

By separating a process of *terrain modelling* (generation of visualisation data from spatial data retrieved from GIS) from *image generation* (of the scene view), the GIS data quality problem indicated before could be overcome. Terrain

modelling does not depend on observer's position, so it suffices to perform it only once before the actual visualisation. Image generation in turn is a cyclic process and creates a scene view in a continuous fashion. Although the terrain model is optimised for better performance of rendering procedures used for implementing the image generation process, the overall execution time of the latter is limited by animation smoothness, required to preserve realism of a view perceived by an observer moving in real-time. A solution adopted to achieve that in the reported project uses a concept of image simplification with regard to the distance of an observer from various parts of a scene.

In graphics systems with relatively low computational power, e.g., a workstation, it is reasonable to calculate various levels of detail for each object in a scene in an off-line mode before image generation. The off-line approach requires, however, special data structures for describing these levels within the terrain model. For graphics system with high computational power, e.g. a cluster, the level of detail can be calculated in real time, along with the image generation. This on-line approach does not require any additional structures in a terrain model, and allows for adding moving objects during interactive animation.

Clustering of workstations for interactive rendering tasks is not a new idea [1]. It was first used to drive large tiled displays and next shifted towards load-balancing of rendering tasks for effective utilisation of the graphics resources available in a cluster. The current trend is to make scalable cluster-based graphics a Web-based service, available to remote workstations.

This is also the aim of the project reported here. However, a significant difference is that the class of clusters used in the CLUSTERIX project may have generally poor graphics capability. Therefore, the Vis3D architecture (see Figure 3) distinguishes a specialised graphics workstation for rendering the scene, and a cluster for processing the terrain model data using a technique of scene simplification driven by movements of the observer. This leads to a two-step approach: terrain modelling regardless of the scene complexity, and parallelised scene optimisation, outlined in Figure 4.

By default, a terrain sector is represented by the visualisation model data with a high detail-level landscape. Therefore the visualisation station can render each sector using its local graphics capability (as pointed by the upper arrow in

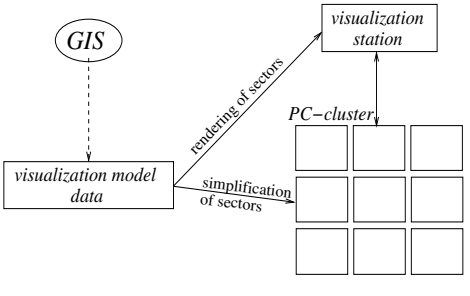


Fig. 3. Basic architecture of the Vis3D application

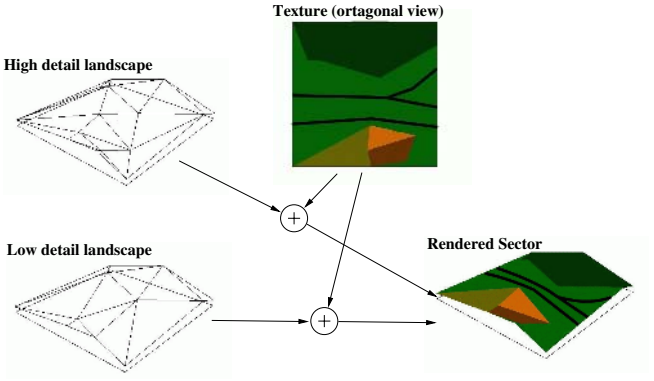


Fig. 4. Sector rendering

Figure 3). Scene optimisation, involving elimination of excessive details (in accordance to the distance from the observer) results in sectors with low detail-level landscapes. From the standpoint of the visualisation station, terrain modelling does not change, except that computation costs incurred on rendering simplified sectors are much lower. Simplification of sectors is performed by the cluster (as indicated in Figure 3 with a lower arrow), delivering next the reduced sector data to the visualisation station for rendering.

4.1 Terrain Modelling

Landscapes with natural and urban elements can make very complicated and irregular structures. The model of terrain developed for Vis3D provides a quick access to data on the terrain being actually within the sight of the observer. In order to enable effective access to detailed data describing the vicinity of the observer the area is divided into a regular grid of sectors. Data allocated to each sector contain information about its location on a scene, geometric complexity required later on for its optimisation, and indexes of textures needed for rendering it. The most important information concerns landscape and its elements, coded as geometric primitives. Different components of the landscape (like a city square or a sporting field) are described by the relevant texture, and parameters defining the properties of its surface. Each area is represented as a group of

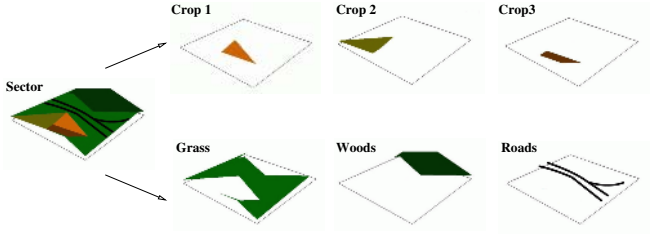


Fig. 5. Layers of the terrain model

primitives of different types, which allow for very detailed rendering and precise definition of interactions with the observer (see Figure 5). Dynamic creation of successive levels of detail uses data transformed accordingly to the landscape of a particular sector, and texture being generated from detailed data representing its orthogonal view put on it, as shown in Figure 4. The landscape of each sector is stripped off artificial elements, like roads or railways, as well as elements of the terrain, like ponds or meadows. It allows to reduce landscape details to the minimum and to speed up the process of sector simplification without compromising on realism – owing to the texture individually selected for each sector and rendered after sector simplification.

4.2 Parallelised Scene Optimisation

Visibility of a real terrain may reach as far as several tens of kilometres, depending on the altitude. Visualisation of the area with the same level of detail everywhere prevents from rendering a scene in a realistically short time, because of the large volume of data, implying unnecessarily high accuracy of the scene view. Sectors being close to the observer should be rendered in a detailed form, while the distant ones may be rendered in a much simpler form. Therefore, the observer's altitude and distance are decisive factors for determining the required level of detail. The idea is outlined in Figure 6.

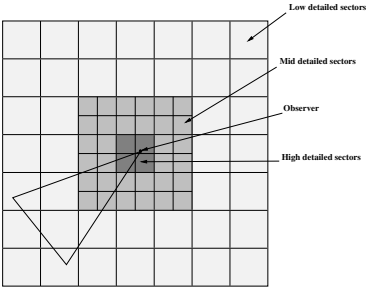


Fig. 6. Distance driven optimisation of scene sectors

The change of levels of detail of the sector occurs with a frequency not less than a time required to dislocate the observer for a distance equal to the actual size of the sector. Closer sectors require relatively less calculation, as less details have to be eliminated from the view, while more distant ones require more calculations to reduce excessive details. Individual load of processors simplifying the scene view at their respective sectors depends on the current position of the observer. Load balancing is possible then by running remote sector simplification tasks on nodes processing close sectors of the scene. Owing to the compact form of the visualisation model described before – about 0.5-1.0 MB for the area of 10 by 10 kilometres, compared to 1 GB of RAM and 20 GB of hard disk space at each cluster node – the database replication (usually posing a problem in cluster based rendering of images) is marginal, and each node may have locally a

complete set of terrain data for the entire area. Such a parallelisation model has two important advantages. Firstly, no frame by frame animation is necessary, as in the classic animation paradigm. Elements of a dynamic scene are calculated continuously, and all relevant sectors are sent in a just-in-time fashion to the visual station for final rendering. Secondly, as the entire scene is calculated simultaneously, i.e., sectors in and out of the observer's site alike, panoramic visualisation to exercise more interactive immersion in a virtual scene is possible just by connecting more graphics workstations to the cluster.

5 Conclusions

Vis3D is a first step towards a 3D graphics application for visualising real terrain based on distributed GIS data, using the meta-cluster capability of the CLUSTERIX system. Such an application may provide in a near future a Web-based service for various systems incorporating interactive immersion in 3D virtual environments for scalable training and simulation exercises. Of particular interest is further development of tools for dynamic acquisition of spatial data from GIS to build-up and update the visualisation model data, as well as techniques for global management of a dynamic scene consisting of data processed by several local clusters distributed over a larger territory.

References

1. G. Hunphreys, M. Houston, R. Ng, R. Frank, S. Ahern, and P.D. Kirchner. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Transact. Graph.*, 21(3), July 2002.
2. id Software. Quake III: Arena. www.idsoftware.com/games, 2002.
3. J. Lebiedź and K. Mieloszyk. Real terrain visualization on the base of GIS data. In *Proc. Int. Workshop on Intelligent Media Technology for Communicative Intelligence IMTCI*, Warsaw, Poland, Sept. 13-14 2004. LNCS, Springer Verlag, 2005, (to appear).
4. K. Mieloszyk and B. Wiszniewski. Component based flight simulation in DIS systems. In Z. Juhasz, P. Kacsuk, and D. Kranzlmuller, editors, *Distributed and parallel systems: cluster and grid computing*, pages 173–181, DAPSYS 2004, Budapest, Hungary, Sept. 19-22 2004. Springer Science.
5. R. Wyrzykowski, N. Meyer, and M. Stroiński. CLUSTERIX: National Cluster of Linux Systems. www.clusterix.pl, 2002.

Service Oriented Architecture for Risk Assessment of Natural Disasters

Martin Maliska, Branislav Simo, Marek Ciglan,
Peter Slizik, and Ladislav Hluchy

Institute of Informatics, Slovak Academy of Sciences
{martin.maliska, branislav.simo, marek.ciglan, peter.slizik,
hluchy.ui}@savba.sk

Abstract. The new FP6 programme project Medigrid[1] is targeted in Global Change and Ecosystems sub-priority. This paper will discuss this project and will describe its primary aim — development of a framework for multi-risk assessment of natural disasters. We have considered several aspect and user requirements during the design of architecture. The service oriented architecture based on the Globus Toolkit 4 has been chosen. The data management services will be implemented on the top of the OGSA-DAI framework with several improvements and extension. We will use a workflow management system for integration purpose. A portal technology has been chosen as user interface.

1 Introduction

The occurrence of natural disasters poses a great threat to people's lives and their properties, not to mention a negative impact to economies of whole regions and countries. In order to make prevention, forecasting and mitigation of such disasters possible a lot of effort has been invested into research of phenomena like forest fires, floods, landslides and soil erosion and others. Currently, there are models available that are capable of computing various aspects of these hazards. However, the data, the models or the results of simulations are not easily available to the audience that could make proper use of them; resources are not available to computationally intensive models and so on.

We are working to overcome some of these problems in the Medigrid project by employing the grid technology. The grid technology will allow us to make the models and data accessible via internet in a secure manner for all partners and, possibly, other parties in the future. It will also allow us to exploit computing resources connected to the grid for execution of demanding simulation models. The aim of the project is to create a distributed framework for multi-risk assessment of natural disasters that will integrate models, which have been developed in previous projects funded by European Commission. These include models for

[1] This work is supported by the project Medigrid EU RTD SustDev FP6-004044, NATO grant "Flood Forecasting Computed on Grid Infrastructures" EST.EAP.CLG 981032 and Slovak national project VEGA 2/3132/23.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 357–363, 2006.

simulation of forest fire behavior and effects, flood modeling and forecasting, landslides and soil erosion simulations. Also, a distributed repository with earth observation data, combined with field measurements is being created, which will provide data to all models using data format conversions when necessary. The entire system of models and data will be shaped further as a multi-risk assessment and decision support information platform.

2 User Requirements

An important aspect of simulation applications supported in the Medigrid project [6] is that some of them are closely bound to the Windows operating system while the others can be executed only on the Linux platform. This means that underlying grid infrastructure must be operable on heterogeneous resources. This is not an easy task as several standard grid data management components are operable on Linux systems only, e.g. current implementation of the GridFTP [7], the basic grid transportation mechanism, requires the Linux platform.

All of the Medigrid applications use geographical data in the computations. The content of geographical data changes rarely; those data are reused again and again by the simulations. Replication of such data sets is a necessity to increase their availability and to reduce access latencies. Some of the data sets can be made available to all partner organizations; however, there are some data files, which are proprietary and can be used only by members of certain organization due to the license agreement issues. The use of such data in the highly distributed environment, such as the Medigrid, is quite delicate and could be potentially dangerous. Much attention must be paid to the data security and data access policies. Owners of the data must then have absolute control over data access policies.

Many input data sources for the applications and many simulation results must be stored, maintained and accessed in the Medigrid system. To facilitate usage of the data managed in the Medigrid, rich metadata must be kept and made available to the users — to discover the data of their interest. Moreover, experimental results of a single simulation can be composed from multiple data files and must be locatable as a single result data set. We need to exploit the concept of the logical file collections within the metadata service.

3 Architecture

The software architecture of the project was designed according to the user requirements discussed in the previous section. Globus Toolkit 4 [2] with several extensions will be used as middleware. Every computational model will be wrapped to a web service. The overview of the architecture is shown on figure 1. The system will comprise of system services, user services and user interfaces discussed in separate subsections.

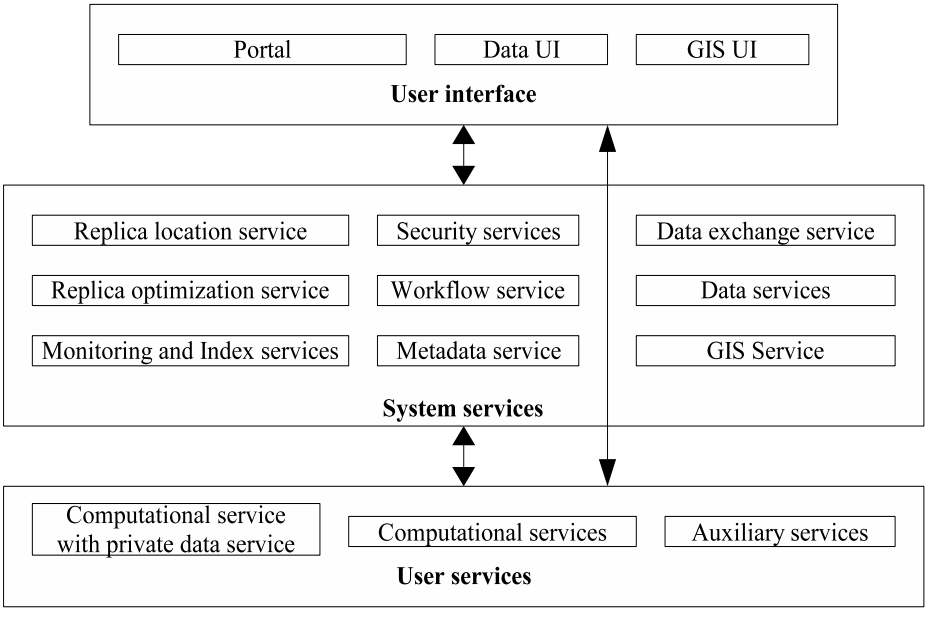


Fig. 1. The Medigrid Architecture

3.1 System Services

After the analysis of the available data management tools for the grid environment, we have identified the data access and integration framework [8] as a likely candidate for realization of the Medigrid data services. OGSA-DAI provides many features that are required by the Medigrid project; it supports GSI authentication and authorization [9], provides file manipulation activities, data transfer activities and allows definition of fine grained security policies. However there are several problems with the current state of the OGSA-DAI implementation. File system manipulation activities do not perform authentication and authorization checks. File reading activity reads whole file into memory before transfer. As the OGSA-DAI system is an easily extensible framework, we have solved those problems by implementing additional activities which overcome mentioned drawbacks. The data transfer speed is significantly lower than the speed achieved by the GridFTP. The OGSA-DAI wraps all of the transferred data pieces in XML documents. Furthermore, it uses Axis as an underlying data transfer mechanism, which uses the SOAP protocol [10], which envelopes transferred data by additional XML data. This produces a huge XML overhead over transferred data that actually slows down the data delivery process. This problem will be resolved by implementing specialized delivery activity for the OGSA-DAI system.

To allow access to the specific data resource in distributed environment, the user/grid service must first know the physical location of the data object. In the Medigrid project we plan to use the Replica Location Service (RLS) — a

simple registry that provides a mechanism for recording the existence of replicas and discovering them. RLS will maintain a mapping between logical file names (LFNs) and physical file names (PFNs).

Data for the simulation models need to be shared between geographically distributed computational facilities. All the models use the spatial data for computation of the simulations; those spatial data are often reused by different simulation jobs. The concept of the data replication can be used to increase the data availability and thus increase the efficiency of the whole system. Under the term of data replication, we understand creation of multiple copies of a single data source across multiple grid nodes. The problem of replica optimization can be divided into two subproblems. Short-term optimization aims at minimizing data transfer time, given a requesting grid node and a logical filename. Long-term optimization concerns global reduction of network traffic in the grid by deciding which files should be replicated (or deleted) and where to place those replicas.

For the purpose of the Medigrid project, we need a sophisticated metadata service that supports user defined attributes and the concept of logical file collections. Under the term of logical file collections, we understand a virtual directory that can contain multiple logical files and/or multiple logical collections. We need this functionality to encapsulate multiple files in a single collection to grasp the logical relation of those files. For example, the simulation result can be composed of several different files and we need to retain the information that those files are members of a single result set. The Metadata Catalog Service (MCS), developed under the GriPhyn project [11], was identified as a suitable metadata service for the Medigrid.

The GIS operations in the Medigrid project will be provided by the open source server MapServer [13]. It runs as a CGI executable in any CGI-enabled web server (e.g.,Apache). As inputs, it directly supports many vector and raster data formats. The output can be either an ordinary JPEG or PNG image, or an HTML page. The HTML output is fully customizable, due to MapServer's support of HTML templates. The output maps consist of multiple layers, which can be switched on and off, according to user's actual needs. The system also includes MapScript, an interface providing access to the MapServer from some well-known programming languages. MapServer provides simple means for navigation: moving to all directions, zooming in and out, recentering of the map, showing and hiding layers, and adding a legend. A slight disadvantage of MapServer is its reduced level of interactivity, due to the HTTP's request-response communication model. However, this disadvantage can be overriden, though not perfectly, with the use of JavaScript. In our project, we plan to use MapServer as the main visualization engine for creating and displaying 2D maps. While the MapServer engine will provide us with a complete set of server functions, we will have the task of making the server work with our data, taking care of data conversions, and providing a nice and useful user interface.

The Medigrid is typical project where the workflow management system is a perfect solution for integration of web services. User services are mainly computational services with very close subject of interest, so the usual situation

is that results from one computation (service) can be used as input data for another computation. Although we have had experiences from the CrossGrid project [12] [1], we decided to use workflow enactor FreeFluo [3] developed in the MyGRID project [4]. A workflow will be described with XScufl [5] language supported by FreeFluo enactor.

Security is one of the key issues in our project. We had established a private certification authority for project members. Current implementation of Globus toolkit 4 Authentication and authorization framework is going to be used. It contains a SAML callout [16] authorization module which enables outsourcing of authorization decisions to an external authorization service. We are now deciding whether we will use Privilege and Role Management Infrastructure Standards Validation [15] or Virtual organization membership service [14] as authorization service.

3.2 User Services

User services are the core of the Medigrid project, so they have to be designed precisely. As it can be seen on Figure 2, a user service will consist of several components. To overtake the problem of potential overload when multiple requests for computation are sent to a service, we have decided to develop the Job Management component. This component will ensure that only one computation will run at time and other requests will be queued in a priority queue. Access to data oriented services (like the metadata catalogue, replica services and data services) will be wrapped by the Data access layer. Because the computational problems are not simple and they could not be solved in real-time, we will create the Notification management component that will inform the registered clients about the state of their computations. User services for users which don't have public

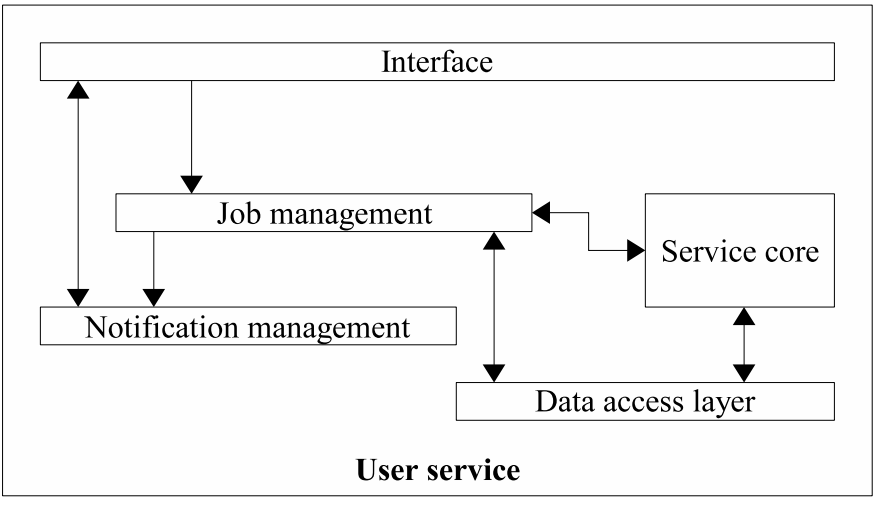


Fig. 2. User service components

data will have their private data services with fully restricted access. When the results are ready to be published, they will be transferred to public data servers. In our project, we will have these types of computational services:

1. Meteorological services
2. Hydrological services
3. Hydraulics services
4. Fire propagation services
5. Land slides services

The auxiliary services will also play a very important role in the project. One of their tasks is the conversion of datasets when result from one computational service will be used as an input to another computational service. Another important type of auxiliary services will be visualization services. The purpose of the visualization services is to present the results of simulations in an easy-comprehensible way. As the output data and relationships among them are usually quite complex, graphical representation can be an essential key to understand them. The MediGrid visualization service will support both 2D maps and 3D presentations of the endangered areas. We have proposed modular architecture of the visualization. There will be a central visualization engine, common for all processes. It will work with its internal data formats. The engine will be encapsulated by input and output modules, which will convert the input data formats into the internal formats, or the internal formats into the output formats, respectively. Such an architecture will allow the developers to add further input and output formats easily.

3.3 User Interface

The Medigrid user interface will be based on JSR 168 compliant portlets. We are going to use the Gridsphere framework. Each type of user services will have its own portlet. The system services will also have their user interfaces. To fulfill the needs of integration of the computational services, several portlets for workflow construction, execution and monitoring will be created. The user interfaces for data management services will play also an important role. They will allow user to manipulate the datasets, search and update metadata, affect the replica optimization and locate replicas. The user will be able to monitor the usage and the effectivity of the Medigrid framework throught portlets for resources monitoring.

A special UI will be used for operations on large datasets (uploading to the system and downloading from system). The main goal of this UI will be elimination of potential overload of portal when manipulating with large datasets. One public data service will be reserved for this purpose as a mediator among private data services and user interface. This user interface will be a java applet runnable directly from the portal.

4 Conclusion

In this paper we presented the proposed architecture of the Medigrid framework. The proposal has been created according to user requirements on security, data management and integration. We were faced to several problems like requirement of private data server with possibility of publishing of results, integration of computational models with different formats of input and output data, management of multiple requests for computation especially on Windows platform, etc. Some of them where already solved and the solution was described. But there are few problems which will need a lot of effort. In the data management, we will need to find out the best algorithms for replica optimization and extend current OGSA-DAI implementation to improve the data transfer speed which is crucial for large datasets. The proposal of the Medigrid project aims to create generic risk management framework. Therefore we need to design a user interface capable of fullfiling the requirements on risk analysis and management operations.

References

1. Hluchy L., Astalos J., Dobrucky M., Habala O., Simo B., Tran V.D.: Flood Forecasting in a Grid Environment. In: Proc. of 5-th Intl. Conf. on Parallel Processing and Applied Mathematics PPAM'2003, R.Wyrzykowski et.al. eds., 2004, LNCS 3019, Springer-Verlag, pp. 831-839
2. Globus toolkit 4 homepage, http://www.globus.org/GT4/
3. FreeFluo workflow enactor, http://freefluo.sourceforge.net/
4. MyGRID project homepage, http://www.mygrid.org.uk
5. XScufl language reference, http://taverna.sourceforge.net/docs/xscuflspecification.html
6. Medigrid project, http://www.vfdb.de/riskcon/programme/medigrid.htm
7. Protocol Extensions to FTP for the Grid, http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf
8. Open Grid Services Architecture Data Access and Integration, http://www.ogsadai.org.uk/
9. Overview of the Grid Security Infrastructure, http://www-unix.globus.org/security/overview.html
10. Soap specification, http://www.w3.org/TR/soap/
11. Grid physics network, http://www.griphyn.org/
12. CrossGrid Exploitation website, http://www.crossgrid.org
13. MapServer homepage, http://mapserver.gis.umn.edu/
14. Virtual Organization Membership Service, http://hep-project-grid-scg.web.cern.ch/hep-project-grid-scg/voms.html
15. Privilege and Role Management Infrastructure Standards Validation, http://www.permis.org
16. Security Assertion Markup Language, http://xml.coverpages.org/saml.html

Porting Thermomechanical Applications to Grid Environment*

Tomasz Olas and Roman Wyrzykowski

Czestochowa University of Technology,
Institute of Computer & Information Sciences,
Dabrowskiego 73, 42-200 Czestochowa, Poland
{olas, roman}@icis.pcz.pl
http://icis.pcz.pl

Abstract. *NuscaS* is an object-oriented package designed at Czestochowa University of Technology to investigate thermomechanical phenomena using FEM modeling. Its functionality includes also implementation on clusters. This paper presents our experience in adaptation of the *NuscaS* system to the CLUSTERIX environment, which is a national grid infrastructure with 12 local Linux PC-clusters connected by the Polish Optical Network PIONIER. Using the MPICH-G2 middleware, the proposed solution allows for running tasks across several local clusters as meta-applications. The performance results of numerical experiments with FEM modeling of casting solidification confirm that CLUSTERIX can be an efficient platform for running numerical meta-applications. However, harnessing its computing power needs to take into account the hierarchical architecture of the infrastructure.

1 Introduction

CLUSTERIX (National Cluster of Linux Systems) is a truly distributed national computing infrastructure with 12 sites (local Linux PC-clusters) located across Poland [1, 16]. CLUSTERIX sites are connected by the Polish Optical Network PIONIER providing the dedicated 1 Gb/s bandwidth. Although the CLUSTERIX grid offers potentially large performance, the key questions facing computational scientists is how to effectively adapt their applications to such a complex and heterogeneous architecture.

Finite element method (FEM) is a powerful tool for simulating different phenomena in various areas of science and technology [13]. However, many applications of this method have too large computational or memory costs for a sequential implementations, to be useful in practice. Parallel computing allows this bottleneck to be overpassed [13, 14]. In particular, an object-oriented environment for the parallel FEM modeling on clusters, called *ParallelNuscaS*, was developed at Czestochowa University of Technology [15]. This environment is dedicated to modeling of such thermomechanical phenomena described by

* This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098 "CLUSTERIX - National Cluster of Linux Systems".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 364–372, 2006.

time dependent PDEs as heat transfer, solidification, stresses, damages of materials, etc.

Efficiently harnessing computing power of grids requires the ability to match requirements and characteristics of applications with grid resources. Challenges in developing grid-enabling applications lie primarily [3] in the high degree of system heterogeneity and dynamic behavior of grid environments. For example, communication between computers of the same site, connected by a high bandwidth and low latency local network, is much faster than communication between nodes of different sites provided by a wide area network characterized by a much high latency. What is also important, computers may reside in different administrative domains with different access control policies, run different software, etc.

It makes programming HPC applications on grids a challenging problem. An important step in this direction is emergence of scientific-application-oriented grid middleware, such as MPICH-G2 [4] that implement the well-established MPI standard on top of grid infrastructure based on the Globus Toolkit [2]. The MPICH-G2 environment significantly spares computational scientists from low-level details about communication handling, network topology, and resource management. However, in spite of these achievements, the development of efficient algorithms for large-scale computational problems, like FEM modeling, that can exploit grid architectures efficiently still remains an exceptionally challenging issue.

This paper presents our experience in adaptation of the *NuscaS* system to the CLUSTERIX grid environment. The proposed solution allows for running tasks across several local clusters as meta-applications, using the MPICH-G2 middleware.

The paper is organized as follows. In Section 2, we shortly present the architecture of the CLUSTERIX grid. The concept and main features of the *ParallelNuscaS* system are described in Section 3. How thermomechanical meta-applications based on ParallelNuscaS are adapted to the CLUSTERIX environment using the MPICH-G2 tool, it is presented in Section 4. Section 5 describes a model problem related to the FEM modeling of castings solidification, which is used for testing the ParallelNuscaS software in the CLUSTERIX environment. Performance results of these tests are presented in Section 6, for both the single-site and cross-site runs. Conclusions are given in Section 7.

2 Architecture of CLUSTERIX Grid

CLUSTERIX is a distributed PC-cluster (or meta-cluster) with 12 local Linux clusters in the core [1], located in independent centers across Poland. They are interconnected via dedicated 1 Gb/s channels provided by the Polish Optical Network PIONIER. At this moment, the CLUSTERIX core includes 254 Intel Itanium2 processors (1.3 GHz, 3 MB cache) in 127 two-way SMP nodes. Each of nodes is equipped with 4 GB or 8 GB RAM, and 73 GB or 146 GB SCSI HDD. Two Gbits-per-second VLANs are used to improve management of network traffic

in local clusters: communication VLAN, using Gigabit Ethernet or Inifiniband, is dedicated to support nodes messages exchange, while connection to the PIONIER backbone is provided through a Gigabit Ethernet L2/L3 coupling switch.

Selected 32-bit machines are dedicated to management of local clusters and the entire infrastructure. While user tasks are allowed to be executed only on computational nodes, each local cluster is equipped with an access node where the Globus Toolkit and local batch system are running. An important element of the CLUSTERIX core is the Data Storage System managed by the CLUSTERIX Data Management System - CDMS [6]. Before execution of an application, input data are fetched from storage elements and transferred to access nodes; after the execution output data are returned from access nodes to storage elements. Currently each storage element is equipped with 2 TB HDD.

The CLUSTERIX middleware is developed as Open Source, and is based on the Globus Toolkit 2.4 and Web Services. The use of Web Services makes the created software easier to reuse, and allows for interoperability with other grid systems on the service level. The important feature of this middleware is ability to manage infrastructure with dynamic changing configuration. In particular, new clusters may be attached to the CLUSTERIX core dynamically. The connection of new clusters to the CLUSTERIX core opens possibilities to access a shared environment with the extraordinary computational power. For example, an experimental installation with 802 Itanium2 CPUs offering a peak performance of about 4,5 TFLOPs has been created.

3 *ParallelNuscaS* System

NuscaS is an object-oriented package for the FEM modeling, developed at Czestochowa University of Technology [12]. *ParallelNuscaS* is an extension of the sequential *NuscaS* software. It is one of pilot applications adapted for the execution in the CLUSTERIX environment.

For uniprocessor nodes, the message-passing model is used in *ParallelNuscaS*. In this case, a FEM mesh is decomposed [7] into a number of submeshes (subdomains), which are then processed concurrently over different processors. We use iterative methods for solving large linear systems with sparse matrices, which are the result of FEM discretization [8]. The kernel of these methods is the sparse matrix-vector multiplication. When implementing this operation in parallel, the overlapping of computation and communication is exploited to reduce the algorithm execution time.

Because both uniprocessor and SMP nodes are considered, we investigate not only the message-passing model of parallel programming, but the hybrid model as well [10]. This model is a mixture of multithreading inside SMP nodes and message passing between them.

4 Meta-applications in CLUSTERIX

According to the project goals, CLUSTERIX is used both for running high-throughput computing applications, as well as large-scale distributed applications

that require parallel use of resources of one or more local clusters. For the experimental verification of the project assumptions and results, selected end-user's applications are deployed to be executed on the CLUSTERIX grid. It is clear that delivering end-user applications able to use distributed resources efficiently will in the end decide on success or failure of computational grids.

These applications fall into two different categories:

- Applications dedicated to run on a single local cluster (single-site execution); this restriction is related only to a single instance of a given application, while different instances of this application can be executed on different local clusters.
- Meta-applications intended to be executed on more than one local cluster (cross-site execution); they will take into account the heterogeneity of the meta-cluster in respect of both the computing power and network performance.

4.1 Running Tasks in CLUSTERIX Environment

The standard procedure for running tasks in the CLUSTERIX environment is illustrated in Fig.1. A user submit its task through the Web portal, which generates a special file with the task description in the XML format. This file is sent to the GridLab Resource Management System (GRMS), which chooses the best resources (one or several clusters) for the task execution, according to the task description, with respect to hardware and software. The task is then submitted using Globus services to batch systems of local clusters. The important step is also assignment of a virtual account that will be used for the task execution. The Virtual User's Account System (VUS) is responsible for performing this step.

After staging the executables and input files using CDMS, the task is executed on computational nodes of the chosen local clusters. CDMS is also responsible for copying the results after finishing the task. Finally, the user is notified

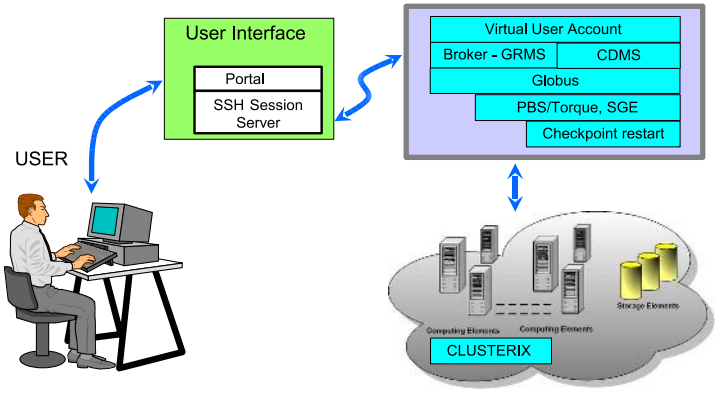


Fig. 1. Running tasks in CLUSTERIX environment

about the task termination through the Web portal (alternative way is to use e-mail or SMS).

4.2 Using MPICH-G2

In the CLUSTERIX project, the MPICH-G2 middleware [4] based on the Globus Toolkit is used as a grid-enabled implementation of the MPI standard. It allows for running multilevel parallel applications across many sites (local clusters). MPICH-G2 extends the MPICH software to use Globus-family services. To improve performance, we use MPICH-based vendor implementations of MPI in local clusters, e.g. MPICH-GM which allows for running MPICH over the Myrinet interconnect.

CLUSTERIX has a hierarchical architecture, with respect to both the memory access and communication. Inside an SMP node, data exchanges between processors are performed through shared memory. SMP nodes are grouped into local clusters, and communications inside them are implemented using such network protocols as Gigabit Ethernet, Myrinet, or Infiniband. They are characterized by high bandwidths and small latencies (especially Myrinet and Infiniband). Finally, local clusters connected by WANs are building blocks for the whole meta-cluster.

Taking into account the hierarchical architecture of the CLUSTERIX infrastructure, it is not a trivial task to adapt the existing applications for effective use in the meta-cluster. It requires parallelization on several levels corresponding to the meta-cluster architecture, taking into account the high level of heterogeneity in network performance between various subsystems of the meta-cluster (see Table 1). In particular, there is a quite complex problem of minimizing the influence of less efficient networking between local clusters on the efficiency of calculations.

Table 1. Hierarchical architecture of CLUSTERIX

	latency	bandwidth
local (vendor MPI)	104 μs	752 $\frac{Mb}{s}$
local (MPICH-G2)	124 μs	745 $\frac{Mb}{s}$
global (MPICH-G2)	10 ms	33 $\frac{Mb}{s}$

5 Testing Problem

The FEM modeling of castings solidification is chosen as a model problem for testing the *ParallelNuscaS* software on CLUSTERIX. This physical phenomenon is governed [12, 14] by a quasi-linear heat conduction equation

$$\nabla(\lambda \nabla T) + \dot{q} \;=\; c\rho \frac{\partial T}{\partial t}, \tag{1}$$

where T, t, λ, ρ, and c denotes respectively temperature, time, coefficient of thermal conductivity, density, and specific heat. This equation contains the term of heat source $\dot{q}$, which describes the rate of latent heat evolution:

$$\dot{q} \;=\; \rho_s L \frac{\partial f_s}{\partial t},$$

where L is latent heat, and f_s is solid phase fraction. The subscript s refers to the solid phase.

For solving equation (1), the so-called apparent heat capacity formulation is used [12]. The testing numerical problem corresponds to solidification of a casting made of the $Al - 2\%Cu$ alloy, in a metal mould.

In our tests, the problem geometry was meshed with 40613, 80401, 159613, 249925, 501001, and 750313 nodes, using triangular finite elements. Every mesh was decomposed using the Metis library [5] into suitable sub-domains. It should be noted that both in the sequential and parallel cases, the average from ten runs was used to estimate the sequential T_1 and parallel T_p execution times, where p is the number of processors. For each run, the time necessary to solve a system of linear equations using the conjugate gradient algorithm was measured.

6 Performance Results

6.1 Single-Site Performance

Fig.2 presents the performance results obtained for a single local cluster, located in Poznan Supercomputing and Networking Center. In spite of using only the Gigabit Ethernet, the results of these experiments are very good since the speedup and efficiency are almost ideal. For example, for the mesh with 501001 nodes, our parallel code gives speedup of $S_p = 9.72$ for $p = 10$ processors.

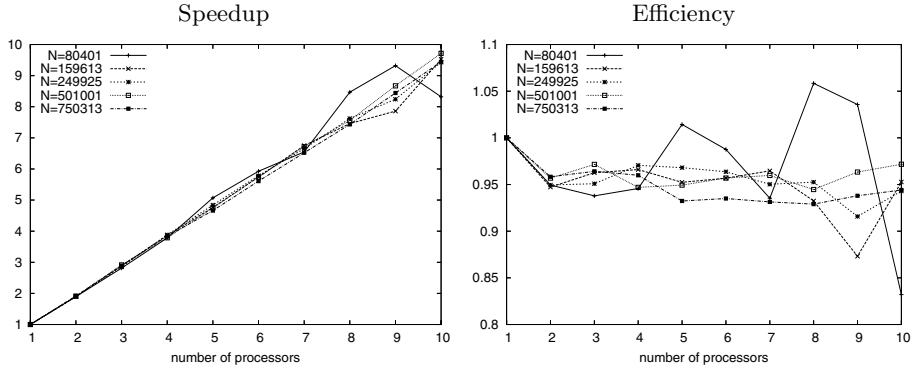


Fig. 2. Testing problem on a single cluster: speedup and efficiency for different mesh sizes versus number of processors

6.2 Cross-Site Performance

Fig. 3 presents the performance results obtained on two distant local clusters, located evenly in Poznan and Czestochowa. These results are rather promising.

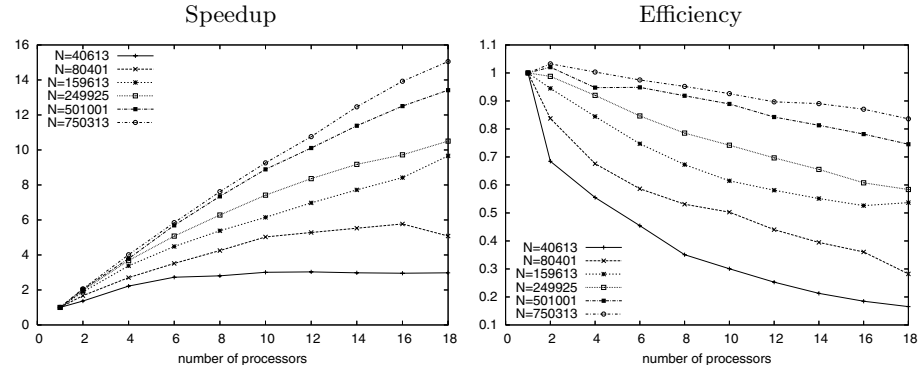


Fig. 3. Testing problem on two distant local clusters: speedup and efficiency for different mesh sizes versus number of processors

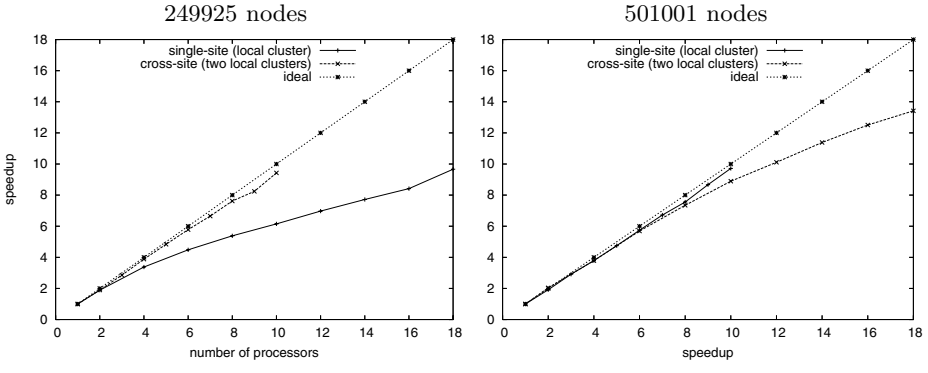


Fig. 4. Comparison of single-site and cross-site performance for meshes with 249925 and 501001 nodes

For example, the speedup for the mesh with 750313 nodes is $S_p = 15.05$, for $p = 18$ processors (9 from Poznan, and 9 from Czestochowa). However, for small meshes the cross-site communication overhead becomes more significant, and decreases the speedup. For example, for the mesh with 159613 nodes, the parallel runtime on 18 nodes is only 9.66 times smaller than the sequential runtime.

For achieving such results, the key points are to use a modified version of the conjugate gradient algorithm with only one synchronization point [8], as well as exploiting the overlapping of computation and communication when implementing the sparse matrix-vector multiplication in parallel.

For large meshes, the obtained values of speedup/efficiency are satisfactory for practical needs. At the same time, the execution of relatively small problems (e.g., with 40613 nodes) as meta-applications is not reasonable because of too large cross-site communication overheads. For such problems, the use of resources of a single local cluster is sufficient in practice.

Fig. 4 presents the comparison of the cross-site and single-site performance. This comparison shows a loss in speedup when more than one local cluster is used. This negative effect is decreasing with the growth of the mesh size.

7 Conclusions

This paper presents our experience in adaptation of the *NuscaS* FEM-dedicated package to the CLUSTERIX grid environment. Using the MPICH-G2 middleware, the proposed solution allows for running tasks across several local clusters as meta-applications. The performance results of numerical experiments with FEM modeling of castings solidification confirm that CLUSTERIX can be an efficient platform for running numerical meta-applications. However, harnessing its computing power needs to take into account the hierarchical architecture of the infrastructure, especially the heterogeneity in the network performance.

References

1. CLUSTERIX Project Homepage, http://www.clusterix.pl
2. Globus Toolkit Homepage, http://www.globus.org/toolkit
3. Dong, S., Karniadakis, G.E., Karonis, N.T.: Cross-site Computations on the TeraGrid. Computing in Science & Engineering **7**, **5** (2005) 14–23
4. Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing **63**, **5** (2003) 551–563
5. Karypis, G., and Kumar, V.: METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices, version 4.0, Tech. Rep., Univ. of MN, Dept. of Computer Sci. and Eng., 1998.
6. Kuczynski, K., Karczewski, K., Wyrzykowski, R.: CLUSTERIX Data Management System and Its Integration with Applications (this volume).
7. Lo, G., Saad, Y.: Iterative Solution of General Sparse Linear Systems on Clusters of Workstations, Tech. Rep. UMSI 96/117 & UM-IBM 96/24, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1996
8. Meisel, M., Meyer, A.: Hierarchically Preconditioned Parallel CG-Solvers with and without Coarse-Mesh Solvers Inside. SFB399-Preprint 97-20, Technische Universitat Chemnitz, 1997
9. Mirghani, B.Y., Tryby, M.E., Baessler, D.A., Karonis, N., Ranjthan, R.S., and Mahinthakumar, K.G.: Development and Performance Analysis of a Simulation-Optimization Framework on TeraGrid Linux Clusters. Proc. LCI International Conference on Linux Clusters: The HPC Revolution 2005, Chapel-Hill, NC, April 2005
10. Olas, T., Karczewski, K., Tomas, A., Wyrzykowski, R.: FEM Computations on Clusters Using Different Models of Parallel Programming. Lect. Notes in Comp. Sci. **2328** (2002) 170-182
11. Olas, T., Wyrzykowski, R., Tomas, A., Karczewski, K.: Performance Modeling of Parallel FEM Computations on Clusters. Lect. Notes in Comp. Sci. **3019** (2004) 189–200

12. Sczygiol, N.: Object-Oriented Analysis of the Numerical Modeling of Castings Solidification. Computer Assisted Mechanics and Engineering Sciences **8** (2001) 79–98
13. Topping, B.H., Khan, A.I.: Parallel Finite Element Computations. Saxe-Coburg Publications, 1996
14. Wyrzykowski, R., Sczygiol, N., Olas, T., Kaniewski, J.: Parallel Finite Element Modeling of Solidification Processes. Lect. Notes in Comp. Sci. **1557** (1999) 183–195
15. Wyrzykowski, R., Olas, T., and Sczygiol, N.: Object-Oriented Approach to Finite Element Modeling on Clusters. Lect. Notes in Comp. Sci. **1947** (2001) 250-257
16. Wyrzykowski, R., Meyer, N., Stroinski, M.: Concept and Implementation of CLUSTERIX: National Cluster of Linux Systems. Proc. LCI International Conference on Linux Clusters: The HPC Revolution 2005, Chapel-Hill, NC, April 2005

Parallel Implementation of Software Package for Modelling Bi–phase Gas–Particle Flows

Sebastian Pluta and Roman Wyrzykowski

Institute of Computer and Information Sciences,
Czestochowa University of Technology
{pluta, roman}@icis.pcz.pl

Abstract. This paper presents the concept and some details of implementation of a software package designed for modeling bi–phase gas–particle flows, using DEM and MP–PIC methods. In particular, this package allows for the parallel implementation of computations on shared-memory systems. The performance of the sequential and parallel versions of the package is investigated. A short characteristic of the testbed is included.

1 Introduction

Bi-phase gas-particle flows are applied in many industrial domains [1, 10], especially in chemical and processing engineering, coal combustion, powder production technology, environmental engineering, petrochemical industry, pharmaceutical industry and others. In this work, the combination of the Euler approach to the gas movement description and the Lagrange approach for the particle movements is used for modelling such flows. The most important numerical methods implementing the above-mentioned combination are DEM (Distinct Element Method) and MP – PIC (Multiphase Particle in Cell). The distinction between motionless cells, associated with the gas movement, and movable cells, associated with the particles, is the basic assumption of our computation methodology [1, 6, 7, 9, 10].

In the previous works [8, 9], we described an object oriented package which implements this methodology. This paper presents our experience in adaptation of the package to its parallel implementation on shared-memory architectures.

The paper is organized as follows. In Section 2, we shortly present the mathematical and numerical models used in simulation, while Section 3 describes the concept of the package which implements these models. How this package is adapted to the shared-memory model of parallel programming, it is presented in Section 4. Performance results of numerical experiments are shown in Section 5. Conclusions are given in Section 6.

2 Mathematical and Numerical Models

The mathematical model for the problem solved includes the following equation of progressive motion for a single grain located in a motionless cell associated with the gas movement, without interactions with other grains [7]:

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 373–379, 2006.

$$(m_p + a_p m_g)\frac{d^2 x_i}{dt^2} = \beta c_v V \left(U_{gi} - \frac{dx_i}{dt} \right) - F_{mgi} \tag{1}$$

where:
m_p – mass of particles,
m_g – mass of gas,
V – cell volume,
a_p – coefficient of gas uplift pressure,
c_v - concentration of particles in the cell,
U_g – velocity of gas,
β – a certain coefficient,
F_{mg} – gravitational force.

Interactions between particles are derived according to the "spring–dumper" model [6]. The important part of calculations of particles motion is detection of collisions between particles [9].

The model of average U_g is applied to determine the velocity of gas for the cuboid geometry of a pipe [1, 9].

3 Sequential Implementation

The choice of adequate data structures is of great importance for the efficient implementation of the model. The nature of the problem requires two separate

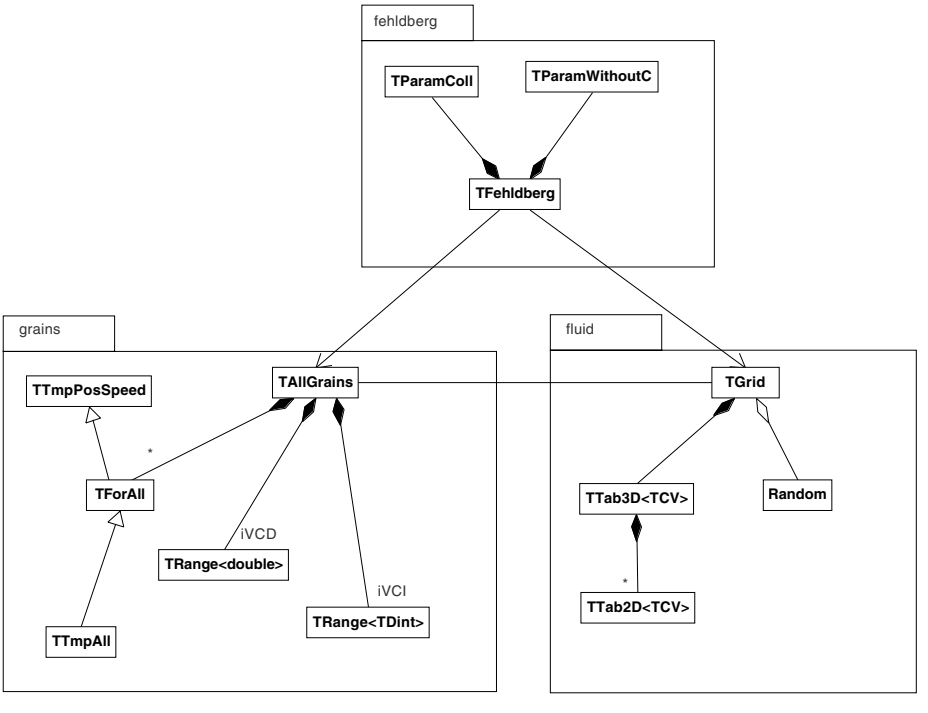


Fig. 1. Architecture of the software package for bi-phase simulation

structures - for gas and for particles of loose phase. As a result, the package implementation includes [8, 9] the following main modules (Fig.1):

- `fluid` - involving data structures for gas, and responsible for solving the gas motion according to avarage speed of gas in the pipe;
- `grains` – involving data structures for particles;
- `fehlberg` - responsible for solving systems of differential equations, using the Fehlberg method, in order to determine the motion of particles.

4 Parallel Implementation

To parallelize computations, we have decided to use the parallel programming model with shared memory. The global access to shared-memory data structures describing parameters of particles allows for eliminating a large communication overhead caused by a distributed-memory implementation. Among possible implementations of the shared-memory model, the *pthreads* library [3] has been used for its portability, and possibility of direct control of running threads to increase the efficiency of computations.

The parallelization affects only modules with a substantial contribution to the total time of simulation. Thus only calculations of gas and particles motion are

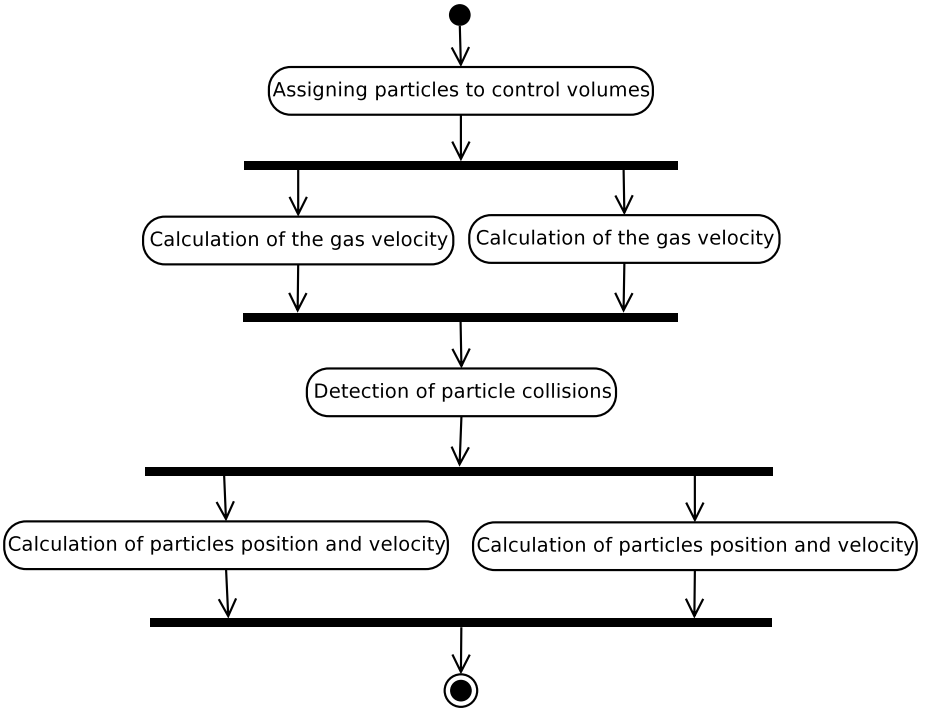


Fig. 2. Decomposition of computations among threads for one time step of simulation

parallelized, while the detection of particles collisions is performed sequentially [9]. For one time step of simulation, the decomposition of computations among threads is shown in Fig. 2, where a case of two threads is considered.

5 Experimental Evaluation of Parallel Implementation

5.1 Testbeds Used in Experiments

In our experiments the following testbeds were used:

1. SMP node equipped with two Athlon MP 1.66 GHz processors (512kB L2 cache),
2. SMP node with two Intel Pentium III 750 Mhz (256 KB L2 cache),
3. SMP node with two Intel Itanium2 1.4 GHz (256 KB L2 and 3 MB L3 caches),
4. SMP nodes with Intel Xeon 2.4 GHz and 3.2 GHz (1 MB L2 cache), with the Hyper–Threading option,
5. desktop machine with Intel Pentium 4 Mobile 2.4 GHz (512 kB L2 cache), with the Hyper–Threading option,
6. desktop machine with Intel Xeon 3.2 GHz with the Hyper–Threading option (1 MB L2 cache),
7. Hyper–Transport node with two AMD Opteron 1.6 GHz (1 MB L2 cache),

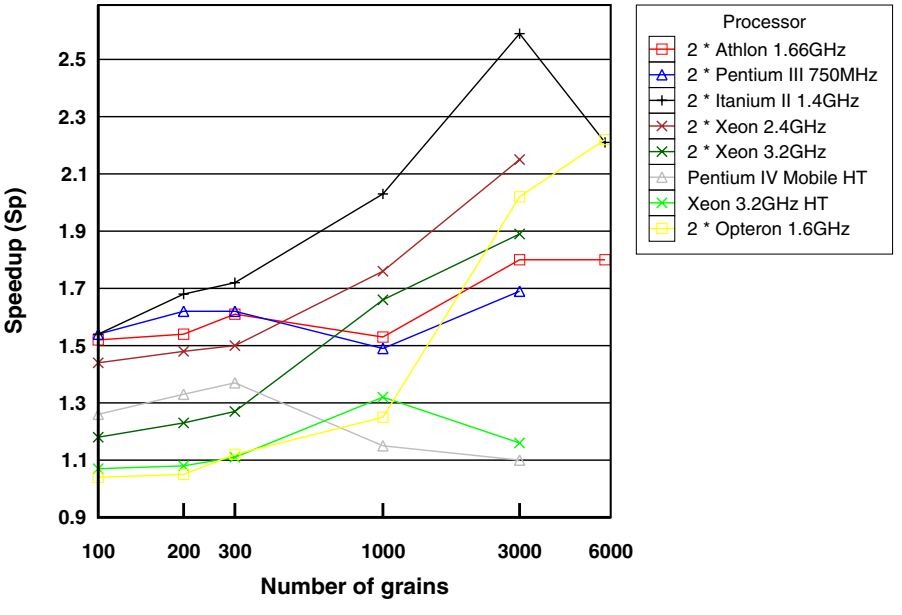


Fig. 3. Speedup S_p versus problem size (number of grains in simulation)

The *Hyper–Threading* technology was exploited for *Intel Xeon* and *Pentium 4 Mobile* processors. This technology allows for creating processors which contain multiple (in fact, two) logical processors per physical processor [11]. As a result, by executing more than one thread simultaneously this technology can generally improve the overall application performance. Furthemore, on SMP machines equipped with *Xeon* processors, tests with *Hyper-Threading* switched off were also performed.

During preliminary research, two basic compilers for PCs were tested: `GNU Compiler Collection` (`gcc`) [2] and *Intel* `icc` [4]. These tests shown a significant advantage of the `icc` compiler. Depending on a hardware platform used in simulations, code created by `icc` is from 36% to 40% faster than code created by `gcc`, version `3.3`, and from 64% to 104% faster than code created by `gcc-2.95`. All tests were carried out using the second level of code optimization, corresponding to `-O2` option of compiler. With respect to these results, all the further tests were performed using only the `icc` compiler.

5.2 Performance Results

In our tests, speedup $S_p = t_1/t_p$ is used as a basic parameter for measuring the implementation scalability. Here time t_p corresponds to the execution of

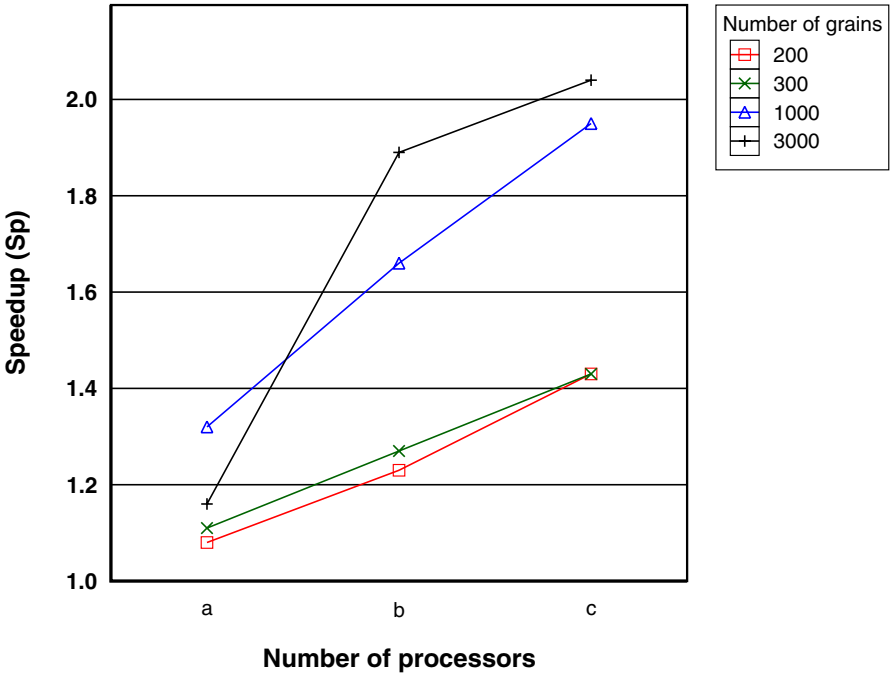


Fig. 4. Speedup S_p for *Intel Xeon* processors: a – one processor with *HT*, b – two processors without *HT*, c – two processors with *HT*

simulation in parallel on p processors, while t_1 stands for the sequential execution time of simulation [3].

The best achieved speedup (see Fig.3) is $S_p = 2.59$, which corresponds to using *Itanium2* processors for the problem with 3000 particles. Such a super-speedup is possible due to the enormous cache in *Itanium 2* (256 kB L2 and 3 MB L3). The large size of cache memory (1 MB of L2 cache) is also a reason for super-speedup in case of *Xeon* procesors. For *Itanium 2* processors, the further growth of the problem size results in the speedup reduction to $S_p = 2.21$ for 6000 perticiples. At the same time, for *Athlon* processors the same problem growth does not cause a speedup reduction. Very promising are scalability results obtained for *Opteron* processors, where only large problem sizes guarantee achieving a high speedup.

The performance evaluation performed for the *Hyper–Threading* (HT) technology gives results consistent with other tests of this technology [5, 11]. When it comes to the problem size, the speedup is from 1.08 do 1.37 – the results much worse than those gained on machines using the *SMP* architecture. Fig. 4 shows speedup for different problem sizes and different number of physical processors, when the same machine was tested in three configurations: (a) with one processor and *HT* switched on; (b) with two processors and *HT* switched off; (c) with two processors and *HT* switched on.

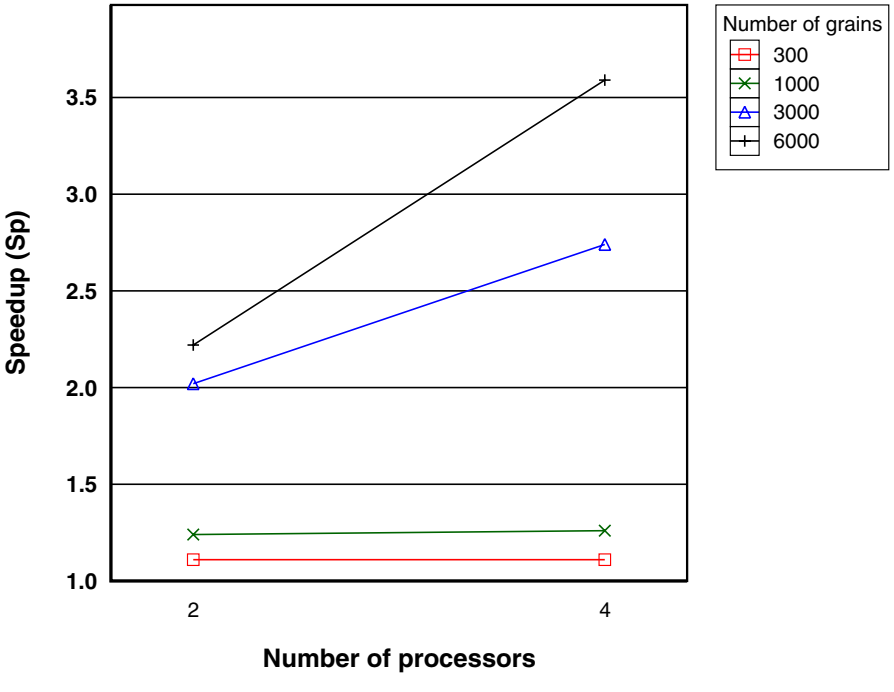


Fig. 5. Speedup for *Opteron* processors

Finally, during the tests with *Opteron* processors, the speedup of 1.1 – 2.2 is achieved for the two-proceesor architecture, depending on the problem size, while the speedup of 1.2 – 3.6 is gained on four processors (see Fig. 5.).

6 Conclusions

The obtained results of performance tests allow us to confirm usefulness of the shared-memory model for the parallel modelling of bi–phase gas–particle flows. This conclusion is especially important having in mind the multicore architecture of new processor designs.

These tests demonstrated also the influence of the implementation of a multiprocessor architecture on the scalibility of computations. With respect to comparison of architectures of *Intel* and *AMD* processors, in general the advantage of machines with *Intel* processors was observed, since these processors allow for achieving the effect of super-speedup. At the same time, the *HyperTransport* communication technology, used in new *Opteron* processors, gives good results only for sufficiently large tasks. Finally, it was shown that the *Hyper–Threading* technology allows for achieving a relatively small increase in performance (average 15%, maximum 37%).

References

1. Bis, Z.: Circulation Fluidisation of Polidispersed Mixtures. Czestochowa University of Technology, Czestochowa, 1999 (in Polish)
2. GCC Homepage, http://www.gnu.org/software/gcc
3. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. Addison-Wesley, 2003
4. Intel Compilers, http://www.intel.com/software/products/compilers
5. Intel Hardware Design Homepage, http://developer.intel.com
6. Leszczynski, J.: The Calculation of a Normal Force Between Multiparticle Contacts Using Fractional Operators. Computational Fluid and Solid Mechanics **2** (2003) 2043–2047
7. Patankar, N.A., Joseph, D.D.: Lagrangian Numerical Simulation of Particulate Flows. Int. J. Multiphase Flow **27** (2001) 1685–1706
8. Pluta, S., Wyrzykowski, R., Leszczynski, J.: Object Oriented Implementation of Modeling Bi-phase Gas-Particle Flows. Lect. Notes in Comp. Sci. **3019** (2004) 738–745
9. Pluta, S.: Organization of computation and development of simulation software by the example of modeling bi-phase flows. PhD thesis, Czestochowa University of Technology, 2005 (in Polish)
10. Sakai, M., Yamamaota, T., Murazaki, M., Miyoshi, Y.: Application of Distinct Element Method to Criticality Evaluation of MOX Fuel Fabrication Facility. Proc. 7th Int. Conf. on Nuclear Criticality Safety - ICNC 2003, 160–166
11. Wackowski, K., Gepner, P.: Hyper-Threading Technology Speeds Clusters. Lect. Notes in Comp. Sci. **3019** (2004) 17–26

Parallel Resolution of the Satisfiability Problem (SAT) with OpenMP and MPI

Daniel Singer and Alain Vagner

LITA - EA 3097,
Université Paul Verlaine de Metz, UFR MIM
Île du Saulcy, 57 045 Metz cedex, France
daniel.singer@univ-metz.fr, alain.vagner@sous-anneau.org

Abstract. The past few years have seen enormous progress in the performance of propositional satisfiability (SAT) solving, and consequently SAT solvers are widely used in industry for many applications. In spite of this progress, there is strong demand for higher SAT algorithms efficiency to solve harder and larger problems. Unfortunately, most modern solvers are sequential and fewer are parallel.

A number of recent propositions was concerned with dynamic workload balancing for parallel SAT solving. Here, it is a complementary approach that only explores an initial static decomposition for workload repartition. The two computational models of *Shared Memory* and *Message Passing* are compared, using OpenMP for Shared Memory and MPI for Message Passing implementations.

1 Introduction

The propositional Satisfiability problem (SAT) is one of the most studied in computer science since it was the first problem proven to be NP-complete by S. Cook in 1971. Nowadays, the Satisfiability problem evidences great practical importance in a wide range of disciplines, including hardware verification, artificial intelligence, computer vision and others. Indeed, a survey of Satisfiability in 1996 [13] contains over 200 references of applications. SAT is especially important in the area of Electronic Design Automation (EDA). In spite of its computational complexity, there is increasing demand for high performance SAT-solving algorithms in industry. Unfortunately, most modern solvers are sequential and fewer are parallel (see [28] for a complete review on parallel resolution of SAT).

The remainder of this paper is organized as follows. Section 2 briefly introduces the SAT problem and major concepts of the field. Section 3 gives an overview of the main techniques used in the efficient implementation of state-of-the-art sequential DPLL solvers. Section 4 briefly describes the main proposed methods to parallelize the core sequential algorithms. Section 5 presents our proposition in parallel resolution of SAT with OpenMP and MPI, followed by a brief concluding Section.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 380–388, 2006.

2 Preliminaries

Let $V = \{v_1, v_1, \ldots v_n\}$ be a set of n boolean variables. A (partial) truth assignment τ for V is a (partial) function: $V \rightarrow \{True, False\}$. Corresponding to each variable v are two literals: v and $\neg v$ called positive and negative literals. A clause C is a set of literals interpreted as a disjunction, $\Box$ denotes the empty clause and unit clauses have a single literal. A formula F is a set of clauses interpreted as a Conjunctive Normal Form (CNF) of a formula of the propositional calculus. A truth assignment τ satisfies a formula F (τ is *a solution*) *iff* it satisfies every clause in F, and the empty formula $\emptyset$ is always satisfied. A truth assignment τ satisfies a clause C *iff* it satisfies at least one literal in C and the empty clause $\Box$ is never satisfied.

Definition 1. *The Satisfiability Problem (SAT):*
-Input: A set of Boolean variables V and a set of clauses $\mathcal{C}$ over V.
-Output: Yes (gives a satisfying truth assignment τ for $\mathcal{C}$ if it exists) or No.

The restriction of SAT to instances where all clauses have at most k literals is denoted k-SAT. Of special interest are 2-SAT which is linearly solvable and 3-SAT is NP-complete. The Max-SAT problem is the optimization variant problem of SAT to find a truth assignment that maximizes the number of satisfied clauses. Nevertheless, the Max-2-SAT problem is well known to be NP-hard. Current research on propositional satisfiability is focused on two classes of solving methods: complete algorithms mostly based on *Backtrack search* and incomplete ones represented by variations of *Local search*. Complete algorithms are guaranteed to find a satisfiable truth assignment (a solution) if the problem is satisfiable, or to prove the problem unsatisfiability. Incomplete algorithms cannot prove the unsatisfiability even though they may be able to find a solution for certain satisfiable instances very quickly. Most of the more successful complete SAT solvers are variants of the Davis Putnam Logemann Loveland: *DPLL* procedure [9]. They work quite well in practice and are the most widely used SAT solvers. We may mention a number of works on the hybridation of incomplete and complete algorithms to solve Boolean Optimization problems. Moreover there is continuing interest in translations betwen CSP (Constraint Satisfaction Problems) and SAT[1, 11, 30]. We mention this aspect because comparatively much more work has been done in the parallel resolution of CSP than SAT ([14, 15]), and [18] studies decomposition methods for parallel resolution.

3 Efficiency of Sequential DPLL SAT Solvers

There has been extensive research effort to develop gradually more efficient SAT solvers ([12, 21]). Empirical evaluation of SAT solvers on benchmark problems has been of particular interest for both fundamental algorithms and theoretical understanding of SAT[5]. Web sites[26, 27] are libraries that collect a number of benchmark problems, solvers and tools to provide a uniform test-bed for solvers. An anual SAT competition is held as a joint event with the SAT conference. Below are the main elements for efficiency of sequential DPLL solvers.

- Better Branching Heuristic: this is the first element for efficiency of sequential solvers. Two factors have to be considered to define good general purpose decision strategies: to find a solution if it exists as fast as possible and to detect a contradiction as early as possible. Moreover, a "good" heuristic is one that does not require too much time to compute and provides a fairly accurate cost estimate. The branching rule problem has received many attentions and achieved many progresses. *Moms* (Maximum number of Occurrences in Minimum Size clauses) is one of the most widely used general heuristic. It favours the shortest clauses to obtain unit clauses and contradiction by UnitPropagation. Many versions of *Moms* heuristics have been proposed. *UP* heuristic, used in *Satz*[22] exploits the power of UnitPropagation by choosing the literal that would produce the maximal number of unit clauses. Recent works on dynamic learning and conflict analysis (see below) define new heuristics such as *VSIDS* (Variable State Independent Decaying Strategy) used in *Chaff*[24].

- Careful UnitPropagation Implementation: DPLL solvers spend the bulk of their effort (greater than 90%) searching for clauses implied in UnitPropagation, sometimes called *Boolean Constraint Propagation*. Therefore, an efficient UnitPropagation procedure implementation is the key for efficiency. [32] is an interesting deep case study on cache performance of SAT solvers showing that "cache friendly data structures is one of the key elements for efficiency". It gives comparative results of different data structures on various applications in term of run times, data access and cache miss rates. It finds that there are still a lot of space for improvements because the speed difference of main memory, L1 and L2 caches tends to be larger.

- Dynamic Learning, Conflict Analysis and Non-Chronological Backtracking: these CSP techniques were introduced in DPLL algorithms and have now become a standard in most of recent SAT solvers. If a conflict is encountered the DPLL algorithm analyses it for backtracking to a level so as to resolve this conflict and a 0-level backtracking means that the problem is unsatisfiable. A clause is called *conflicting clause* if it has all its literals assigned to *False*. Advanced conflict analysis relies on an *implication graph* to determine the actual reasons for the conflict. This permits to backtrack up more than one level of the decision stack and, at the same time, to add some clauses called *conflict clauses* to a database. This last operation is the base for the *learning process* which plays a very important role in pruning the search space.

- Specific SAT Problems Processing: DPLL solvers typically suppose CNF encoded problems but this is seldom the natural formulation of "real world" applications. Indeed, the CNF representation provides conceptual simplicity and implementational efficiency, it also entails considerable loss of information about the problem's structure that could be exploited in the search. There is a new interest in studying the CNF conversion for DPLL solving in different domains such as Planning, Bounded Model Checking (BMC) or Automatic Reasoning to improve search efficiency.

4 Parallel Resolution of SAT

Sequential computer performance improvements are the most significant factor in explaining the few existing works on Parallel Algorithms for Satisfiability compared to sequential ones. Indeed, the challenge to parallel processing is substantial in this area because there are still many problems considered out of reach for the best currently available solvers. We refer to [28] for a recent review of Parallel Resolution of SAT with complete algorithms, but we outline in this section the major progresses in this direction. An important characteristic of the SAT search space is that it is hard to predict the time needed to complete a specific branch. To cope with this problem, most of the parallel algorithms *dynamically partition* the search space assigning work to the available threads during run-time. The most difficult part consists of balancing the workload in such a way that on the one side idle time should be limited, and on the other side the workload balancing process should consume as few computing and communication time as possible.

[2] is the first real parallel implementation of the DPLL procedure on a message based MIMD machine, and it is the reference work of the domain. Excellent efficiencies have been obtained on a Transputer system with up to 256 T800 processors and with two different connexion topologies, the linear array and the 2-dimensional grid. *PSATO* [31] is the first DPLL solver for distributed architectures, and it introduces the central concept of *guiding path* to define non-overlapping portions of the search space to be examined. *//Satz*[19] is a parallel-distributed DPLL solver based on the *master-slave* communication model and *work stealing* for workload balancing. This work emphasizes the *ping-pong phenomenon* which may occur in workload balancing. [20] presents experimantal results on a cluster of 216 PCs interconnected by a Fast-Ethernet network where significant speedup is obtained. *PaSAT*[3, 4] is the first parallel DPLL solver with *intelligent Backtracking* and lemma exchange for *learning* (see Section 3). Feldman et al.[10] present a parallel multithreaded SAT solver on a single multiprocessor workstation with a *shared memory architecture*. It shows the general disadvantageousness of parallel execution of a backtrack-search algorithm on a single multiprocessor workstation, due to increased cache misses. More precisely, it observes the negative effect on otherwise highly optimized cache performance of the sequential algorithm. *GridSAT*[6, 7] is the first DPLL solver designed to solve real hard previously unsolved problems on a large number of widely distributed and heterogeneous resources: the Grid. Its philosophy is to keep the execution as sequential as possible and to use parallelism only when it is needed. It is based on *zChaff*[24] as sequential core solver and it implements a distributed learning clause database system. The baseline Grid infrastructure is provided by the *Globus* system. The experimental results are obtained on different but non-dedicated nationally distributed Grids, and a number of various challenge problems of the SAT'2002 conference is presented as test applications. To our best knowledge, [8] is the unique published work to investigate the parallel functional programming for DPLL implementation, inspite of its natural recursive expression.

5 Our Proposition

This section presents our proposition in parallel SAT solving. It is a complementary approach of all the previous ones restricted to dynamic workload balancing, in the sense that it only explores an initial decomposition for workload repartition. The two computational models of *Shared Memory* and *Message Passing* are compared, using OpenMP[25] for Shared Memory and MPI[29] for Message Passing implementations. Moreover our approach is to be as possible independent of the sequential solver running on all the processors for parallel execution. This is of particular importance because of the rapid evolution of the sequential state-of-the-art solvers. We put a parallel layer upon the target solver that is viewed as a blackbox. Among all the recent freeware DPLL implementations we have experimented *zChaff*, *Sato*, *kcnfs* and *Satz*. Here, for lack of place we will only present partial results obtained with *Satz* to illustrate the approach feasibilty, however the best absolute execution times have been obtained with *zChaff*. The sequential solver *Satz* developped by Chu Min Li [22] is written in C which enables the use of OpenMP and MPI as well without any extra processing. An important feature of *Satz* compared to other DPLL implementations is that it explores independently left and right subtrees making easier parallel implementation. It has no intelligent backtracking and sophisticated conflict analysis for learning, thus reducing the potential communications for lemma exchange between processors.

5.1 Initial Decomposition Strategy

The first step of our parallel proposition aims to obtain at most 2^k independent subproblems assigning both possible values $true$ and $false$ to some k "well chosen" variables. At each variable choice, the simplifications obtained by Unit-Propagation are achieved. All the subproblems are placed in a stack that can be then dynamically allocated to processors all along the parallel execution. The subproblems are defined by their associated guiding path as previously presented. In OpenMP implementation this is obtained by a simple parallel $forloop$ with a dynamic allocation strategy directive `#pragma` omp for schedule(dynamic). We refer to [16] for a detailed comparative study of the different repartition strategies enabled by OpenMP. In MPI implementation, a classical master-slaves communication protocol has been defined. The value for the k parameter may be adapted to the available processors number such that $2^k >> Nbproc$. This k parameter reflects the parallel *granularity* of our application (see below). In the reported experience it never goes beyond 10 giving at most 1024 potential tasks. The strategy for the choice of these k variables used to initially partition the problem is of particular importance. We will consider the three following strategies (see Section 3): *Satz*, the branching heuristic of *Satz* to give comparative results with the sequential resolution; *Moms*, the classical heuristic and *Rand*, the random choice of variables.

5.2 Experimental Results

Experimentation is conducted on a SGI Origin 3800 machine thanks to the CINES[1]. Its configuration is 768 R14000/500MHz processors, and 384Go of memory. Its architecture is of ccNUMA type, made of a number of interconnected blocks of processors giving 1.6GB/s data transfer rate.

The first application is called the *longmult* family, and it comes from the *Bounded Model Checking* domain. Each of the instances is associated with one output bit of a 16x16 shift and add multiplier. All the instances are unsatisfiable[2]. The *longmult14* and *longmult15* hardest ones are presented. The second application is called the *DES* family, and it comes from the logical cryptanalysis domain [23]. It is a new way to generate hard and structured SAT problems by light encoding a few numbers of tours of the DES encrypton system, and all these instances are satisfiable. Only the hardest ones *b1-k1.1* and *b1-k1.2* are presented here.

The following tables present for each problem instance: its reference, the number of variables v, the number of clauses c, the sequential CPU time of *Satz*, the respective parallel time and efficiency obtained with up to 32 processors.

Table 1 gives comparative results of OpenMP and MPI implementations on the *longmult* family with the *Satz* initial decomposition strategy and a granularity giving about 120 subproblems for *longmult14* and 200 subproblems for *longmult15*. Actually, there was a contradiction between the use of OpenMP for this application and our basic choice not to go inside the solver code. State-of-the-art DPLL solvers such as *Satz* or *zChaff* make intensive use of dynamic data structures, but unfortunately OpenMP does not yet permit the dynamic private (not shared) memory allocation, thus leading to mandatory memory management overhead ([15, 17]). In all the cases the MPI implementation overcomes the OpenMP one, but both provide noticeable linear speedup until 8 processors, then a regular decreasing efficiency for MPI and out-of-memory for OpenMP with more processors.

Table 1. OpenMP-MPI with *Satz* decomposition strategy

Pb.	Strat.	Seq	4 Pr.	Ef_4	8 Pr.	Ef_8	16 Pr.	Ef_{16}	32 Pr.	Ef_{32}
Lm14	*Satz*	4053	t4	e4	t8	e8	t16	e16	t32	e32
7.176v	OMP		1060	0.95	540	0.93	384	0.66	?	?
22.389c	MPI		971	1.04	503	1.00	327	0.77	284	0.44
Lm15	*Satz*	4865	t4	e4	t8	e8	t16	e16	t32	e32
7.807v	OMP		1251	0.97	665	0.91	?	?	?	?
24.351c	MPI		1211	1.00	622	0.97	361	0.84	274	0.55

Table 2 gives the comparative results of the three decomposition strategies *Satz*, *Rand* and *Moms* with MPI implementation on both *longmult* unsatisfiable

[1] *Centre Informatique National de l'Enseignement Supérieur.*
[2] http://www.cs.cmu.edu/ modelcheck/bmc.html

Table 2. Decomposition strategies with MPI

Pb.	Strat.	Seq	4 Pr.	Ef_4	8 Pr.	Ef_8	16 Pr.	Ef_{16}	32 Pr.	Ef_{32}
Lm14	*Satz*	4053	971	1.04	503	1.00	327	0.77	284	0.44
7.176v	Moms		1750	0.58	891	0.57	552	0.46	291	0.43
22.389c	Rand		2050	0.49	1286	0.39	748	0.34	410	0.31
Lm15	*Satz*	4865	1211	1.00	622	0.97	361	0.84	274	0.55
7.807v	Moms		2027	0.60	1102	0.55	568	0.53	350	0.43
24.351c	Rand		2448	0.50	1505	0.33	1108	0.27	554	0.27
b1-k1.1	*Satz*	6352	1319	1.20	660	1.20	336	1.18	173	1.15
307v	Moms		1562	1.01	797	0.99	423	0.94	150	1.32
1731c	Rand		1823	0.87	702	1.13	428	1.08	187	1.06
b1-k1.2	*Satz*	7709	1595	1.20	839	1.14	430	1.12	220	1.10
398v	Moms		1994	0.96	921	1.04	401	1.20	154	1.56
2124c	Rand		65	29	114	8.45	105	4.58	1461	0.16

Table 3. Granularity study

Pb.	NB-Pbs.	Seq	4 Pr.	Ef_4	8 Pr.	Ef_8	16 Pr.	Ef_{16}	32 Pr.	Ef_{32}
Lm14	32	4053	1308	0.77	1182	0.43	1070	0.23	1079	0.12
7.176v	60		1099	0.92	721	0.70	610	0.41	605	0.20
22.389c	112		990	1.02	513	0.99	344	0.73	292	0.43
Lm15	46	4865	1384	0.88	1009	0.60	832	0.36	786	0.19
7.807v	92		1273	0.95	728	0.83	568	0.53	456	0.33
24.351c	184		1247	0.97	628	0.97	364	0.83	277	0.55

family and *DES* satisfiable one. We may notice the *Satz* strategy to be always near the best one in this experiment. It shows superlinear speedup for the satisfiable instances. A non surprisingly random behaviour of the *Rand* strategy is observed too. It is wortwhile to notice that both *Rand* and *Moms* strategies are much more easily implemented compared to the *Satz* one which runs the solver up to a depth bound. Note that CPU time needed for the decomposition step is not reported because it can be seen as preprocessing and it is not significant compared to the search time.

The last Table 3 presents the comparative results obtained with different granularity values for the *longmult* unsatisfiable family. For the *DES* satisfiable family the granularity is much more interpreted in terms of non-determinism. It gives for each number of generated subproblems, its efficiency obtained with the *Satz* decomposition strategy in MPI implementation. It shows the important effect of this parameter on the overall efficiency especially for scalability.

6 Conclusion

After introducing the different components for sequential efficiency of the state-of-the-art solvers, we give the essential steps towards the parallel framework

progress of this last decade. However significant parallel efficiencies are still obtained by a number of recent propositions, more detailed research in all the directions are needed to improve the parallel performance. A simple static initial decomposition strategy may be applied to obtain reasonable efficiencies and then be combined with a more sophisticated dynamic workload strategy. The main characteristic of our approach is that it is completely independent of the sequential solver, thus can be viewed as a first brick of a future platform for parallel resolution of SAT.

The two parallel computational models used in this study are *Shared Memory* and *Message Passing*. It is not unheard of for both models to be applied simultaneously -threads on shared memory for "intra-node computations" and message passing among them for "inter-node communications". Such an hybrid approach could become standard and we will use it for further progress. Indeed new improvements can be done in the sequential resolution, the next step for a widespread use of Satisfiabiliy technique in real world applications remains the parallel efficiency challenge.

References

1. H. Bennaceur, The satisfiability problem regarded as a constraint satisfaction problem, In *Proc. of ECAI'96*, pp.155-159, 1996.
2. M. Böehm, E. Speckenmeyer, A fast Parallel Sat Solver - efficient work load balancing, In *Third Int. Symp. on AI and Maths.* AIMSA, Fort Lauderdale, Florida USA, 1994.
3. W. Blochinger, C. Sinz, W. Kchlin, PaSAT-Parallel SAT-Checking with Lemma Exchange: Implementation and Applications, In *Proc. of SAT'2001*[21], 2001.
4. W. Blochinger, C. Sinz, W. Kchlin, Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969-994, 2003.
5. S. A. Cook, D. G. Mitchell, Finding Hard Instance of the Satisfiability Problem: a survey, In *DIMACS Series in Discrete Maths. and TCS.*, AMS, Vol.35, pp.1-17, 1997.
6. W. Chrabakh, R. Wolski, GridSAT: a Chaff-based Distributed SAT Solver for the Grid, *Super Computing Conference, SC'2003*, Phoenix Arizona, USA 2003.
7. W. Chrabakh, R. Wolski, Solving "hard" stisfiability problems using GridSAT, 2004. (http:: www.cs.ucsb.edu/∼chrabakh/papers/gridsat-hp.pdf)
8. M. Cope, I. Gent, K. Hammond, Parallel Heuristic Search in Haskell, In *Trends in Functional Programming*, Vol.2, pp.65-73, S. Gilmore ed., Intellect Books, Bristol, UK, 2000.
9. M. Davis, G. Logeman, D. Loveland, A machine program for Theorem Proving, CACM, Vol.5 (7), 1962.
10. Y. Feldman, N. Derschowitz, Z. Hanna, Parallel Multithreaded Satisfiability Solver: Design and Implementation. *PDMC 2004*, 2004.
11. R. Génisson, P. Jégou, Davis-Putnam were already checking forward, In *Proc. of ECAI'96*, pp.180-184, 1996.
12. I. Gent, H, van Maaren, T. Walsh eds., *SAT 2000, Highlights of Satisfiability Research in the Year 2000*, Frontiers in AI and Applications, Vol. 63, Kluwer AC. Publ., 2000.

13. J. Gu, P.W. Purdom, J. Franco, B.W. Wah, Algorithms for Satisfiability (SAT) problem: a Survey, In *DIMACS Series in Dicrete Maths. and TCS.*, AMS, Vol.35, pp. 19-152, 1996.
14. Z. Habbas, M. Krajecki, D. Singer, Parallel resolution of CSP with OpenMP. In *Proc. of the second European Workshop on OpenMP, EWOMP'00*, Edinburgh, Scotland, pp.1-8, 2000.
15. Z. Habbas, M. Krajecki, D. Singer, Shared memory implementation of CSP resolution. In *Proc. of HLPP'2001*, Orléans, France, 2001.
16. Z. Habbas, M. Krajecki, D. Singer, The Langford's Problem: a Challenge for Parallel Resolution of CSP, In *Proc. of PPAM'2001*, LNCS Vol. 2328, Naleczow, Poland, 2001.
17. Z. Habbas, M. Krajecki, D. Singer, Parallelizing Combinatorial Search in Shared Memory, In *Proc. of the third European Workshop on OpenMP, EWOMP'02*, Roma, Italy, 2002 .
18. Z. Habbas, M. Krajecki, D. Singer, Decomposition Techniques for Parallel Resolution of Constraint Satisfaction Problems in Shared Memory: a Comparative Study. Special issue of ICPP-HPSECA01, Int. Jour. of Computational Science and Engineering (IJCSE), to be published, 2005.
19. B. Jurkowiak, C.M. Li, G. Utard, Parallelizing Satz Using Dynamic Workload Balancing. In *Proc. of SAT'2001*[21], 2001.
20. B. Jurkowiak, Programmation Haute Performance pour la Résolution des problèmes SAT et CSP, *Thèse de l'Université de Picardie*, Amiens, 2004.
21. H. Kautz, B. Selman: *Proc. of the Workshop on Theory and Applications of Satisfiability Testing, (SAT'2001)*, Elsevier Science Publishers, Electronic Notes in Discrete Mathematics Vol.9. 2001.
http://www.elsevier.nl/gej-ng/31/29/24/42/show/Products/notes/cover.htt
22. C. M. Li, Anbulagan, Heuristics based on unit propagation for satisfiability problems. In *15th Int. Joint Conference on AI, IJCAI'97*, Morgan Kaufmann Pub., Nagoya Japon, pp.366-371, 1997.
23. F. Massacci, L. Marraro, Logical cryptanalysis as a SAT-problem: Encoding and analysis of the U.S. Data Encryption Standard, *Journal of Automated Reasoning*, Vol.24(1-2), pp.165-203, 2000, (http://www.dis.uniroma1.it/ massacci/papers/).
24. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an Efficient SAT Solver, *Proc. of the 39th. DAC*, Las Vegas USA, 2001.
25. OpenMP Architecture Review Board, *OpenMP C and C++ Application Program Interface*, http://www.openmp.org.
26. SATLIB : http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/.
27. SATLive: http://www.satlive.org.
28. D. Singer, Parallel Resolution of the Satisfiability Problem: a Survey, In *Parallel Combinatorial Optimization*, El-Ghazali Talbi ed., Wiley and Sons, to appear.
29. M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J. Dongarra, MPI: the complete reference, the MIT Press, 1996.
30. T. Walsh, SAT versus CSP, *Proc. CP'2000*, LNCS 1894, pp. 441-456, Springer, 2000.
31. H. Zang, M.P. Bonacina, J. Hsiang, PSATO: a distributed propositional prover and its applications to Quasigroup problems, *Journal of Symbolic Computation*, Vol.21, pp.543-560, 1996.
32. L. Zang, S. Malik Cache Performance of SAT Solvers: a Case Study for Efficient Implementation of Algorithms, *Proc. of SAT'2003*, 2003.

Grid Technology for the Collaborative Enterprise

Žiga Turk

University of Ljubljana, Slovenia
ziga.turk@fgg.uni-lj.si

Abstract. In the global economy an increasingly popular concept of organization is that of a collaborative enterprise. An essential ingredient of such an enterprise is the information technology infrastructure that enable the coordination and the sharing of information and other resources. Grids have been proposed as the infrastructure for such organizations, however, to date they have been delivering high performance for computation and data while a lot remains to be developed for the grids to be readily useful in day to day enterprising. In particular, for the grid concept to be usefull it must redefine its understanding of a resource. This paper defines the collaborative enterprise identifies the role grids could play in engineering processes taking place in such organizations. It presents a high level architecture for grids based engineering services and tools, reviews it and defines the research and development issues. It is based on the results of an on-going European Union project InteliGrid - Interoperability of Virtual Organizations on a Complex Semantic Grid.

1 Introduction

Since 1980s a problem of the islands of computation has been emerging in practically every industry. Every intellectual work was getting IT support but was poorly interconnecting with other related processes thus the name "island of computation". Research started on how to interconnect those tools so that the slow and error prone re-entering of data could be skipped and complex processes involving several tools could have been automated. A well known illustration of this problem in the domain of architecture, engineering and construction is the "islands of automation" picture [1]. Conceptually, the integration solutions have been betting on the agreement on a commonly accepted and standardized data structures, such as the ISO-STEP or IAI-IFC standards. These data structures were derived from what has been called a "conceptual model" of the area of discourse. This approach was conceived at a time when file transfer (on floppy disks, CDs or as email attachments) was the usual method of information exchange. Their architectures and technical solutions are based on file-transfer idea (e.g. STEP physical files). However, since the interesting part of these standards have been the data structures describing the problem domain, the actual research prototypes used whatever was the state of the art for interoperability at the time. This included CORBA, COM and DCOM, internet and Web and recently the semantic web (overview these technologies in [2]). The Internet also provided the communication infrastructure for "a temporary network of companies, suppliers,

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 389–398, 2006.

customers, or employees ... with the purpose of delivering a service or product" (www.powerhomebiz.com/Glossary/glossary-T.htm). In short such networks are called virtual or networked organizations or collaborative enterprises (VO, NO, CE). Another popular definition of a VO is "a temporary collaboration to exploit a business opportunity" [3]. The term became a buzzword in the late 1990s, however, some industries have always been organized in such a way. Every construction project, for example, is such a temporary collaborative network of companies, suppliers and customers. Traditionally the communication medium for these networks in construction have been drawings on paper. Recently the paper is being replaced by digital files and models.

2 Grid Computing

The statement by Foster et al. [4] captures the essential requirements of collaboration inside a CE: " the problem is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations ... not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving in industry. This sharing is highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs". This statement became one of the definitions of grid computing, particularly for the evolution of grid technology towards semantic grid. It gave ground to the optimism that perhaps grid technology could provide the solution to the old problem of the islands of computation and offer infrastructure for the collaborative enterprise. Since the early 2000 there has been an increasing collaboration among the research communities of grid computing, semantic web and the collaborative enterprise communities which resulted in a better understanding among them, including their vocabularies. One project that such cross fertilization is happening is the InteliGrid project funded by the EU (www.InteliGrid.com). The author believes that it is the understanding of the term "resource" that has been causing some misunderstandings and that resulted in a novel and original interpretation of the roles of the grids in networked enterprizes.

3 Resources

In grid computing context (in which also Foster's definition was written) the term resource is used to denote machines, hosts and the CPU cycles, storage or communication facilities. In the WSRF, a WS-resource is defined as: "A Web service having an association with a stateful resource". A resource is definitively something on the other side of the keyboard. In engineering and industrial context the high level process model of industrial activities as phrased in the ISO Framework Standard originating in the 1950s has been "Resources are used in Processes that will result in Results and all these objects have Properties and

Characteristics [5]. This understanding of a resource is much wider. In it, a resource is anything needed to produce a product or a service and includes real world items such as people, raw materials, components, sub-assemblies as well stuff like energy and, last but not least, information technology that is needed to support the production process. This resource is predominantly something on this side of the keyboard. Foster's grid only becomes relevant for networked enterprising if and only if the term resource adopts the wider industrial meaning. A direct implication of this, however, is the extension of the universe of discourse that the grid middleware is concerned with. Not only purely IT concepts but domain concepts need to be addressed as well. Which implies that the so called semantic grid is not only interested in the semantic annotation of grid resources with concepts from some grid resource ontology, but in concepts from the domain ontology. The relation of these two domains is a research issue addressed in the InteliGrid project. Some early ideas on the conceptual and architectural implications are addressed in this paper.

4 Vision, Goals, and Requirements

The vision of the next generation collaboration platform for networked enterprises is to enable the industries with challenging integration and interoperability needs a flexible, secure, robust, ambient accessible, interoperable, pay-per-demand access to (1) information, (2) communication and (3) processing infrastructure. In a construction related virtual organization the group of companies and contractors involved would be different each time. Some, like the chief designer or main contractor would stay on the project longer, some would just come in, do their job, and get out; quickly and dynamically. A grid could provide a platform, which lets them get in and out of an extremely complex virtual enterprise, built around complex, structured information fast. The specific goal is to provide infrastructure where a user would simply get building information model information from the grid and also put it there. The complexity of the IT that will be used to physically store this information, protect it, make it available to other etc. would be totally hidden from the end user. Large scale use of services and agents, each with a potentially different schema, would not allow for her knowing the individual schemas. She would use terms from her professional vocabulary, from a generic engineering ontology, to find relevant information and services. Similarly, software authors would make the software commit to an engineering ontology semantically compliant with the standard building model and technically compliant with the grid middleware protocols. The "Save As ..." option would be replaced by "Save to grid" and the "Load ..." with an ontological query mechanism, that would search the grid for relevant information. The specific requirements of the industry include:

- **Security.** By using any kind of shared infrastructure, an organization is exposing some of its resources to other - partner organizations; in the case of project webs, one neutral location is used for the exchange of files. In the case of grids the resources could be much more scattered. The number

of providers of resources is much larger. Even a pool of resources within the company could be offered to the VO. Because of such scenarios, the grid should provide reliable authentication, secure and strongly encrypted communication from ground-up.
- **On demand access to services.** Engineering projects are distinctive and unique. This also requires specialized tools and software. It is not economical to buy tools that are used once; instead renting them and using them on-line, paying per use, not per license is a likely future option for all except the very popular engineering software. In addition, some simulation and analysis software might require computing power not available within the company. The services would therefore be rented and executing on the fastest and most powerful machine (or machines) on the grid.
- **On-demand access to information.** Product model databases structured according to STEP or IFC standards may be large. Logically, they would have to appear as a central database with all information about a designed product. Physically they would be distributed in a safe, reliable and redundant way across the grid.
- **Engineering language.** Although the language that the engineers use to describe the buildings is closer to STEP or IFC concepts than it is to lines and words in a non-structured data, the natural language is still on a higher semantic level. This level needs to be reflected in the architecture as well.

5 Architecture

The ideas presented are based on service-oriented architecture (SOA) and model driven automation (MDA) [6]. In a service oriented architectures, the unit of modularity is a service. Model-driven automation (MDA) supposes that services are defined my models, that models encapsulated within layers. The architecture presented in this paper is a high level architecture. One of goals when developing it was that it should have been fairly generic. This fact can be proven by trying to fit existing architectures of systems developed over the last decade into it. It is also used to identify the pieces that exist and the pieces that need to be developed.

5.1 End User View

As a way to enable the understanding of the architecture we start by presenting how an end user view might experience a services oriented system. From top to bottom (Fig. 1):

- **Shells and GUI layer.** She accesses the computing resources through various shells (e.g. Windows Explorer, KDE), browser based shells of collaborative environments (e.g ProjectWeb, CONJECT) and, of-course, she interacts with applications.
- **Applications and services layer.** This consists of tools that assist in the evolution of design and contribute to the information that is the results of the supporting process.

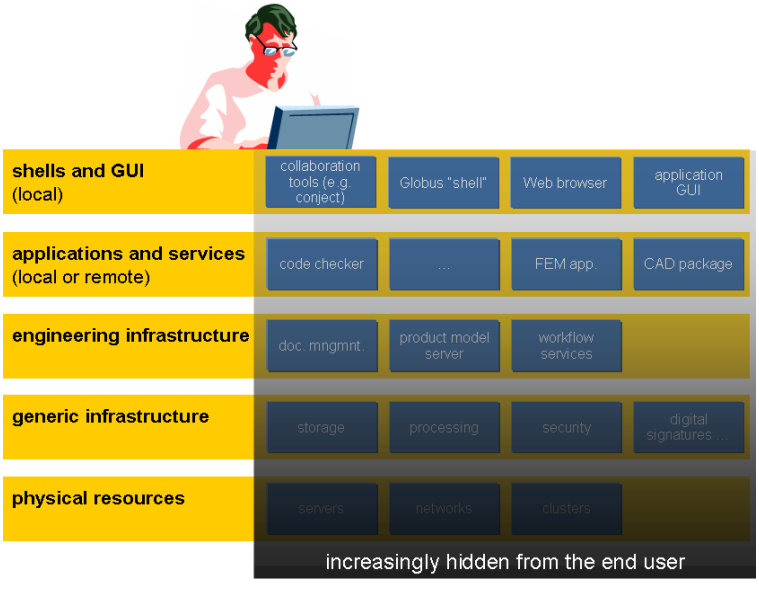


Fig. 1. End user perspective

- **Engineering infrastructure layer.** This layer is specific to engineering and AEC but does not change the design information.
- **Generic infrastructure layer.** This layer is infrastructure as well, but not specific and not aware of engineering or AEC. Tolls on this level could also be used in banking, medicine etc.
- **Physical resources.** Hardware and networks.

5.2 The Three Layers

The mechanisms that we use to structure the overall architecture is to split the discussion into (Fig. 2):

- concepts: this is something that exists in the form of standards, ideas, graphs, schemas, ontologies, notions etc.

Fig. 2. The three main layers of the architecture

- services and applications: this layer consists of software that can be compiled, installed, executed, that runs and communicates with other software.
- to run the applications and services need primitive resources such as networking, storage, CPU etc.

5.3 Conceptual Layer

In the architecture, this layer must provide information on what concepts exist in the system, and how they are encoded (Fig. 3). The concepts are organized into two axes. On the vertical access is what we call the conceptual stack.

- in the common sense layer are all the concepts we can think of; the CYC project [7] was an attempt to encode these concepts in a form of an ontology.
- in the engineering layer we have the concepts that the engineers that will use the system deal with; walls, bridges, windows, reinforcement bars etc. etc.
- in the services and grid layer are IT concepts that are describing the IT elements of which the system is built.
- in the resource layer are concepts describing low level resources, such as CPU, memory, networking, security etc.

On the horizontal axis is the so called semantic Web stack [8]. From left to right:

- Unicode is the way in which characters are encoded;
- Universal Resource Identifiers (URI) are they way in which the resources are identified
- XML and XML Schema Languages are the way in which structured information is transferred and its structure defined.
- RDF and RDF Schema have the same role in the narrower scope of resources.
- Ontologies define what exist for a number of agents or services on a higher semantic level that XML Schema. OWL family of languages is currently the most popular way of encoding the ontologies.
- Layers related to risk, logic and inference, proof and trust are hotly debated; they should culminate making sure that a web service can be trusted. Few mature technologies and solutions for these layers exist, therefore they are drawn in a dashed line.

Finally the intersections of the horizontal and vertical layering are examined. Existing and proven technologies are drawn with a grey background, future work as hollow, dashed rectangles. The items will be discussed top to bottom, left to right:

- common sense layer: efforts like CYC would fit in here.
- engineering layer: the schema layer is quite well developed with standards like IFC and STEP; there may be more work needed related to the process and communication. The authors are not aware of any work in construction IT that would have addressed the RDF. Whether it is needed or not is a research

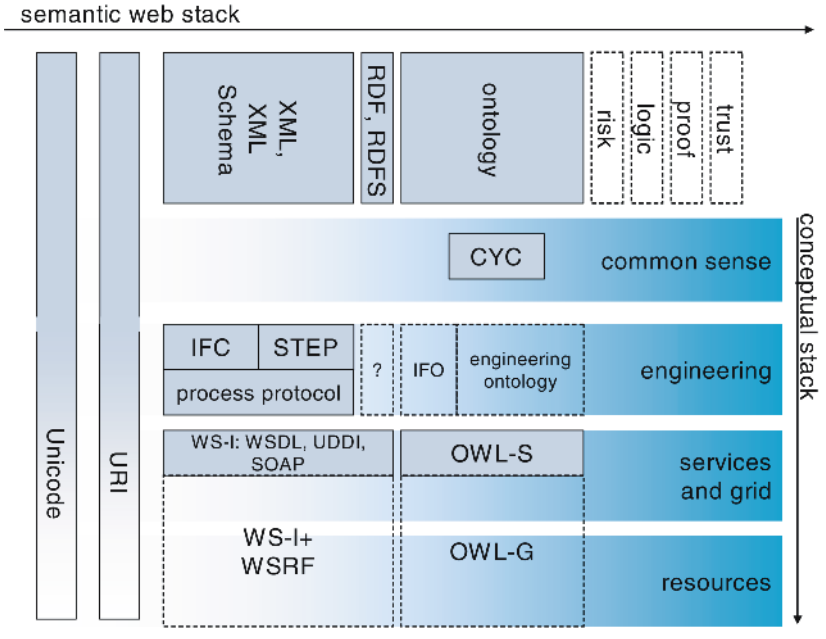


Fig. 3. Conceptual layers

question. On the ontology layer there seems to be a need for "Industry Foundation Ontology" (IFO) that could perhaps be derived on one hand from the IFC models and on the other from the work in the classification systems. This IFO could be further generalized into an engineering ontology.

- services layer: The web services interoperability (WS-I www.ws-i.org) provides a well developed set of standards, conventions and frameworks to address the web-services-kind-of functions but do not sufficiently address grid technology. The WS-I+ and the WSRF framework [9] are extending the concepts not only towards grid services but towards lower level grid resources as well. OWL-S (http://www.daml.org/services/owl-s/1.0/) is an ontology in which web services can be described; similarly there may be a need for extending it so that grid services can be described. We are naming this OWL-G (G for grid).
- resource layer: current ideas of grid development go onto the direction that anything is viewed as a service - the low level or virtualized hardware and networked resources as well, therefore, in our graph, the boxes from the services layer have been extended into this layer as well.

5.4 Services and Applications Layer

In the services and application layer anything is ether a service or an application (Fig. 4). These services implement the concepts and standards defined in the conceptual layer and use the resources of the physical resource layer. They talk

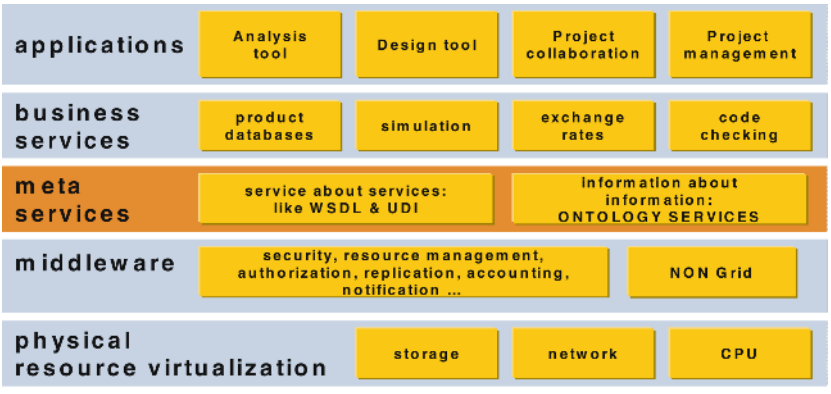


Fig. 4. Services and application layer

to each other using the web services protocols such as, but not limited to, SOAP or XML-RPC. Layers from top to bottom:

- applications: this is software through which people interact with the system. The applications can be rather autonomous and stand alone (for example a CAD modeler works off-line, the user, in the end, uploads the drawing to a shared repository). Other applications may depend on resources that are on-line. For example an application may look up product specification from an on-line database or have a complex simulation performed on an on-line supercomputer.
- business services: this software is on line and exposes it functionality in such a way that it can be used by remote clients. A project web site, for example, is such a primitive service. Web services based product model server could be a next step in its evolution. Labeling them "business services" we want to stress that these services are specific to a particular business (like construction).
- meta services are services about services. At these services, a service from the layer above can register so that it can later be fund by users. The services in the left hand side are rather well known and deal with established web services concepts. The right hand box is new and will have to deal with what ontology to particular services commit to and how to map concepts from the ontology into the schema with which they work.
- middleware provides the generic infrastructure for a service oriented architecture such as security and authentication mechanisms, resource management, accounting, notification etc. These mechanisms are much ricer in the grid than in the web services context, where the code that executes the service and its API are very tightly connected and where access and security policies are managed by each service individually.
- physical resource virtualization: current architectures are used to dealing with physical storage, files, network; developments in grid computing go into the direction where these basic resources are virtualized, requested and

fulfilled from a pool of resources, transparently to the service that is using them.

6 Critical Discussion

The high level architecture that was presented in the previous section tells us more on how to view at a particular system than how to implement it. This section provides a critical review. Strengths. The architecture uses some well established reference architectures as its baseline: the service oriented architecture, the semantic web stack and a common sense logic of specializing concepts from common sense through engineering to IT ones. It would be quite easy to map the architectures of interoperable systems for construction, such as ToCEE [10] or ISTforCE [11] into this architecture. Contrary to several others, it makes a very clear distinction about the issues that each of the diagrams is addressing. It is generic - only a few rectangles on a few layers are specific to AEC. This means that the implementation could make full use of technologies developed for any services oriented system. One could argue that construction does not need grid technology. The presented architecture abstracts the lower level services and it is irrelevant for higher level ones if the actual implementation is grid or web services based. Weaknesses. The architecture is depending on some developments in the semantic web / grid area and assumes that the web services solutions will extend to the grid itself. It is also rather conservative so that it could take advantage of existing technologies. The architecture is not described in a standard notation, but for high level architectures these notations hardly exist. The UML/RUP is only applicable on a detailed level for the design of one service. Opportunities. By closing the gap between engineering schemas and the WS-I platform AEC services could start capitalizing on the synergies of web service and product model work. By developing the Industry Foundation Ontology it could harmonize the otherwise divergent developments in the IAI-IFC and the ISO 12006-2 Classification Standards. Threats. Too much effort is spent on handling technology related issues and mastering the rapidly evolving web services and semantic web standards that indeed sometimes seem to loose touch with the reality of end user requirements. Not enough time remains to address the unique engineering issues. The technological and semantic baseline on which the next generation of engineering services would (according to this architecture) have to be built may be to complex and demanding for an average SME developing solutions for the AEC.

7 Conclusions

A high level architecture for a service oriented system for doing collaborative enterprising has been proposed. It bridges between the IT and engineering understanding of resources and virtual organizations. It is generic enough so that several of the past developments can fit into it. It is being refined in an on-going R&D project InteliGrid.

Acknowledgement

The work presented in this paper has been conducted in the context of an EU project InteliGrid and has been influenced by the thoughts and ideas of the projects partners beyond the author list. In particularly the paper owes to the discussions with Peter Katranuschkov, Krzysztof Kurowski, Vlado Stankovski, Matev Dolenc and Rober Balder. The financial support of the EU and of the industrial partners is gratefully acknowledged.

References

1. Hannus, M.: Islands of Automation in Construction. first published 1987, http://cic.vtt.fi/hannus/islands/ (last visited 2005)
2. Coyle, F.: XML, Web Services, and the Data Revolution. Addison-Wesley, Boston (2002)
3. Kazi, A. S., Hannus, M., Laitinen, J., Nummelin, O.: Distributed engineering in construction: findings from the IMS GLOBEMEN project. ITcon Vol. 6, Special Issue Information and Communication Technology Advances in the European Construction Industry, pg. 129-148, http://www.itcon.org/2001/10 (2001)
4. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (2002)
5. Ekholm, A.: ISO 12006-2 and IFC Prerequisites for Coordination of Standards for Classification and Interoperability. ITcon Vol. 10, pg. 275-289, http://www.itcon.org/2005/19 (2005)
6. Brown, A.: An introduction to Model Driven Architecture. IBM developerWorks, http://www-106.ibm.com/developerworks/rational/library/3100.htm (2004)
7. Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., Shepherd, M.: Cyc: Toward Programs with Common Sense. Communications of the ACM, 33(8), 30-49 (1990)
8. Berners-Lee, T.: XML and the Web. talk at XML World 2000 in Boston, Massachusetts, USA, http://www.w3.org/2000/Talks/0906-xmlweb-tbl/slide1-6.html (2002)
9. Atkinson, M., DeRoure, D., Dunlop, A., Fox, G., Henderson, P., Hey, T., Paton, N., Newhouse, S., Parastatidis, S., Trefethen, A., Watson, P., Webber, J.: Web Service Grids: An Evolutionary Approach, Open Middleware Infrastructure Institute, http://www.omii.ac.uk/paper-web-service-grids.pdf (2004)
10. Katranuschkov, P., Scherer, R.: Schema mapping and object matching: a STEP-based approach to engineering data management in open integrated environments. Construction on the information highway. CIB proceedings Turk Z (ed.), Univ. Of Ljubl jana, Slovenia http://itc.scix.net/cgi-bin/works/Show?w78-1996-335 (1996)
11. Katranuschkov, P., Scherer, R.,Turk Z.: Intelligent services and tools for concurrent engineering? An approach towards the next generation of collaboration platforms. ITcon Vol. 6, Special Issue Information and Communication Technology Advances in the European Construction Industry, pg. 111-128, http://www.itcon.org/2001/9 (2001)

Large Scalable Simulations of Mammalian Visual Cortex*

Grzegorz M. Wojcik and Wieslaw A. Kaminski

Institute of Computer Science,
Department of Complex Systems and Neurodynamics,
Maria Curie-Sklodowska University,
Plac Marii Curie-Sklodowskiej 5, 20-031 Lublin, Poland
gmwojcik@gmail.com

Abstract. Large artificial neural networks are examined. Structures discussed in this article simulate the cortex of mammalian visual system and its dynamics. Simulations of thousands of Hodgkin-Huxley neurons always require high computational power. Discussion of such networks parallelisation is presented in some detail. Analysis of simulation time, algorithm's speedup as a function of processors' number and density of connections is discussed as well.

1 Introduction

Human brain built of about 10^{11} neural cells is a hard object of simulation even for contemporary super-computers. Some idea of whole brain modelling was suggested by Maass [1] and since then it has been called Liquid State Machine (LSM) [2][3]. In general, the brain (or a fragment of it) is treated as a liquid. Mammalian brains' cortex is built of neurons organised in microcircuits [4]. Microcircuits form columns and the function of each column depends on its location in the brain. Cortical microcircuits turn out to be very good "liquids" for computing on perturbations because of the large diversity of their elements, neurons, synapses and the large variety of mechanisms and time constants characterising their interactions, involving recurrent connections on multiple spatial scales. Like Turing machine, the model of LSM is based on strict mathematical framework that guarantees under ideal conditions universal computational power [1].

Applying ideas of liquid computing [1] allows to decrease the number of neurons in constructed model. In addition, we can dramatically shorten the simulation time using cluster-based parallelised simulations of groups of microcircuits. In this communication we present some method of visual system's model parallelisation and benchmarking results.

* This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098 "CLUSTERIX - National Cluster of Linux Systems".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 399–405, 2006.

2 Model of Mammalian Visual System

Discussed model of mammalian visual system consists three main modules (Fig. 1). The idea of LSM requires such architecture, so we have: "Input" (retina), "Liquid" and output called "Readout" [1]. The Retina is built on 16x16 square-shaped grid and it is divided into 16 patches (4x4). Each patch is connected with HHLSM (Hodgkin-Huxley Liquid State Machine) column which simulates

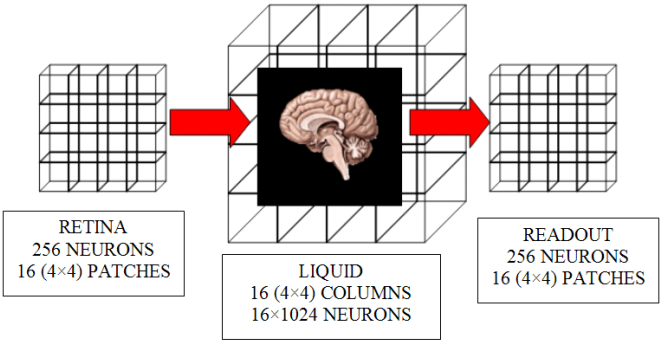


Fig. 1. Scheme of simulated visual system

the Lateral Geniculate Nuclei (LGN) and ensemble of cortical microcircuits. HHLSM consists 1024 cells put on 16x16 grid. There are layers arranged in each column (Fig. 2). Set of columns simulates the Liquid which is connected with the Readout device. The Readout's architecture is similar to the Retina, in analogy it is divided into 16 patches with 16 cells in each patch. Connections among layers and neurons of each layer are established with some probability i.e.

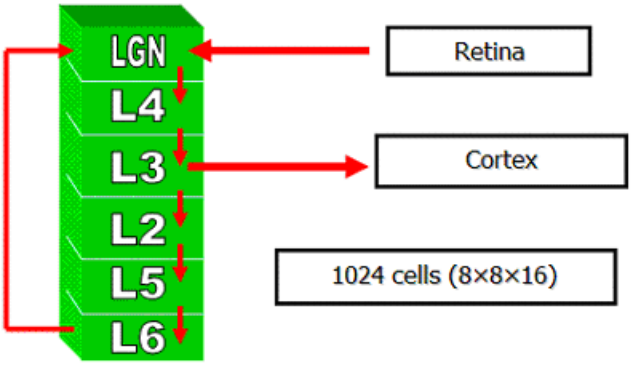


Fig. 2. Structure of HHLSM column as the fundamental microcircuit

p=10%. All simulations discussed in this paper are conducted in parallel version of GENESIS for MPI environment (for detail see Appendix A). Neurons used in simulations are built according to Hodgkin-Huxley model [5] (for detail see Appendix B).

Such model can be easily scaled into multiprocessor simulation. In referred research each column and its corresponding retinal or readout patches should be simulated on one node. Note that in that case we require 16 processors for the best realisation of the model and additional one for control of simulation. However, both the Retina and the Readout may be easily divided into 4 (2×2), 64 (8×8) or 256 (16×16) patches, depending on the number of processors available. Thus, if each patch is connected with corresponding HHLSM column - we will have possibility to conduct a simulation of about 256 thousands Hodgkin-Huxley neural cells.

3 Parallelisation and Results

We investigate the model consisting 16896 neurons (as the Liquid is simulated by ensemble of 16 HHLSM columns). 30% of randomly chosen retinal cells are stimulated and the signal is transformed by the Liquid. As a result we obtain some activity of the Readout device. We simulate $20ms$ of biological work for such system. The main objective of the referred research is to check the dependence of simulation time from the number of processors used for simulation and

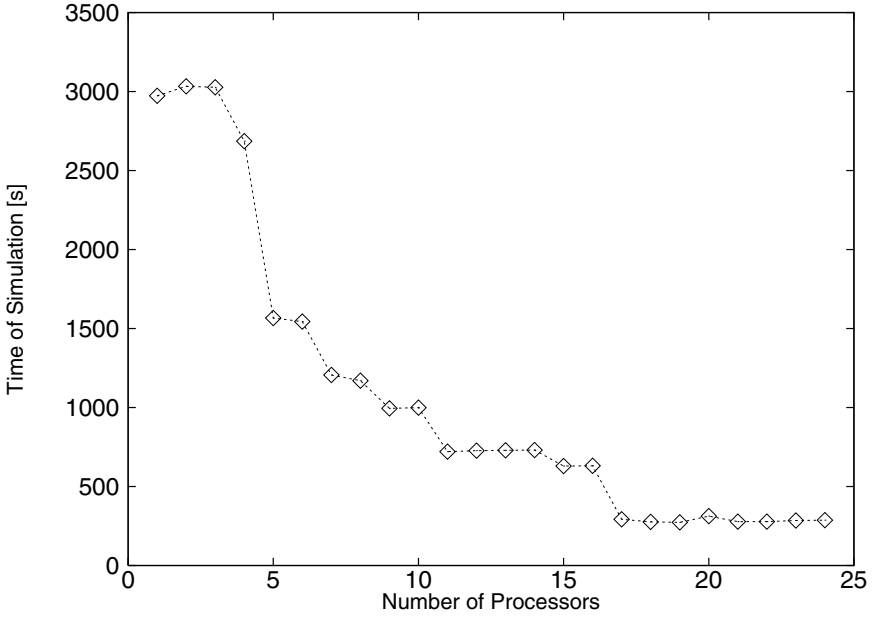


Fig. 3. Time of simulation as a function of processors' number

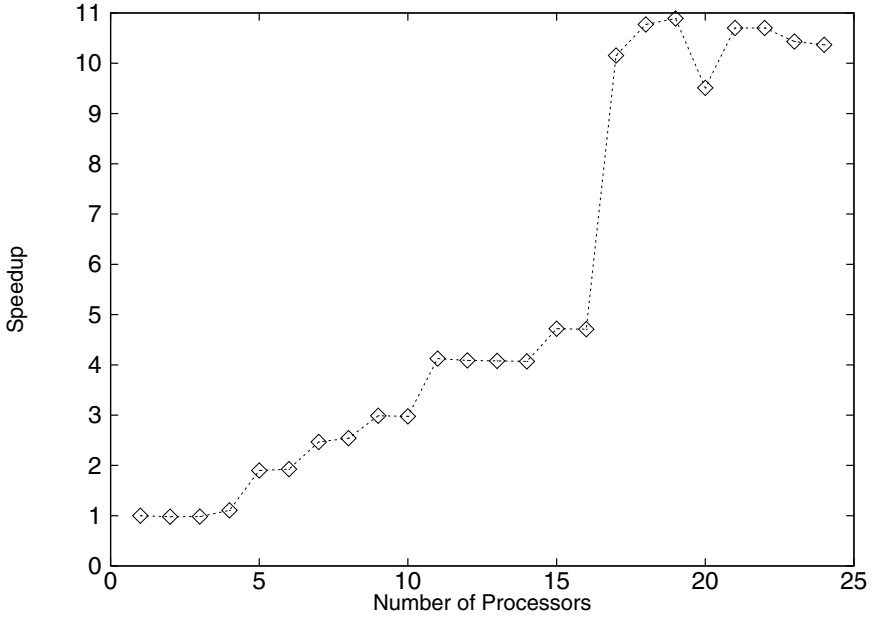


Fig. 4. Speedup of simulation as a function of processors' number

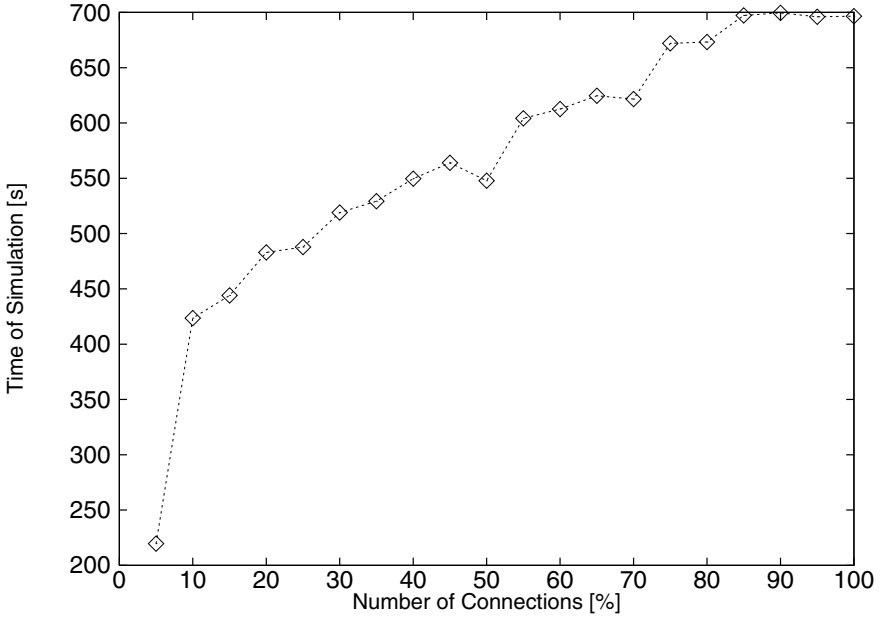


Fig. 5. Time of simulation as a function of the probability of connections arranged inside each HHLSM column (for detail see text)

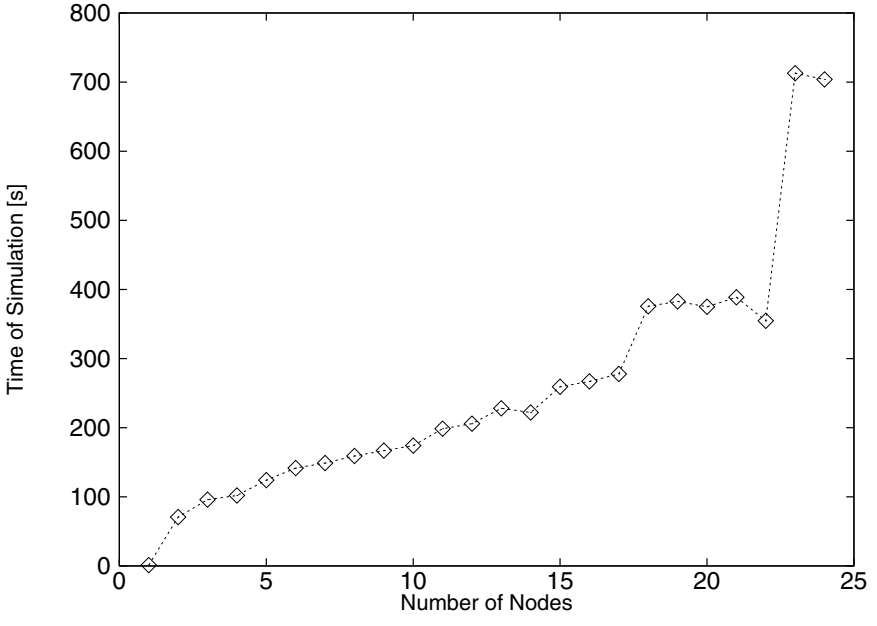


Fig. 6. Time of simulation as a function of MPI threads. The number of MPI threads is equal to the number of active HHLSM columns.

from the percentage number of connections established among neurons in the liquid.

As we mentioned before, the architecture of the network implies that the problem can be most effectively parallelised into sixteen main nodes with one controlling node. Results confirm our expectations (see. Fig. 3). Simulation's time reaches its minimum when the model parallelised for 17 nodes runs on 17 processors. Note that increasing the number of processors is useless for the algorithm with 17-node parallelisation implemented.

In analogy, Fig. 4 presents the simulation's speedup as a function of the processors' number. In the best case simulation is about 11 times faster than the one computed with from 1 or even 3 CPUs.

Fig. 5 presents the simulation's time as a function of the probability of connections arranged inside each HHLSM column. One can note that the time of simulation does not increase rapidly and for the full connection it reaches about 700 seconds.

We also investigated the model with some HHLSM columns being disabled. Fig. 6 shows how the time of simulation depends on the number of MPI threads. The number of MPI threads is equal to the number of active HHLSM columns. Neurons in columns are connected with the probability $p = 10\%$. Results obtained for 16 HHLSMs are comparable with these from Fig. 3, however, applying additional MPI threads in model parallelised for 17 nodes seriously increases the simulation's time.

4 Conclusions

In this paper we report results of mammalian visual cortex' simulations. We simulated about 16 thousands Hodgkin-Huxley neurons organised in layers and cortical columns - microcircuits. Modular structure of visual cortex allows for applying good parallelisation as particular microcircuits can be simulated on separate nodes. Our model is scalable. Though, we can easily increase the number of neurons in each cortical column which will let us run simulations consisting of more than 256 thousands Hodgkin-Huxley neurons. This will help us build more realistic model of visual cortex.

Communication among nodes is arranged by parallel version of GENESIS simulator. For this article we present one of the first results obtained in GENESIS parallelised for Linux MPI. Achieving the speedup of 11 for 17 processors is satisfactory and lets us predict similar magnitudes for much larger, scaled simulations.

All of discussed simulations were conducted on the local cluster. Our machine is part of CLUSTERIX - Polish GRID project [7]. Having access to 800 processors and increasing the number of simulated microcircuits we can imagine a structure consisting several millions of neural cells simulated in similar way. This will lead to creation of very sophisticated models and such attitude can open for us quite new field of computational complex systems' research.

Acknowledgements

This work has been supported by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098.

References

1. Maass W., Natschlager T., Markram H.: Real-time Computing Without Stable States: A New Framework for Neural Computation Based on perturbations. Neural Computations. **14(11)** (2002) 2531–2560
2. Kaminski W.A., Wojcik G.M.: Liquid State Machine Built of Hodgkin-Huxley Neurons - Pattern Recognition and Informational Entropy. Annales Informatica UMCS, **1** (2003) 107–113
3. Wojcik G.M., Kaminski W.A.: Liquid State Machine Built of Hodgkin-Huxley Neurons and Pattern Recognition. Neurocomputing **239** (2004) 245–251
4. Gupta A., Wang Y., Markram H.: Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. Science **287** (2000) 273–278
5. Hodgkin A. L., Huxley A. F.: A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in nerve. J. Physiol. **117** (1952) 500–544
6. Bower J. M., Beeman D.: The Book of GENESIS - Exploring Realistic Neural Models with the GEneral NEural SImulation System. Telos, New York (1995)
7. CLUSTERIX - National Cluster of Linux Systems: http://clusterix.pcz.pl

Appendix A: Details of Simulations' Hardware and Software Environment

The local cluster used for all simulations and discussed in this contribution is built of 12 machines and 1 additional machine - so-called "access node". Each SMP machine has two 64-bit 1.4 GHz Itanium2 IA64 processors with 4 GB of RAM memory. Cluster works under control of Debian Linux Sarge (v. 3.1) and 1.2.6 kernel version. The model is simulated in GEneral NEural SImulation System GENESIS v.2.2.1 with its MPI extension. A gcc compiler was used for general system configuration. In near future we expect to recompile the system with Intel CC/FC 8.1 compiler.

Appendix B: Details of Hodgkin-Huxley Neurons

Our HHLSMs consist of multicompartmental neurons with two dendrites compartments, a soma, and an axon. Dendrites contain synaptically activated channel and the soma has voltage activated Hodgkin-Huxley sodium and potassium channels. The behaviour of each compartment is equivalent to the behaviour of some electrical circuit and is described by differential equation:

$$C_m \frac{dV_m}{dt} = \frac{(E_m - V_m)}{R_m} + \sum_k [(E_k - V_m)G_k] + \\ + \frac{(V_m^{'} - V_m)}{R_a^{'}} + \frac{(V_m^{''} - V_m)}{R_a}. \quad (1)$$

Thus, each circuit is characterised by a group of parameters set as follows: resistances $R_a = 0.3\,\Omega$, $R_m = 0.33\,\Omega$, capacity $C_m = 0.01\,F$, and potential $E_m = 0.07\,V$. For the soma compartment $E_k = 0.0594\,V$ whilst for the dendrite $E_k = 0.07\,V$. Conductance for each type of ionic channels is chosen to be: $G_K = 360\,\Omega^{-1}$ and $G_{Na} = 1200\,\Omega^{-1}$. The soma has a circular shape with the diameter of $30\,\mu m$, while dendrites and axon are cable like with the length of $100\,\mu m$. All other parameters are chosen as suggested by GENESIS authors to simulate behaviour of the biological-like neurons [6]. More details concerning Hodgkin-Huxley model one can find in [5].

Optimised Scheduling of Grid Resources Using Hybrid Evolutionary Algorithms

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

Forschungszentrum Karlsruhe GmbH,
Institute for Applied Computer Science,
P.O. Box 3640, 76021 Karlsruhe, Germany
{wilfried.jakob, quinte, uwe.stucky, wolfgang.suess}@iai.fzk.de

Abstract. The present contribution shall illustrate the necessity of planning and optimising resource allocation in a grid. Requirements to be met by a resource management system will be defined. These requirements are comparable with the requirements on planning systems in other fields, e.g. production planning systems. Here, various methods have already been developed for optimised planning. Suitable methods are Evolutionary Algorithms. Based on an example from the field of production planning, the performance of these methods is demonstrated and use in the GORBA resource broker shall be described.

1 Introduction

With the growing acceptance of grid computing, the number of resources in a grid environment and the number of users increase constantly. For the best possible usage of grid resources and most rapid execution, efficient planning of the grid resources is required. This contribution shall illustrate the use and benefits of modern resource management methods in a grid environment. It will be shown that there still is considerable need for the use of optimisation processes in allocation planning. This gap shall be closed by the global optimising resource broker GORBA (Global Optimising Resource Broker and Allocator) that is currently being developed. The concept of GORBA and the underlying optimisation processes shall be outlined.

2 Resource Management in a Grid Environment

A resource management system in a grid environment is responsible for allocating grid resources to waiting requests for resources. In this context, the following requirements are made on a modern resource management system [1, 2]:

- Quality of Services (QoS)
  - guaranteed resource usage (advanced reservations)
  - negotiation of resource usage
  - deadlines (+ malleability)

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 406–413, 2006.

  - co-allocation for multi-site jobs and complex workflows
  - service level agreements (SLA)
  - different QoS levels
- Reliability / Fault-Tolerant Scheduling
  - failure detection & recovery
- Grid Economics
  - payment and penalties for resource usage, failures, and violated SLAs
  - load balancing in the grid

These requirements are made by both the user and the supplier of grid services. As a rule, each user attaches importance to cheap and a rapid execution of the job with guaranteed response times and the possibility of using and reserving certain resources (QoS). The supplier tries to reach a homogeneous usage of his resources and to serve all users equally well (grid economics). To meet these partly contradicting requirements, the best possible solution has to be found. Both users and suppliers of grid services additionally ask for reliability and fault tolerance.

Resource management systems can be divided into queuing systems and planning systems [1, 3]. The difference between both systems lies in the planned time window and the size of the set of jobs considered. Queuing systems try to allocate the resources available at a certain time to the current waiting request for resources. Resource planning for the future for all waiting requests is not done. In contrast to this, planning systems plan for the present and future, which results in an assignment of start times to all requests. Today, almost all resource management systems belong to the class of queuing systems. Contrary to queuing systems, planning systems require more information, such as the duration of execution, long-term availability of ressources, etc. For this reason, implementation of queuing systems usually is much easier. However, a queuing system is efficient in case of a low usage of the system only. In the case of increased usage, the queuing system reveals considerable weaknesses with respect to the quality of services, resource usage, and execution time of the individual grid jobs. For instance, for waiting grid jobs no statements can be made with respect to the presumable time of job execution.

Presently, resource management systems [4] exist in e.g. the grid systems Unicore [5], Nimrod/G [6], and Condor-G [7]. Unicore allows for the listing of suitable resources, together with the costs and the presumable execution time. Condor-G allows for resource finding according to criteria given by the user. In Nimrod/G, conventional optimisation in terms of costs or time or both is possible. Usually, resource management systems in grid environments are mere "resource finding systems" with manual resource allocation. At best, optimum scheduling of the actual job takes place with previous planning being maintained (e.g. Nimrod/G).

Automatic resource allocation should be made such that the above requirements on a resource management system, e.g. best possible usage of resources, guaranteed and short response time, etc., are fulfilled. Comparable problems are dealt with in many industrial resource planning tasks, such as in production

planning. A typical task of production planning is the allocation of alternative processing stations (i.e. resources) to partial jobs in a given order. The main objectives include execution of all jobs within their due dates and as rapidly as possible, preferred treatment of rush orders, homogeneous resource usage, and efficient and fast replanning due to equipment breakdown, cancellation of jobs or new orders. Such planning problems are NP-complete and as no efficient exact solution methods are known for this class of problems, heuristic methods are applied to find appropriate solutions within an acceptable time. The planning problems in a grid environment are very similar to the previously described planning problems. Differences consist in the variable availability of resources and in the difficulty to predict the duration of execution of individual jobs, which aggravates planning and frequently gives rise to replanning.

Our objective is the development of a resource management system for a grid environment, which fulfils the above requirements in the best possible manner. It generates allocation plans for the resources existing and the jobs to be executed, in which it is specified when each job is executed on which resource. For the optimisation of the allocation plans, the optimisation tool HyGLEAM (*Hy*brid *G*enera*l*-purpose *E*volutionary *A*lgorithm and *M*ethod) developed at our institute and tested for a variety of applications is used [8, 9]. The concept of our global optimising resource broker GORBA shall be explained below. A similar approach is described in [10].

3 GORBA - Global Optimising Resource Broker and Allocator

Before focusing on our resource broker GORBA, the concept of resource management in a grid environment, in which GORBA is embedded, shall be explained [11]. Our concept of a heterogeneous grid environment is based on describing the grid task as an instantiated workflow, called application job. An application job consists of a workflow definition and the corresponding data. Here, any user-defined structure of the workflow is allowed (e.g. parallelism, sequences, splitting, or joining) [12].

Handling of application jobs requires a dedicated grid middleware. This grid middleware receives the application job from the application and analyses and distributes parts of the application job in the grid. For this purpose, the grid middleware divides the application job into single grid jobs as described by the workflow. After processing these single grid jobs, the grid middleware collects the overall result and sends it back to the application. Figure 1 shows the principle of our grid environment.

Resource management is divided into two services, the resource broker and the job manager. Figure 1 shows the details of our resource management systems. The resource broker GORBA receives the application job that consists of a workflow definition and the corresponding data. It analyses the workflow and generates the single grid jobs from the application job. The resource broker acquires the capacities of the work nodes by using the information service and

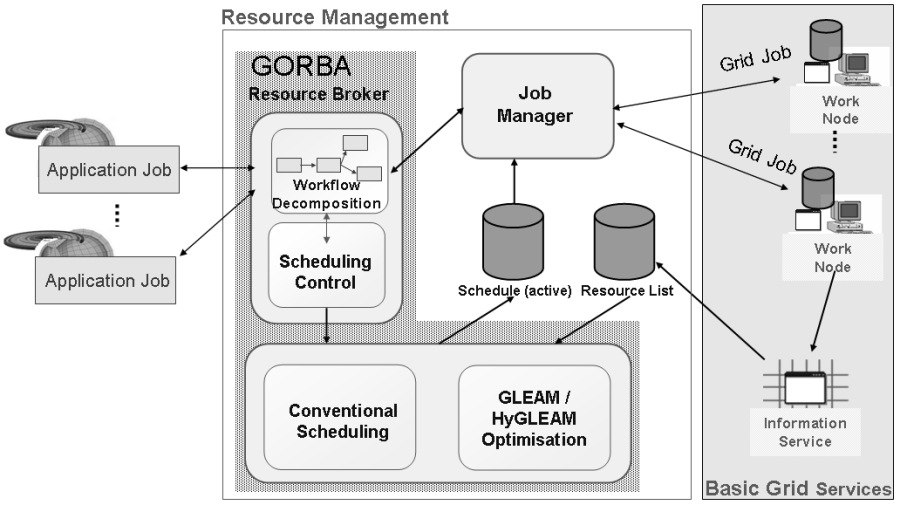


Fig. 1. Resource Broker GORBA embedded in a grid environment

plans the distribution of the single grid jobs. For this task, GORBA is equipped with two planning components that are based on conventional and evolutionary processes, respectively. In any case, conventional planning is made. In cases of smaller usage, this conventional planning will be completely sufficient. As soon as higher usage results in allocation conflicts or waiting situations, the conventional planning result is taken as basis of subsequent planning using the Evolutionary Algorithm HyGLEAM. Thus, it is ensured that the planning results of HyGLEAM have the quality of the conventional planning results at least.

GORBA generates an optimised allocation plan that distributes the grid jobs to the grid resources. The optimisation objectives are:

- Favourable/best allocation of all jobs not yet started to all resources, unless the latter are occupied by jobs already started.
- Individual weighing between costs and execution time per application job.
- Option "rush order": specification of acceptable additional costs per time unit of earlier execution compared to a given time.
- Specification of global secondary objectives, such as homogenous working loads.

For every objective a quality function is defined, which delivers a normalised quality value, from which a weighted sum is calculated. Additionally, penalty functions are used for situations like the violation of due dates. Later extensions of GORBA will aim at optimised data storage in terms of low-cost storage locations, reasonable transfer costs, good network performance, etc. Permanent

replanning takes place in case of the following events: new application job, cancellation of an application job, resource failure, new resources, and application job execution with a deviation from planning time larger than a given limit.

A schedule generated by GORBA is handed over to the job manager for execution. Therefore, the job manager has access to the grid jobs and the assigned work nodes. It is responsible for the execution of the grid job on the assigned work node. After a grid job has been completed, the job manager transmits the intermediate result to the next grid job or a possibly occurring error to the resource broker. GORBA can be embedded in any grid middleware (e.g. Globus [13]) and can use their basic grid services.

Open problems consist in the difficult comparability of various hardware and software platforms with respect to execution time, the presently frequently lacking availability of information by the work nodes, and the lacking accuracy of time estimates in the job description by the user. We think that these objectives must be tackled, if an improvement of QoS is required regardless which kind of planning method is used, as this is a general problem of the transition from queuing to planning systems.

3.1 Hybrid Evolutionary Algorithm HyGLEAM

HyGLEAM is a hybrid consisting of application-independent local search algorithms and the Evolutionary Algorithm GLEAM (General Learning Evolutionary Algorithm and Method) [14]. GLEAM is an Evolutionary Algorithm of its own that combines elements from Evolution Strategy and real-coded Genetic Algorithms with data structuring concepts from computer science. Coding is based on chromosomes consisting of problem-configurable gene types. The definition of a gene type constitutes its set of real, integer or Boolean parameters together with their ranges of values allowing for mutation operators that take explicit restrictions into account. Among others, GLEAM uses mutation operators influenced by the Evolution Strategy in so far, as small parameter changes are more likely than greater ones. Mutation can also change the gene order and add or delete genes in the case of dynamic chromosomes. GLEAM uses ranking-based selection and elitist offspring acceptance. A detailed description of the present state of GLEAM can be found in [15]. To keep the hybrid generally applicable suitable local search algorithms must be derivative-free and able to handle restrictions. Two well-known procedures from the sixties were chosen, since they meet these requirements and are known to be powerful local search procedures: the Rosenbrock algorithm [16] and the Complex method [17]. We use an implementation according to Schwefel, who gives a detailed description of both algorithms together with experimental results [18]. Figure 2 shows the pseudo code of that hybridisation method of HyGLEAM we use for scheduling (memetic algorithm part of HyGLEAM). As this paper focuses on scheduling of grid jobs and due to the lack of space HyGLEAM and its basic algorithms have been described here very briefly only and the interested reader is referred to given literature.

```
Initialise and evaluate start population
REPEAT UNTIL stop condition is satisfied (generational loop)
  FOR all individuals of the population
    Choose partner  (ranking-based selection)
    FOR all genetic operations consisting of one or more genetic operators
      Produce offspring and evaluate them
    Improve best offspring by selected local searcher
    IF Lamarckian evolution
      Update chromosome of the best offspring to its improved version
    Accept or reject best offspring according to the acceptance rule
Deliver best individual (and further, if required) as result
```

Fig. 2. Pseudo code of the used hybridisation of HyGLEAM

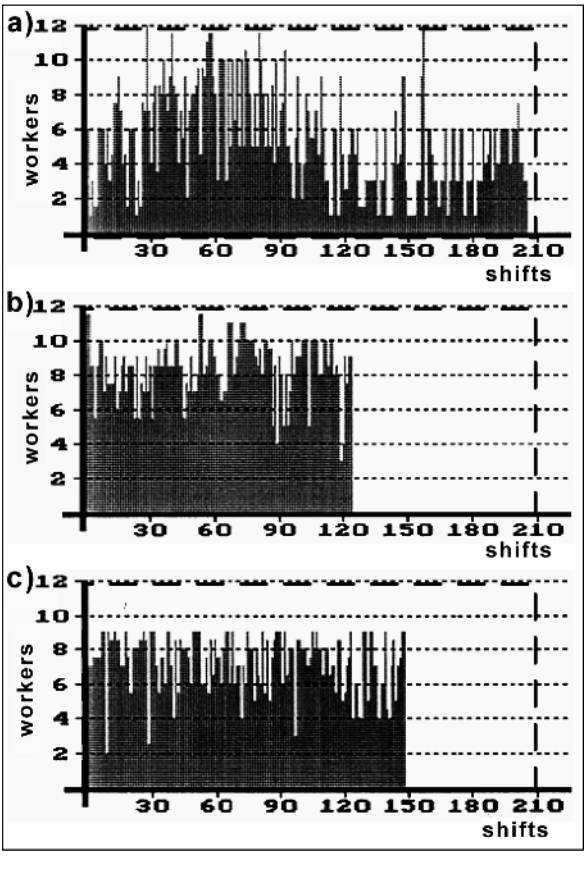


Fig. 3. Results of the scheduling task from chemical industry. The maximum numbers of workers required per shift is shown for a) the manual schedule, b) a time-optimised schedule and, c) a worker- and time-optimised schedule both made by GLEAM.

3.2 Scheduling Example from Chemical Industry

In order to demonstrate that EAs can be applied successfully to scheduling tasks, the results of a scheduling and resource optimisation problem solved by Blume and Gerbe using GLEAM shall be reported [19]. with batches with varying numbers of workers being required during the different phases of each batch. The objective of scheduling these batches means a maximum reduction of production time and peak number of workers per shift (human resource). Restrictions like due dates of batches, necessary pre-products from other batches, and the availability of shared equipment must also be observed. Allocation conflicts are solved by the sequence of the batches within a chromosome. As that can be overwritten by suitable changes of the starting times, however, the combinatorial aspect is limited to solving allocation conflicts. The concrete planning task reported here consists of 87 batches for which a manually created schedule served as a standard of comparison. It requires 12 workers at maximum and lasted nearly 210 shifts, as shown in Fig. 3a. The number of shifts can be reduced to 123 (59%), if the upper limit for the human resource needs only to be adhered to, see Fig. 3b. This was achieved by a significant increase of the portion of labour time spent on work during a shift. If both, production time and the number of required workers are to be reduced, the best solution found is a reduction to 148 shifts (70%) and a maximum of 9 workers per shift (75%) as shown in Fig. 3c. This is equivalent to a reduction in man hours of 52% of the manual solution. Besides this task, others have been performed, including replanning, because of new orders and equipment failures. The major result of this is that plans of similar quality were produced in a shorter time compared to the initial planning. As this task has a lot of similarities to the scheduling of job sequences described by workflows and competing for resources we can expect a relevant benefit for optimised resource brokering.

4 Conclusion and Future Work

The presented contribution emphasises the necessity of planning and optimising resource allocation in a grid. With the concept of a global optimising resource broker GORBA, a system was presented, which can fulfil these planning tasks. It is based on the Evolutionary Algorithm HyGLEAM, the performance of which has already been approved in a number of applications [8, 9, 15, 19]. As an example, optimisation of a production planning task with GLEAM was described in this paper.

At the moment, it is worked on implementing a first prototype of GORBA, which will then be used to perform reference studies and benchmark tests. The reference studies will be aimed at analysing and evaluating the optimisation processes implemented in GORBA under different load conditions. Depending on the test results, a parallelisation of HyGLEAM may be considered to improve its performance and to allow fast replanning solutions.

In the next step, the optimisation options of GORBA are planned to be extended to cover data-related resources as well.

References

1. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in HPC Resource Management Systems: Queuing vs. Planning. Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP) at GGF8, Seattle, WA, USA, June 24, 2003, LNCS 2862, Springer Verlag, New York (2003) 1–20
2. Foster, I., Roy, A., Sander, V.: A Quality of Service Architecture that Combines Reservation and Application Adaption; Proc. of the 8th Intern. Workshop on Quality of Service, 2000
3. Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: Evaluation of Job-Scheduling Strategies for Grid Computing. Proceedings of 1st IEEE/ACM Int. Workshop on Grid Computing at 7th Int. Conf. on High Performance Computing (HiPC-2000), LNCS 1971, Springer Verlag, Berlin (2000) 191–202
4. Moreno, R.A.: Job Scheduling and Resource Management Techniques in Dynamic Grid Environments. (2002)
5. Fellows, D.K.: Abstraction of Resource Broker Interface. Deliverable D 2.4a/UoM, University of Manchester (2002)
6. Buyya, R.: Economic-based Distributed Resource Management and Scheduling for Grid Computing. Dissertation, Monash University, Melbourne, Australia (2002)
7. Roy, A., Livny, M.: Condor and Preemptive Resume Scheduling. In Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.): Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publisher (2003) 135–144
8. Jakob, W.: HyGLEAM - An Approach to Generally Applicable Hybridization of Evolutionary Algorithms. In: Merelo, J.J., et. al. (eds.): Proceedings of PPSN VII, LNCS, Vol. 2439. Springer Verlag, Berlin (2002) 527–536
9. Jakob, W.: Eine neue Methodik zur Erhöhung der Leistungsfähigkeit Evolutionärer Algorithmen durch die Integration lokaler Suchverfahren. Doctoral thesis, FZKA 6965, University of Karlsruhe (in German) (2004), see also: www.iai.fzk.de/˜jakob/HyGLEAM/
10. Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics for Scheduling Jobs on Computational Grids. Int. Conf. on Advanced Comp. and Communications (2000)
11. Halstenberg, S., Stucky, K.U., Süß, W.: A grid environment for simulation and optimization and a first implementation of a biomedical application. Proceedings of the OTM 2004 Workshops, Agia Napa, Cyprus, October 25-29, 2004, LNCS 3292, Springer Verlag, Berlin (2004), 59–67
12. Fischer, L. (Ed.): Workflow Handbook 2001, ISBN 0-9703509-0-2
13. http://www.globus.org
14. Blume, C.: GLEAM - A System for Intuitive Learning. In: Schwefel, H.P., Männer, R. (eds): Conf. Proc. of PPSN I. LNCS 496, Springer Verlag, Berlin (1990) 48–54
15. Blume, C., Jakob, W.: GLEAM - An Evolutionary Algorithm for Planning and Control Based on Evolution Strategy. In: Cantú-Paz, E. (ed): GECCO - 2002, Vol. Late Breaking Papers (2002) 31–38
16. Rosenbrock, H.H.: An Automatic Method for Finding the Greatest or Least Value of a Function. The Computer Journal, 3 (1960) 175–184
17. Box, M.J.: A New Method of Constrained Optimization and a Comparison with Other Methods. The Computer Journal, 8 (1965) 42–52
18. Schwefel, H.-P.: Evolution and Optimum Seeking. John Wiley & Sons, New York (1995)
19. Blume, C., Gerbe, M.: Deutliche Senkung der Produktionskosten durch Optimierung des Ressourceneinsatzes. atp 36, 5/94, Oldenbourg Verlag, München (in German) (1994) 25–29

A New Library for Evolutionary Algorithms*

Stanisław Gawiejnowicz, Tomasz Onak, and Cezary Suwalski

Adam Mickiewicz University,
Faculty of Mathematics and Computer Science,
Umultowska 87, 61-614 Poznań, Poland
{stgawiej, suwalski}@amu.edu.pl

Abstract. In the paper a new library for construction of evolutionary algorithms is presented. The library offers greater flexiblity than other known libraries of this type due to application of C# interfaces. The process of construction of an evolutionary program with the use of the library is discussed. The results of the practical evaluation of this library on a set of instances of the job shop problem are presented.

Keywords: C#; evolutionary algorithms; library; job shop scheduling.

1 Introduction

Many combinatorial optimization problems are NP-hard, [6]. This means that unless $P = NP$, no polynomial-time exact algorithms exist for solving such problems. However, since many real-life problems are NP-hard, we need effective tools for approximate solving of these problems in polynomial time. The *Evolutionary Algorithm* (EA in short, see Fig. 1) is an example of such a tool.

```
t ← 0;
initialization(P^0);   // base population
evaluation(P^0);
while (not stop_condition) do
    T^t ← preselection(P^t);   // temporary population
    O^t ← crossing_and_mutation(T^t);
    evaluation(O^t);
    P^{t+1} ← postselection(P^t, O^t);   // offspring population
    t ← t + 1;
end
```

Fig. 1. Pseudo-code of an evolutionary algorithm

Recall that an EA is a combination of a *Genetic Algorithm* (GA in short), *Genetic Programming* (GP), *Evolutionary Strategy* (ES) and *Evolutionary Programming* (EP), see [4, 10] for more details.

* The research has been partially supported by the Ministry of Science and Information Society Technologies of Poland, grant no. 4T11C 03925.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 414–421, 2006.

General form of EA. The simplest EA works on the *base*, *offspring* and *temporary* populations of *individuals*. In the *initialization* step of EA, the base population $\mathbf{P}^0$ is create in a random way. Next, *evaluation* of $\mathbf{P}^0$ is performed, i.e. for each individual from $\mathbf{P}^0$ a *fitness function* is calculated. In the main step of EA, the offspring population $\mathbf{P}^t$, $t > 0$, is created. First, in the process of *preselection*, the temporary population $\mathbf{T}^t$ is built from the best (with respect to the fitness function) individuals of $\mathbf{P}^t$. Next, individuals from $\mathbf{T}^t$ are *crossed* and *mutated*, which leads to the next offspring population $\mathbf{P}^{t+1}$. The process is continued until a certain *stop condition* is met (see Fig. 1).

Applications of EAs. Evolutionary algorithms have been successfully applied to solve such difficult problems as automatic generation of computer programs and electronic circuits [8] or the travelling salesman problem [10]. We refer the reader to [4, 10] for more details on applications of EAs.

Construction of EAs. Several approaches to the construction of evolutionary algorithms have been proposed. In one approach, we build a library of classes, which are next used for the construction of EAs. There are such libraries implemented in C++ [5] and Java [9]. To the best of our knowledge, such libraries in C# do not exist.

2 General Description of the TEAC Library

Our library, *TEAC* (*Toolbox for Evolutionary Algorithms in C#*), [11], has been created in order to give programmers a tool for construction of new evolutionary algorithms in a comfortable way. Great effort has been made in order to give the library expandability and effectiveness. At some extent the TEAC library has been influenced by the TEA library [5], written in C++. However, the MPI routines and C++ templates used in TEA made impossible a simple conversion of TEA into C# and a lot of code of the library had to be written anew.

Language. The TEAC library has been written in C# on Microsoft's .NET platform. We used this platform deliberately, although it is not as efficient as the native C++ language. However, C# offers greater flexibility, since the library can be used in connection with other languages of the .NET platform. Moreover, *Web services* which are accessible in .NET, can serve in future editions of the TEAC library as transport layer in parallel EAs.

Architecture. All objects defined in the TEAC library have been divided into *evolutionary processes* (i.e. the ones which use selection and other genetic operators, see Fig. 2) and *evolutionary materials* (i.e. chromosomes, genomes, individuals and populations, see Fig. 3). We extensively used C# *interfaces* in order to construct general evolutionary algorithms. The implementation rich in interfaces is the main feature of the TEAC library, compared to the above mentioned TEA library. These interfaces give the opportunity to expand the TEAC classes according to specific needs in an easy and comfortable way.

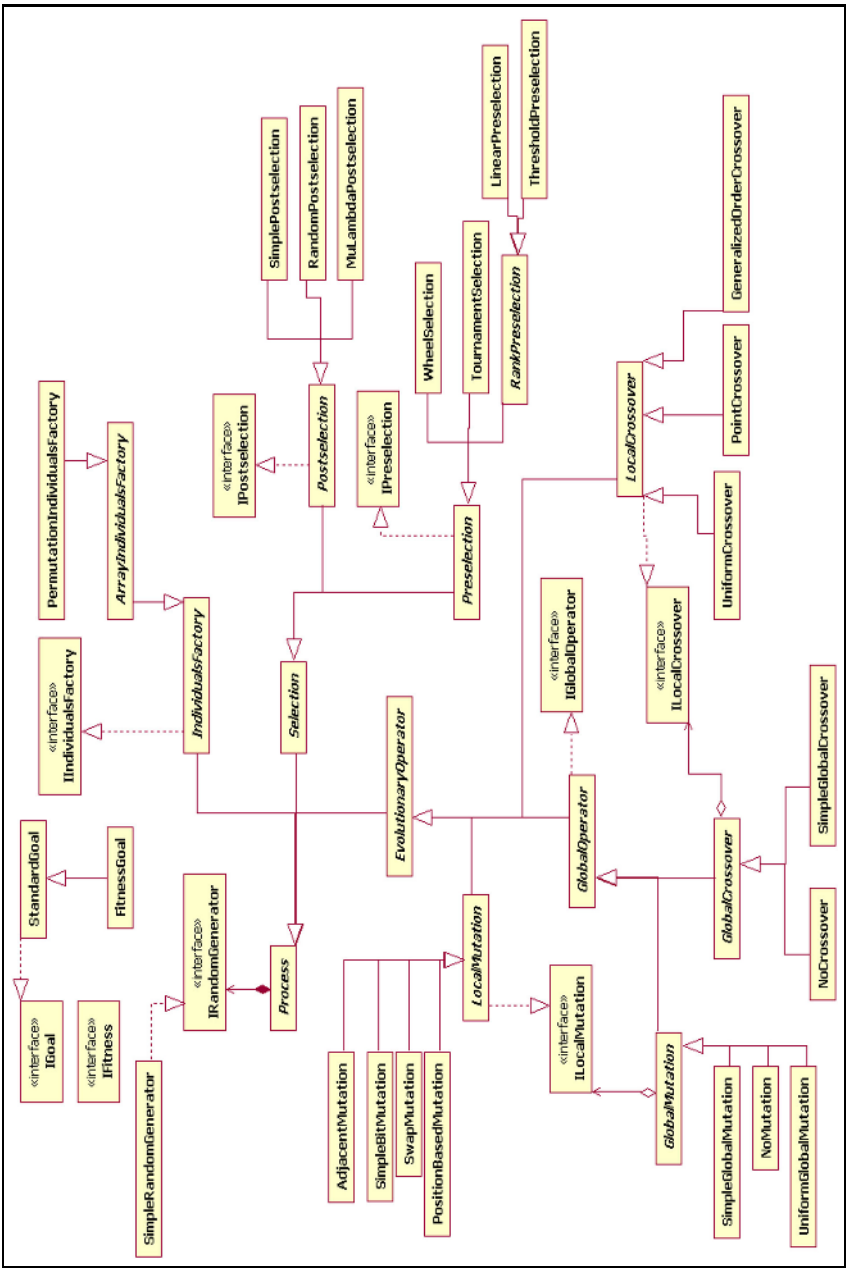


Fig. 2. Hierarchy of Evolutionary Processes Classes in the TEAC Library

Since in the construction of the library we used the *MVC* (*Model-View-Controller*) model, all objects and algorithms are separated from control and user interfaces. This make the structure of the library more transparent.

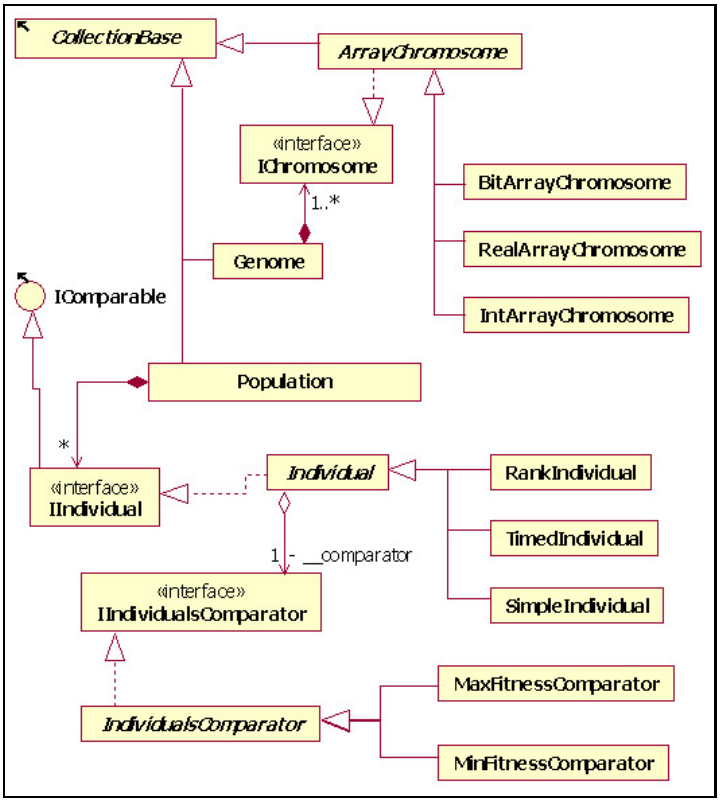


Fig. 3. Hierarchy of Evolutionary Materials Classes in the TEAC Library

Selection is represented by classes that inherit from `Preselection` and `Postselection` classes. We have implemented, among others, operators of proportional (rulette), tournament, rank linear and threshold selection and operators of trivial, random, $(\mu, \lambda)-$type and $(\mu + \lambda)-$type postselection.

Mutation and crossover operators. The TEAC library includes, among others, as operators of bit, replacing, adjacent and positional mutation as operators of multi-point, uniform and GOX (Generalized Order Crossover) crossover.

Genetic operators. Several genetic operators have been implemented in the TEAC library. They have been divided into *global* operators (acting on the whole population) and *local* operators (acting on particular individuals). This division allows the user of the library better tuning of particular operator parameters. All genetic operators are descendants of `EvolutionaryOperator` class. Crossover operators do not change either individuals or populations but only create new ones. Mutation operators may change individuals from the initial population or create new individuals depending on the value of `CloneMutated` property, defined in `LocalMutation` class (see Fig. 2).

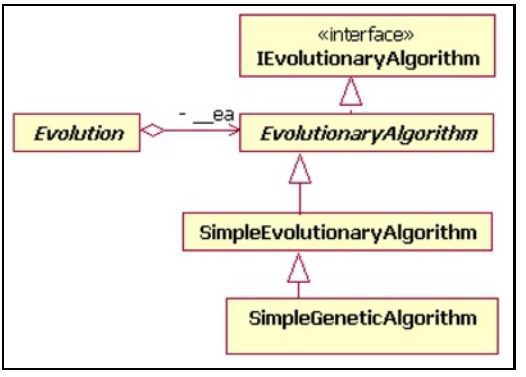


Fig. 4. Hierarchy of Evolutionary Algorithms Classes in the TEAC Library

Factory of individuals, represented by interface `IIndividualsFactory`, is the process that generates new individuals. The user of the TEAC library defines the above process by choosing an implementation of the factory interface, which allows to separate the process of the generation of new individuals from the starting mechanisms of their evolution.

Evolutionary materials are all objects representing the "parts" of evolving organisms or their groups. The objects that inherit from classes `Genome` and `Population` or from interfaces `IChromosome` and `IIndividual` belong here.

Population class operates on individuals interfaces only. Thanks to that, the TEAC library classes can be useful, even in the case of introducing new individuals, totally different from these implemented in TEA, [5]. The only requirement is the need of an implementation of `IIndividual` interface.

Algorithms. The TEAC library also delivers an implementation of example evolutionary algorithms. Classes `SimpleEvolutionaryAlgorithm` and `SimpleGeneticAlgorithm` define such algorithms. The `Evolution` class, playing the role of a monitor for evolutionary algorithms, has also been implemented. The classes of the TEAC library, specific to evolutionary algorithms, are presented in Fig. 4. We refer the reader to thesis [11] for more details on the TEAC library.

3 Computational Evaluation of the TEAC Library

We conducted two computational experiments in order to evaluate the quality of the TEAC library. In the first experiment, a set of instances of job shop problem was solved by an EA constructed using the TEAC library. In the second one, in a similar way, a set of instances of VLSI layout problem was solved. Due to space limitations, we describe only the results of the first experiment.

Problem formulation. The job shop scheduling problem is one of the most difficult problems in the scheduling theory. The two-machine case is NP-hard in the ordinary sense and the three-machine problem is NP-hard in the strong

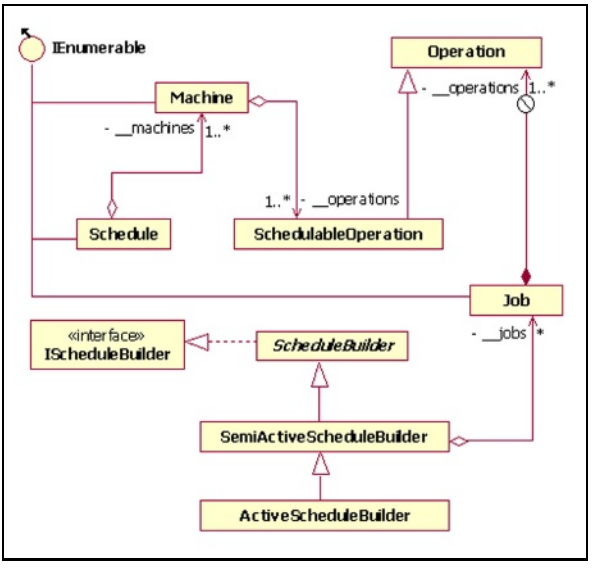


Fig. 5. Hierarchy of Classes in the EA for Job Shop Problem

sense, [3]. The problem can be formulated as follows. We are given a set of independent and nonpreemptable jobs J_j, $1 \leq j \leq n$, that have to be executed on machines $M_i, 1 \leq i \leq m$. Job J_j consists of operations $O_{jk}, 1 \leq k \leq j_k$, and operation O_{ji} has to be executed on machine M_i. The order of execution of operations of a particular job is fixed but it can be different for different jobs. The criterion of optimality of a schedule is the maximum completion time of all jobs. For more details on the problem we refer the reader to reviews [2, 7].

Representation of solutions. A solution of a scheduling problem is usually presented by a *Gantt chart*, [3]. The chart can be represented in an EA in two ways: by an *immediate* representation (a genome of an individual is a schedule itself) and an *intermediate* representation (an individual is decoded in some way in order to obtain a feasible schedule). The first representation, however, needs operators that are job-specific. Hence, we need a *schedule builder*, that "knows" how to build a schedule. In our experiment we applied *permutation with repetitions* (*pwr*) coding and we used an integral chromosome, implemented by `IntArrayChromosome` class. We used a builder of active schedules, defined by `ActiveScheduleBuilder` class (see Fig. 5), that decoded chromosomes into schedules (being objects of `Schedule` class), evaluated next by the `JobShopFitness` method.

Selection and crossing-over. In the experiment we used rank preselection, defined by `TournamentSelection` class, that returned twice bigger population than the initial one. The size of tournament was equal to 5.

For crossing we used GOX operator, implemented by `GeneralizedOrderCrossover` class. Postselection was random.

Mutation. Since we used *pwr* coding, there was a great number of possible mutation operators. We applied position based mutation operator (`PositionBasedMutation` class), order-based mutation operator (`SwapMutation` class) and adjacent two-job exchange mutation operator (`AdjacentMutation` class).

Algorithm. The schema of the EA was identical to the simple evolutionary algorithm. We saved the best current schedule, found by the algorithm defined by `SimpleEvolutionaryClass` class. The algorithm stopped after 2000 generations.

The hierarchy of the classes implemented in the EA used in the experiment for the job shop problem is given in Fig. 5.

Datasets. For tests we used data from FT06, FT10, FT20 and selected LA files, accessible in OR-Library [1]. The experiment was conducted on a PC with Athlon 2500+ CPU and Microsoft © Windows XP operating system.

Table 1. Results of Computational Experiment for Job Shop Problem

File	Size	OPT	AVR	Best	Time
FT06	6x6	55	55.0	55	40.50
FT10	10x10	930	979.1	955	130.44
FT20	20x5	1,165	1,228.8	1,202	199.44
LA01	10x5	666	666.0	666	66.02
LA16	10x10	945	972.9	956	141.70
LA20	10x10	902	914.7	907	141.08
LA21	15x10	1,046	1,111.6	1,097	288.40
LA25	15x10	977	1,032.5	1,019	253.66
LA28	20x10	1,216	1,302.2	1,286	412.35
LA29	20x10	1,152	1,266.9	1,248	409.20
LA39	15x15	1,233	1,320.3	1,291	384.13
LA40	15x15	1,222	1,312.4	1,288	385.36

Results. The solutions obtained by our EA for the job shop problem are presented in Table 1. Column `Size` gives the size of a particular instance, `nxm`, where `n` and `m` are the numbers of jobs and machines, respectively. Columns `OPT`, `AVR` and `Best` show the optimal, the average and the best found result, respectively. Column `Time` presents the running time of the EA (in seconds). All these results are average values of 10 independent runs of the algorithm.

Experiment conclusions. The experiment gave satisfactory results, though no particular effort has been made in order to tune the parameters of the applied EA. The schedules were close to optimal ones and were obtained in reasonable time. An average error of the obtained solutions was from 0% (FT06, LA01) to 9.97% (LA29). Since the size of the experiment was small, in the future we plan to investigate test instances of a greater size.

4 Conclusions

In the paper we presented a new library for construction of evolutionary algorithms, TEAC, implemented in C#. We described main features of the library and illustrated its application to solving a job shop scheduling problem.

The present version of the TEAC library is not the final one. We wish to conduct further research in two directions. First, we plan to prepare a parallel version of the library. The present version of our library can be parallelized only via multi-thread system calls offered by Windows environment. In order to achieve a real parallel library, we want to apply Web services, available on .NET platform. Second, we also wish to conduct more exhaustive experiments with better tuning of the applied EA parameters.

References

1. J.E. Beasley, OR-Library, see Web page `http://msmcga.ms.ic.ac.uk/info.html`
2. J. Blazewicz, W. Domschke and E. Pesch, The job-shop scheduling problem: Conventional and new solution techniques, *Euro. J. Optl Res.* **93** (1996), 1–33.
3. J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Węglarz, *Scheduling Computer and Manufacturing Processes*, 2nd ed., Springer 2001.
4. P. Calégari, G. Coray, A. Hertz, D. Kobler and P. Kuonen, A taxonomy of evolutionary algorithms in combinatorial optimization, *Journal of Heuristics*, **5** (1999), 145–158.
5. M. Emmerich and R. Hosenberg, TEA - a C++ library for the design of evolutionary algorithms, report CI-106/01, SFB 531, University of Dortmund, 2001.
6. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.
7. A.S. Jain and S. Meeran, Deterministic job-shop scheduling: Past, present and future, *Euro. J. Optl Res.* **113** (1999), 390–434.
8. J.R. Koza, Genetic programming, in: J.G. Williams and A. Kent (eds.), Encyclopaedia of Computer Science and Technology, Dekker, 1998, 29-43.
9. S. Luke et al, A Java-based evolutionary computation and genetic programming research system, see Web page `http://cs.gmu.edu/~eclab/projects/ecj/`
10. Z. Michalewicz, Genetic algoritms + data structures = evolutionary programs, Springer, 1994.
11. T. Onak, *Evolutionary algorithms and their selected applications*, M. S. Thesis, Faculty of Mathematics and Computer Science, Adam Mickiewicz University, 2004 (in Polish).

Grid-Based Evolutionary Optimization of Structures

Wacław Kuś[1] and Tadeusz Burczyński[1,2]

[1] Department for Strength of Materials and Computational Mechanics,
Silesian University of Technology, Konarskiego 18a, 44-100 Glwice, Poland
{waclaw.kus, tadeusz.burczynski}@polsl.pl
[2] Institute for Computer Modelling,
Cracow University of Technology Cracow, Poland

Abstract. The paper is devoted to computational grids applications in evolutionary optimization of mechanical structures. The LCG2 and UNICORE grid middleware are used. The optimization is performed by means of the distributed evolutionary algorithm. The fitness function is computed using the finite element method. The numerical example is presented in the paper.

1 Introduction

The shape optimization of structures can be solved using methods based on sensitivity analysis information or non-gradient methods based on genetic algorithms [13]. Applications of evolutionary algorithms in optimization need only information about values of an objective (fitness) function. The fitness function is calculated for each chromosome in each generation by solving the boundary - value problem by means of the Finite Element Method (FEM)[8][17]. This approach does not need information about the gradient of the fitness function and gives the great probability of finding the global optimum. The main drawback of this approach is the long time of calculations. The applications of the distributed evolutionary algorithms [15] can shorten the time of calculations[9][1][2][3][4].

The computational grids enable to use distributed computational resources. The authorization is one of the most important elements of grids. The Public Key Infrastructure is used in most grid projects. The Virtual Organizations (VO) created by people with similar interests or working on similar projects allow to create grids and share resources.

The use of grid techniques in optimizations can lead to improvements in hardware and software utilization. The other advantages of grids are simple and uniform end user communication portals/programs. The first evolutionary optimization tests [10] were performed using Condor package[5]. The plugins and programs for evolutionary optimization of structures using UNICORE environment[16] were presented in [11]. The application of LCG middleware[12] and Crossgrid[6] project resources is presented in the paper.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 422–429, 2006.

2 Optimization of Structures Using the Distributed Evolutionary Algorithm

Sequential genetic and evolutionary algorithms are well known and applied in many areas of optimization problems. The main disadvantage of these algorithms is the long time needed for computation. The distributed evolutionary algorithms (DEA) work similarly to many evolutionary algorithms operating on subpopulations. The evolutionary algorithms exchange chromosomes during a migration phase between subpopulations. When DEA is used the number of fitness function evaluations can be lower in comparison with sequential and parallel evolutionary algorithms. DEA work in the parallel manner, usually. Each of the evolutionary algorithms in DEA works on a different processing unit. The theoretical reduction of time could be bigger then the number of processing units. The flowchart of the distributed evolutionary algorithm for one subpopulation is presented in Fig. 1. The sample DEA with four subpopulations is shown in Fig. 2. The starting subpopulation of chromosomes is created randomly. The

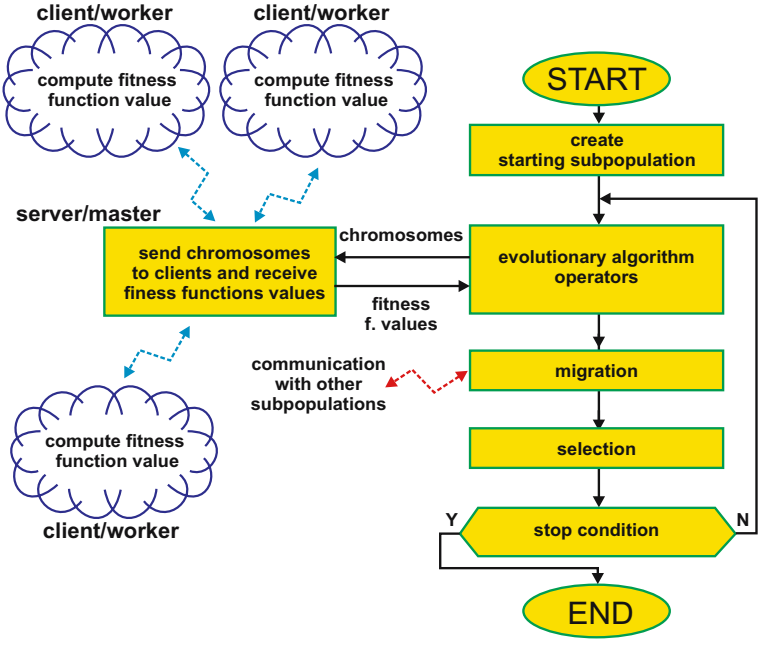


Fig. 1. The flowchart of the distributed evolutionary algorithm for one subpopulation

evolutionary operators change chromosomes and the fitness function value for each chromosome is computed. The migration exchanges a part of chromosomes between subpopulations. The selection decides which chromosomes will be in the new population. The selection is done randomly, but the fitter chromosomes have bigger probability to be in the new population. The selection is performed on

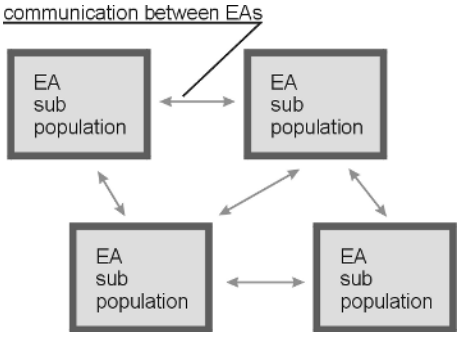


Fig. 2. DEA with 4 subpopulations

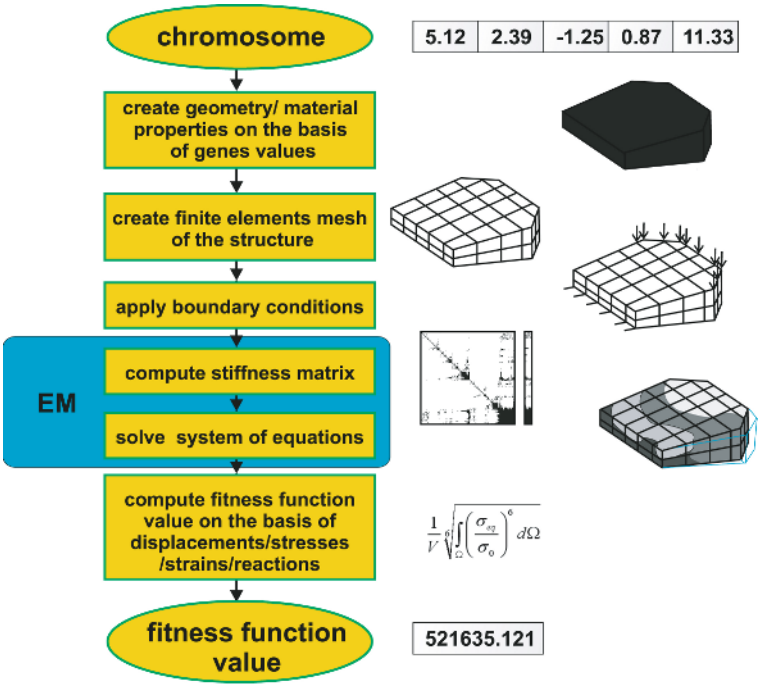


Fig. 3. Fitness function evaluation using FEM

chromosomes changed by operators and immigrants. The next iteration is performed if the stop condition is not fulfilled. The stop condition can be expressed as a maximum number of iterations.

Computation of the fitness function in optimization problems is performed by means of results of the FEM analysis. The genes describe the shape, material properties, topology of the structure. The structure is meshed and proper

boundary conditions are applied before FEM analysis. The flowchart of fitness function evaluation is presented in Fig. 3.

3 Evolutionary Optimization Using Grid Based on LCG Middleware

The goal of the LCG project [12] is to create middleware (based on Globus Toolkit) which allows to create big grids. The LCG project is connected with Large Hadron Collider project realized in CERN. Many European grid projects use LCG as software basis, for example Crossgird [6], EGEE [7]. The grids consist of user interface (computer for submitting, monitoring jobs), resource broker (computer which authorizes users, transfers files across grid, decides which resources will be used by user), gatekeepers (computers which translate the resource brokers job requests into working nodes job requests), working nodes (computers executing jobs) and storage elements (computers allowing to high performance access to storage data). The computer elements of the grid are distributed in many sites. The resource broker decides on the basis of job description provided by the user, current sites load and virtual organization policies, which computing elements should be used. The communication between computational sites and user are performed using resource broker, also jobs monitoring and fetching jobs results.

The simplest way to use such grids is to submit the evolutionary optimization job. The Crossgrid project testbed enables to use MPICH [14] jobs. The distributed evolutionary algorithm can be implemented using MPICH library. There is submition of one job for one optimization problem.

4 UNICORE EAOPT Plugin

The UNICORE environment enables to perform computational tasks with use of computers without deep knowledge about target computers operating systems, directory structure etc. The UNICORE client module is written in Java and can be used on most computer systems currently available. The client is very flexible and prepared to use third party plugins. The special classes to be used in plugins like file browsers are defined in UNICORE. The communication between client module and the target system can be performed with the use of predefined classes.

The plugin proposed in the paper can be applied to prepare the evolutionary optimization job. The parameters of the distributed evolutionary algorithm like the number of chromosomes, genes, subpopulations, probabilities of operators, constraints on design variables can be loaded using plugin. The EAOPT is also responsible for transferring files to target computer and the execution of job. The transfer of output files is also performed after computations.

The EAOPT plugin is shown in Fig. 4.

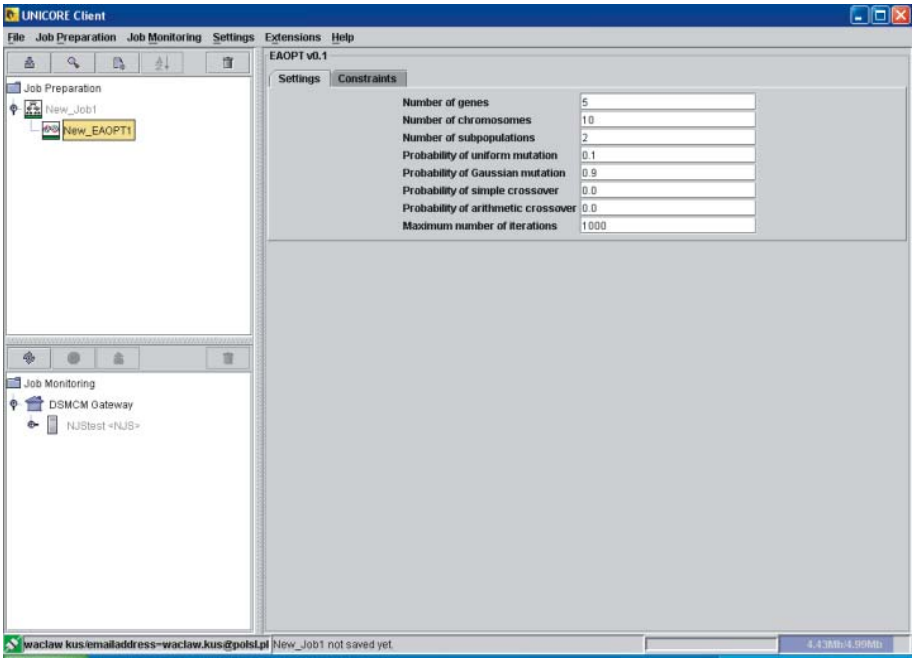


Fig. 4. The EAOPT plugin

5 Numerical Test Performed Using LCG2

The minimization of the mean equivalent stresses in a plate is considered. The plate is loaded using two load schemas. The fitness function is expressed as a sum of mean equivalent stresses for two load cases:

$$F = \frac{1}{V}\int_{\Omega_1} \sigma_{eq} \mathrm{d}\Omega_1 + \frac{1}{V}\int_{\Omega_2} \sigma_{eq} \mathrm{d}\Omega_2 \tag{1}$$

where V is a volume of the plate, σ_{eq} means equivalent Huber-Mises stresses, Ω_1 is the plates area for the first load case, Ω_2 is the plates area for the second load case. The chromosome contains 6 genes (g_0-g_6) and describes shape of the plate as shown in Fig. 5.

The LCG grid environment and CrossGrid testbed were used during evolutionary optimization. The MPICH version of the coevolutionary algorithm was used (the tests were performed using two processors, the location of the used clusters were chosen by the resource broker). The parallelization of evolutionary algorithms can achieve good efficiency when low number of processors are used. The speedup near 2 was obtained using two processors. The best found result is shown in Fig. 6. The distribution of equivalent stresses for both load cases is presented. The $Eq.stress$ means equivalent stress value for contour number No.

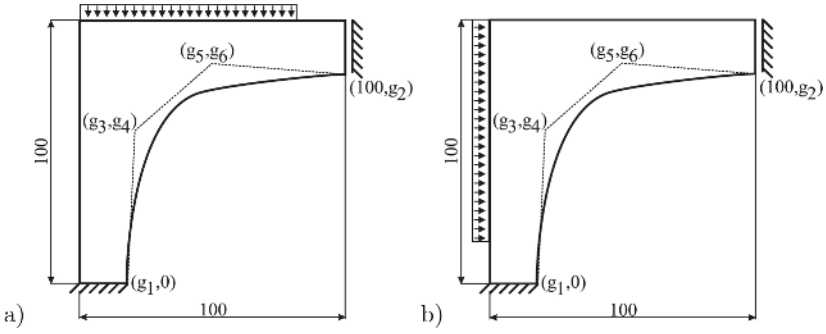


Fig. 5. The geometry of the plate, a) first load case, b) second load case

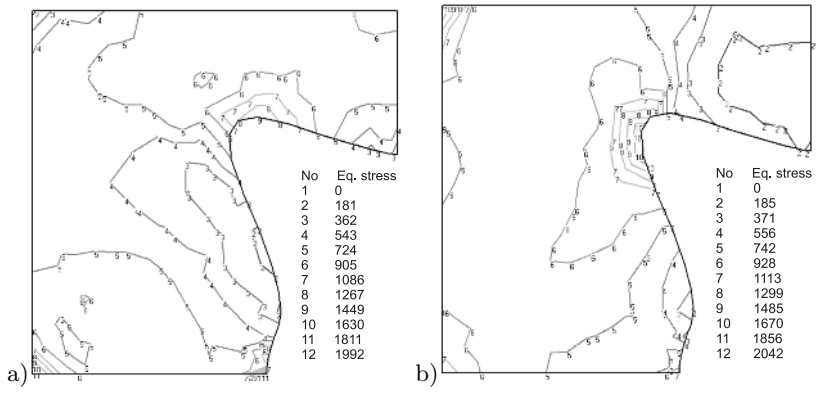


Fig. 6. The best result, a) equivalent stresses map for best found chromosome for the first load case, b) for the second load case

6 Numerical Test Performed Using UNICORE Plugin EAOPT

The goal of the test is to solve an identification problem. The identification problem can be expressed as an optimization problem. The aim of the identification is to find the position of the center and the radius of a void in a plate on the base of measured displacements.

The geometry of the plate is presented in Fig. 7. The plate is made from elasto-plastic material. The displacements are measured in sensor points (Fig. 7).

The fitness function is expressed as:

$$F = \sum_{i=1}^{n} |u_i - \hat{u}_i| \tag{2}$$

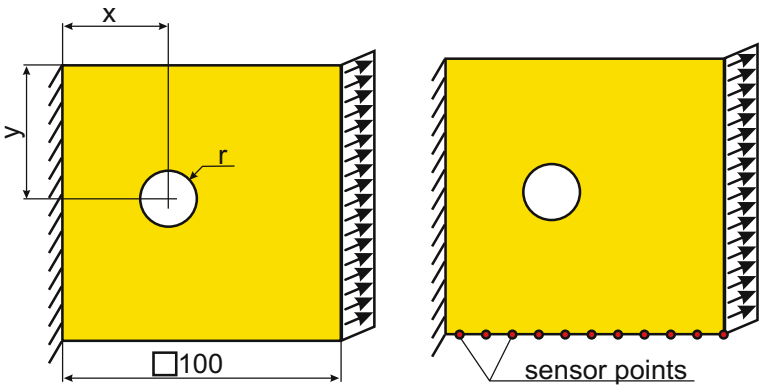


Fig. 7. The geometry of the plate

where n is the number of sensor points, u_i are equivalent displacements in i-th sensor point and $\hat{u}_i$ are equivalent displacements in $i-th$ point computed using the FEM for current chromosome.

The parameters of the distributed evolutionary algorithm are: the number of subpopulations - 2, the number of chromosomes -10, the number of variables - 3. The gaussian mutation and simple crossover operators were used. The genes of the chromosomes contains floating point values.

The results (Table 1) in the last iteration are very close to the searched one.

Table 1. The results of identification

design variable	exact value	value for the best chromosome in the first iteration	error in the first iteration	value for the best in last iteration	error in last iteration
x	35	60.431	72.6%	35.001	0.005%
y	50	48.035	3.929%	48.980	2.040%
r	10	10.064	0.646%	10.064	0.646%

7 Conclusions

The coupling of distributed evolutionary algorithm, finite element method and computational grid creates modern, powerful and efficient structures optimization tool. The evolutionary computation using grid environments opens new possibilities. The access to powerful computational distributed resources enables to perform computationally intensive jobs. The one time login and use of resource broker allow simple access to many clusters in virtual organization. The location and performance of the resources are not need to be known before submitting jobs.

Acknowledgement

The research is financed from the Polish science budget resources in the years 2005-2008 as the research project.

References

1. Burczyński T., Kuś W., Distributed evolutionary algorithms in shape optimization of nonlinear structures. Lectures Notes on Computational Science 2328, Springer, 2002.
2. Burczyński T., Kuś W., Długosz A., Poteralski A., Szczepanik M., Sequential and Distributed Evolutionary Computations in Structural Optimization. Lecture Notes on Artificial Intelligence 3070, Springer, 2004.
3. Burczyński T., Kuś W., Długosz A., Orantek P., Optimization and defect identification using distributed evolutionary algorithms, Engineering Applications of Artificial Intelligence, No. 4, Vol. 17, pp. 337-344, 2004.
4. T. Burczyński, W. Kuś, Optimization of structures using distributed and parallel evolutionary algorithms Parallel Processing and Applied Mathematics, PPAM2003, Revised papers, Lecture Notes on Computational Sciences 3019, Springer, pp. 572-579, 2004.
5. Condor, High Throughput Computing, http://www.cs.wisc.edu/condor/
6. Crossgrid project home page, http://www.crossgrid.org
7. EGEE Enablig Grids for E-Science in Europe home page, http://www.eu-egee.org
8. M. Kleiber (red.), Handbook of Computational Solid Mechanics, Springer- Verlag, 1998.
9. Kuś W., Burczyński T., Evolutionary optimization of elasto-plastic solids. In: Methods of Artificial Intelligence in Mechanics and Mechanical Engineering (eds. T.Burczyski and W.Cholewa), Gliwice, 2000.
10. W. Kuś, T. Burczyński, Computer implementation of the coevolutionary algorithm with Condor scheduler, KAEIOG 2004, Kazimierz, 2004.
11. W. Kuś, T. Burczyński, Distributed evolutionary algorithm plugin for UNICORE system, Proc. 4th Cracow Grid Workshop, Cracow, 2004.
12. LCG project home page, http://lcg.web.cern.ch/LCG/default.htm
13. Z. Michalewicz, Genetic algorithms + data structures =evolutionary algorithms. Springer-Verlag, Berlin, 1996.
14. MPICH project home page, http://www-unix.mcs.anl.gov/mpi/mpich/
15. R. Tanese, Distributed Genetic Algorithms. Proc. 3rd ICGA, pp.434-439, Ed. J.D. Schaffer. San Mateo, USA, 1989.
16. UNICORE Plus Final Report - Uniform Interface to Computing Resources, Joint Project Report for the BMBF Project UNICORE Plus, Grant Number: 01 IR 001 A-D, 2003.
17. O. C. Zienkiewicz, R. L. Taylor, The Finite Element Method. The Basis, Vol. 1-2, Butterworth, Oxford, 2000.

Parallelizing Evolutionary Algorithms for Clustering Data*

Wojciech Kwedlo

Faculty of Computer Science, Białystok Technical University,
Wiejska 45a, 15-351 Białystok, Poland
wkwedlo@ii.pb.bialystok.pl
http://aragorn.pb.bialystok.pl/~wkwedlo

Abstract. In the paper the problem of using an evolutionary algorithm to partition a dataset into a known number of clusters is considered. A novel approach, based on data decomposition, for parallel computing of the fitness function is proposed. Both the learning set and the population of the evolutionary algorithm are distributed among processors. Processors form a pipeline using the ring topology. In a single step each processor computes the local fitness of its current subpopulation while sending the previous subpopulation to the successor and receiving next subpopulation from the predecessor. Thus it is possible to overlap communication and computation using non-blocking MPI routines. Our approach to parallel fitness computation was applied to differential evolution algorithm. The results of initial experiments show, that for large datasets the algorithm is capable of achieving very good scalability.

1 Introduction

The aim of clustering [7] is to partition a set of patterns, called the *learning set*, into homogenous and disjoint groups called clusters. Clustering has many important applications in scientific research. One of most commonly used clustering techniques is *K–means* algorithm [7, 2]. It is easy to implement and computationally efficient. However its main drawback is the possibility of being trapped in a local optimum.

In recent years many clustering methods based on *evolutionary algorithms* (EAs) have been proposed [6, 9, 11]. EAs [10] are stochastic search techniques inspired by the process of biological evolution. Unlike local optimization methods e.g. K–means they simultaneously process a *population* of problem solutions, which gives them the ability to escape local optima. However this ability comes at the expense of very high computational complexity. This problem is especially important in clustering applications where evaluation of each solution requires

* This work was supported by the Ministry of Scientific Research and Information Technology, Poland, under the project 6 T11 2003 C/06098 "Clusterix - National Cluster of Linux Systems". All computational experiments were performed on the cluster, built in the framework of this project, at Częstochowa University of Technology.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 430–438, 2006.

reading the whole learning set. A possible method for alleviating this drawback is a parallel implementation of an evolutionary algorithm [1].

The main contribution of the paper is a novel method for parallel computing of the fitness function of the EA population. Although we have tested the method on the clustering problem it can be applied to other learning problems (e.g supervised learning of decision tree and neural network classifiers [2]), which have similar structure. In this paper the method is used to speed up a *differential evolution* algorithm (DE) [14]. DE is a relatively new evolutionary algorithm, which requires the representation of solution by real-valued vectors. It has been successfully applied to many difficult optimization problems [12]. According to our knowledge only one application of DE (based on sequential computing) to the clustering problem has been proposed [11].

The paper is organized as follows. In the next section the differential evolution algorithm is presented. Section three describes the problem of a clustering a dataset into a known number of partitions and shows, how evolutionary algorithms can be applied to that problem. The next section presents the method for parallelization of the algorithm. Some initial experimental results concerning scalability of the algorithm are presented in the section five. The last section concludes the paper.

2 Differential Evolution Algorithm

In this section the most popular *DE/rand/1/bin* differential evolution method is presented. For the more detailed description the reader is referred to [14].

Like all evolutionary algorithms, differential evolution maintains a population $U = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_S\}$ of S solutions to the optimization problem. Usually each solution takes the form of a D-dimensional real-valued vector, i.e. $\mathbf{u}_i \in \mathcal{R}^D$. At the begin members of the population are initialized randomly. The algorithm advances in *generations*. Each generation involves three consecutive phases: *reproduction* (creation of a temporary population), computing of the objective function (called the *fitness* in the EA terminology) of all members of the temporary population, and *selection*.

Reproduction in differential evolution creates a temporary population $Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_S\}$ of *trial vectors*. For each solution $\mathbf{u}_i$ a corresponding trial vector $\mathbf{y}_i$ is created. Each element $y_{i,j}$ (where $j = 1 \ldots D$) of the trial vector $\mathbf{y}_i$ is generated as:

$$y_{i,j} = \begin{cases} u_{a,j} + F * (u_{b,j} - u_{c,j}) & \text{if } \texttt{rnd()} < CR \\ u_{i,j} & \text{otherwise} \end{cases}.$$

In the above expression $F \in [0, 2]$ is a user supplied parameter called *mutation coefficient*. $a, b, c \in 1, \ldots, S$ are randomly selected in such way, that $a \neq b \neq c \neq i$. `rnd()` denotes a random number from the uniform distribution on $[0, 1)$, which is generated independently for each j. $CR \in [0, 1]$ is another user supplied parameter called *crossover factor*. The parameters F and CR influence convergence

speed and robustness of optimization process. The choice of their optimal values is an application dependent task [14]. In our experiments we used $CR = 0.01$ and $F = 0.5$.

The remaining two phases of DE generation are computation of the fitness for all members of the trial population Y and selection. Selection in differential evolution is very simple. The fitness of each trial solution $\mathbf{y}_i$ is compared to the fitness of the corresponding original solution $\mathbf{u}_i$. The trial vector replaces the original in U if its fitness is better. Otherwise the trial vector is discarded.

3 Clustering with Evolutionary Algorithms

Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ be a N–dimensional *learning set* of M objects, such as $X \subset \mathcal{R}^N$. The elements of X are called *feature vectors*. The problem of clustering into K groups (clusters) can be defined as the search for the optimal partition $G^* = \{C_1, C_2, \ldots, C_K\}$, (where $\forall_{i \neq j} C_i \cap C_j = \emptyset$, $\bigcup_{i=1}^{K} C_i = X$), such as:

$$G^* = \arg\min_{G} f(X, G),$$

where $f(X, G)$ is a *criterion* function. There are many possible criterion functions described in the literature [7]. Among them a commonly used one is the *square error* (SE) [7]. It can be defined as:

$$SE(X, G) = \sum_{i=1}^{K} \sum_{\mathbf{x}_j \in C_i} d^2(\mathbf{x}_j, \mathbf{m}_i), \tag{1}$$

where d^2 is a squared distance (e.g. Euclidean), $\mathbf{m}_i$ is the center of the cluster C_i, which can be expressed as:

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j.$$

In order to apply differential evolution approach to the above problem, we have to work out a method for encoding partitions of the learning set by real-valued vectors. The most natural idea [6, 9, 11] consists in representing each cluster by a *prototype* vector. Each feature vector is assigned to the cluster represented by the closest prototype. The solution of the clustering problem takes the form of a vector $\mathbf{u} \in \mathcal{R}^{KN}$, which can be described as a composition of K prototypes:

$$\mathbf{u} = [\mathbf{u}^{(1)}; \mathbf{u}^{(2)}; \ldots; \mathbf{u}^{(K)}],$$

where $\mathbf{u}^{(i)} \in \mathcal{R}^N$ is the prototype of the i-th cluster. The square error criterion can be reformulated as:

$$SE(X, \mathbf{u}) = \sum_{i=1}^{M} \min_{j=1 \ldots K} d^2(\mathbf{x}_i, \mathbf{u}^{(j)}). \tag{2}$$

The above approach allows us to use the DE algorithm to directly solve clustering problem by using function (2) as the fitness and taking the problem dimension $D = N * K$. The sole remaining issue is the initialization of the population U. In our experiments we used a simple method, in which for each $\mathbf{u} \in U$ randomly chosen feature vectors from X were assigned to cluster prototypes $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(K)}$.

4 Parallelization of the DE for the Clustering Problem

As equation (2) shows, the computational complexity of computing fitness function for all solutions in the trial population Y is $\mathcal{O}(KNSM)$. This is the most expensive phase of the DE generation, because the complexity of the other phases i.e. selection and reproduction is not greater than $\mathcal{O}(KNS)$ and the number of feature vectors M can be vary large in practical applications. Efficient parallelization of the fitness computation is crucial to achieving good scalability.

4.1 Computation of the Fitness

The most natural approach to parallelizing fitness computation makes use of data decomposition. If the learning set X is evenly partitioned into P subsets $X_1, X_2, \ldots, X_P$, such as $\bigcup_{i=1}^{P} X_i$ and $\forall_{i \neq j} X_i \cap X_j = \emptyset$ than the fitness (2) of a trial vector $\mathbf{y}_i$ can be expressed as:

$$SE(X, \mathbf{y}_i) = \sum_{i=1}^{P} SE(X_i, \mathbf{y}_i),$$

In the remainder of the paper by the *local fitness* we will call the fitness computed using one of the subsets $X_1, X_2, \ldots, X_P$. The fitness computed using the whole learning set X will be called the *global fitness*.

One possible method for parallelization [8] uses a *master–slave* model. Each subset X_i is placed in a separate slave processor. To compute the global fitness the master processor broadcasts the population to the slaves. Each slave computes its local fitness function for each population member. The vectors containing local fitness values are send back to the master, which adds them up getting the global fitness.

The advantage of the master-slave method is the simplicity of the implementation of selection and reproduction. They are executed sequentially on the master processor. However this simplicity limits scalability of the method. Moreover, during the communication the master and the slaves are idle, which limits scalability further.

In this paper we propose a new method for parallelization of the fitness computation, which is based on the *pipeline* approach. In this method the learning set is, like in master-slave approach, evenly distributed among the processors. Additionally the population Y of trial vectors is evenly partitioned into subpopulations $Y_1, Y_2, \ldots, Y_P$. Each subpopulation is initially placed in a single processor. Figure 1 shows the partition of the learning set X and an initial partition of

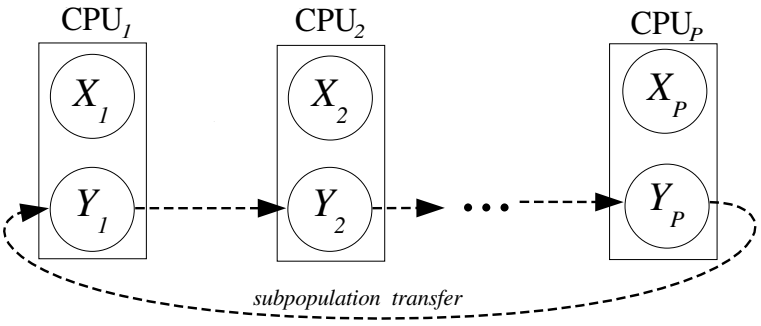


Fig. 1. Initial data decomposition and direction of subpopulation transfer

```
SubPopulation LocalCompute, LocalSend, LocalReceve;
double LocalFitness[S];
LocalCompute=Yj; LocalSend=Yj;
for (int i=0; i<P-1; i++) {
     MPI_Request rq1,rq2;
     MPI_ISEND(NEXT_CPU_IN_RING,&rq1,LocalSend);
     MPI_IRECV(PREV_CPU_IN_RING,&rq2,LocalReceve);

     ComputeLocalFitness(LocalCompute,LocalFitness);
     MPI_WAITALL(&rq1,&rq2);

     SendPop=LocalCompute;
     LocalCompute=LocalReceive;
}
ComputeLocalFitness(LocalPop);
```

Fig. 2. Pseudocode for parallel computation of the fitness for the processor j

the trial population Y into P processors of a parallel computer. Each subset X_j remains assigned to the processor CPU_j during the execution of the algorithm. Processors form a ring structure, as shown on Figure 1. The computation of fitness is performed in P steps. In a single step a processor computes a local fitness of its current sub-population, sends the previous sub-population to its successor in the ring and receives the next sub-population from its predecessor. An important advantage of this approach is the possibility of performing communication and computation simultaneously using nonblocking MPI operations [13]. The pseudocode for such implementation is shown on Figure 2. Functions `MPI_ISEND` and `MPI_IRECV` start non-blocking send and receive operations respectively. `ComputeLocalFitness` computes the local fitness of the current trial subpopulation stored in `LocalCompute` and places local fitness in proper position in the `LocalFitness` array. `MPI_WAITALL` waits for the completion of send and receive operations.

After execution of the algorithm shown on figure 2 each processor received all the subpopulations which form the trial population and computed the local fitness for them. Thus the local fitness of all the members of the distributed trial population is stored in the `LocalFitness` vector. To compute the global fitness `LocalFitness` vectors from all P processors have to be added. In order to perform this addition all processors call the `MPI_ALLREDUCE` [13] collective communication routine. This routine uses an $\mathcal{O}(\log P)$ algorithm to compute the sum and sends it to all P processors.

If we omit the cost of the `MPI_ALLREDUCE`, the above method has a potential for achieving linear speedup. Although it requires P computation steps, each step is speeded up P^2 times because booth the size of the learning set and the size of the population are reduced P times. Moreover using non-blocking operations allows us to hide the cost of communication.

4.2 Reproduction and Selection

The selection and the reproduction are performed all follows. Each processor keeps a local copy of the current population U. During the reproduction phase processor j generates, according to (2), only its initial sub-population Y_j. Because $|Y_j| = |Y|/P = S/P$ the reproduction phase is speeded up P times. During computation of the fitness each processor stores incoming trial sub-populations $Y_1, Y_2, \ldots, Y_P$. After the computation of the fitness by the algorithm described in the previous subsection, the processor was visited by each sub-population, which means that it now can reconstruct the complete population Y of trial vectors. Moreover the processor has global fitness values for all the elements of Y, obtained by executing `MPI_ALLREDUCE` routine. Having Y, fitness of all elements of Y, and from the previous generation U with corresponding fitness values each processor is able to perform the selection and update U. Identical selection is performed by each processor independently. Hence the selection is the only phase, which is not speeded up in comparison with the sequential version of DE.

5 Experimental Results

In this section some initial experimental results are presented. Previous studies have shown that evolutionary algorithms are easily able to find good quality solutions for the clustering problem (see e.g. [6, 9], and especially [11] for the results obtained by the DE). Therefore in this paper we focus on scalability of the new parallel method.

In all the experiments we used a cluster of seven SMP servers connected by a Gigabit Ethernet network. Each server in the cluster is equipped with two Itanium II 1.4GHz CPUs with 1MB L3 cache and 2GB of RAM. The servers run Debian operating system based on Linux kernel version 2.6 The DE algorithm was implemented in C++ language and linked to MPICH version 1.2 [5] MPI library.

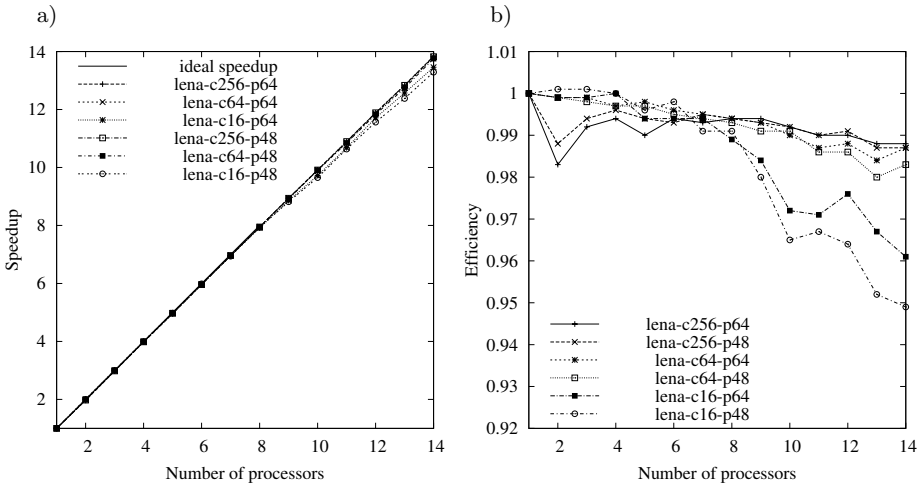


Fig. 3. Computational speedup a) and efficiency b) for the *lena* dataset

For initial experiments we used a dataset derived from the well-known *lena* image, which is a standard image processing benchmark. The learning set was generated by converting each pixel from 128x128 pixel (16384 pixels total) version of the image to a three–dimensional feature vector containing RGB values in the range $[0, 255]$. In this image processing context the problem of clustering is called the *vector quantization* [3].

As a basis for calculating speedup the average time of a single generation was used. This time was estimated by running the DE for 100 generations, then by dividing the total computation time by the number of generations. The clustering algorithm was run with varying sizes of the DE population and with varying numbers of clusters. The speedup obtained in the experiments is shown on the figure 3a. Each line labelled as *"lena–cx–py"* corresponds to one experiment, in which the number of clusters was equal x and the size of the population equal y. The choice of x and y allowed us to test the algorithm using a very wide range of granularity of the communication pattern. This granularity can be measured by the length of the shortest interval between two consecutive message exchanges. As figure 2 shows this length can be roughly approximated by the time of single generation divided by the number of processors P (if $P > 1$ there are $P - 1$ evenly spaced message exchanges in a generation). The most extreme values of this metric are 3.2 seconds (for lena-c256-p64 on 2 processors) and 3.3 milliseconds (for lena-c16-p48 on 14 processors).

Because all the curves on figure 3a are close to the optimal linear speedup we have also calculated the *efficiency*, which is defined [4] as the ratio of achieved speedup to the ideal linear speedup. The efficiency is plotted on figure 3b. The plot indicates lower efficiency for cases, in which communication pattern is more fine-grained (i.e. small number of clusters and many processors). However, even in the worst case the efficiency is higher than 0.9.

6 Conclusions and Future Work

In the paper a new method for parallel computation of the fitness function of EA was proposed. Although we have tested the method on clustering problem using differential evolution algorithm, it can be applied to other learning problems as well. Preliminary experimental results show, that for large datasets our approach is able to achieve near linear scalability.

The most important drawback of our approach is the limitation of maximum speedup by the size of EA population. This problem can be tackled by using a two-level hierarchical parallelization. At the lower level a single parallel DE algorithm described in this paper could be employed. At the higher level multiple DE algorithms could be executed using the *parallel island* [1] model. In this model multiple evolutionary algorithms execute independently in separate islands. However, from time to time a solution is able to migrate ("swim") from one island to another. Booth experimental and theoretical studies show [1], that the island model of multiple EAs offers faster convergence in comparison with the single EA.

The above two-level schema can be easily mapped into a *metacluster* consisting of several geographically distributed local clusters connected by a wide area network (WAN). Each cluster could serve as an island for one parallel EA. The migration of solutions would take place across WAN. Because the communication pattern of migration is very coarse grained (i.e. the migration takes place after many consecutive generations), the higher latency of WAN would not impair the scalability of the EA. In the near future we are going to implement the two–level method and test it in the Polish national metacluster built in the framework of the Clusterix project [15].

References

1. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 6(5):443–462, 2002.
2. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2001.
3. A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, 1992.
4. A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, Second Edition, 2003.
5. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing standard. *Parallel Computing*, 22: 789–828, 1996.
6. L. O. Hall, B. Ozyurt, and J. C. Bezdek. Clustering with a genetically guided optimized approach. *IEEE Trans. Evolutionary Computation*, 3(2):103–112, 1999.
7. A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
8. W. Kwedlo. A parallel evolutionary algorithm for discovery of decision rules. In *Lecture Notes in Computer Science 3019*. Springer Verlag, 2004.
9. U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33:1455–1465, 2000.

10. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1996.
11. S. Paterlini and T. Krink. High performance clustering with differential evolution. In *Proceedings of the Sixth Congress on Evolutionary Computation (CEC-2004)*, pages 2004–2011. IEEE Press, 2004.
12. K. Price and R. Storn. Minimizing the real functions of the ICEC'96 optimization contest by differential evolution. In *IEEE International Conference on Evolutionary Computation ICEC'96*, pages 842–844, 1996.
13. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1996.
14. R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.
15. R. Wyrzykowski, N. Meyer, and M. Stroiński. CLUSTERIX: National cluster of Linux systems. In *Across GRIDS 2004*, Nicosia, Cyprus, 2004.

Evolutionary Adaptation in Non-stationary Environments: A Case Study

Andrzej Obuchowicz and Dariusz Wawrzyniak

Institute of Control and Computation Engineering,
University of Zielona Góra, ul. Podgórna 50, 65-246 Zielona Góra, Poland
A.Obuchowicz@issi.uz.zgora.pl, D.Wawrzyniak@weit.uz.zgora.pl
http://www.issi.uz.zgora.pl

Abstract. In recent years the problem of adaptation in time-varying landscapes has been intensively studied by many groups of researches. The number of publications successively grows. This domain of research is important for many technical branches, e.g. the optimal control, the learning process of neural networks, the fault detection in dynamic systems. In this work some taxonomy of non-stationary environments as well as behavior of a simple evolutionary process in such landscapes are presented and illustrated.

1 Introduction

Most optimization algorithms assume static objective function: they search for a near-optimum solution with respect to some fixed measure (or set of measures), whether it is maximization of profits, minimization of a completion time for some tasks, minimization of production costs, etc. However, real-world applications operate in dynamic environments, where it is often necessary to modify the current solution due to various changes in the environment. Thus it is important to investigate properties of adaptive algorithms which do not require re-start every time a change is recorded.

We are interested in solving such non-stationary problems with evolutionary computation techniques. It would be interesting to investigate the behavior of a evolutionary process in these scenarios. In this paper we discuss the nature of simple evolutionary model in the non-stationary adaptation problems.

The paper is organized as follows. In section 2 some taxonomy of non-stationary environments and a measure of the severity of changes are introduced. Quality rates for adaptation algorithms for different classes of non-stationary problems are described in section 3. Section 4 presents the simulating investigation of properties of the simple evolutionary process in a non-stationary environment. Finally the paper is summarized.

2 Non-stationary Environments

A non-stationary optimization problem in general can be formulated as follows:

$$\max f(\boldsymbol{x}, t) \big| \big(c_i(\boldsymbol{x}, t) \leq 0, i = 1, \ldots, m, \boldsymbol{x} \in \mathcal{D}(t) \big), \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 439–446, 2006.

where $f(\boldsymbol{x},t)$ is an objective function, $c_i(\boldsymbol{x},t)$ denotes an i^{th} constraint and $\mathcal{D}(t)$ is a space of solutions.

In general $f(\boldsymbol{x},t)$, $c_i(\boldsymbol{x},t)$ and $\mathcal{D}(t)$ can be time varying simultaneously. But physically it occurs very seldom. The classification of all possible cases, which elements of the sequence $(f,\{c_i\}_{i=1}^m,\mathcal{D})$ are varying in time, is proposed in [18]. In this work we focus on non-stationary landscapes with the empty set of constrains and the space of solution invariable in time.

There are a number of criteria along which non-stationary environments can be categorized [2]: frequency of changes, predictability of changes, regularity of changes, severity of changes. In this paper we focus on the last criterion.

The nature of changes can manifest in their speed and range. The classification under this criterion is difficult because of estimation subjectivity whether changes are sudden or adiabatic, wide or local. In the case of the fitness function varying in time, some measure of changes in a given subspace $\Omega \subset \mathcal{D}$ and a given time interval T can be introduced [18]

- continuous domain
  (let $f(\boldsymbol{x},t) \in L_2(\Omega), \forall t$)

$$M(\Omega,T,t) = \frac{1}{T}\sum_{\tau=t+1}^{t+T} \frac{\int\ldots\int_{\Omega}\big(f(\boldsymbol{x},\tau)-f(\boldsymbol{x},\tau-1)\big)^2 d\omega}{\int\ldots\int_{\Omega} f^2(\boldsymbol{x},\tau)d\omega}, \tag{2}$$

  where $d\omega = dx_1 dx_2 \ldots dx_n$;
- discrete domain

$$M(\Omega,T,t) = \frac{1}{T}\sum_{\tau=t+1}^{t+T} \frac{\sum_{\boldsymbol{x}_i\in\Omega}\big(f(\boldsymbol{x},\tau)-f(\boldsymbol{x},\tau-1)\big)^2}{\sum_{\boldsymbol{x}_i\in\Omega} f^2(\boldsymbol{x}_i,\tau)}. \tag{3}$$

The measure $M(\Omega,T,t)$ describes the average speed of relative fitness changes in subspace Ω taken over the time interval T which can be considered as a sampling interval, i.e. the time interval between two successive calculations of the fitness function. One may define two constants Θ_a and Θ_c ($\Theta_a < \Theta_c$) for given searching problem in order to classify changes of the fitness function:

- $M(\Omega,T,t) < \Theta_a$ — the adiabatic changes ($\mathcal{S}_1$), which guarantee approximately stationary state of evolutionary search. The population "keeps up" with the changed optimum. There are usually no quality differences between this problem and stationary problems.
- $\Theta_a < M(\Omega,T,t) < \Theta_c$ — the indirect changes ($\mathcal{S}_2$). It is the most interesting case. An effectiveness of the searching process significantly depends on a chosen searching strategy and its input parameters.
- $M(\Omega,T,t) > \Theta_c$ — the turbulent changes ($\mathcal{S}_3$). In this case, usually the search procedure have to be restarted and tuned to the completely new problem after the change has occurred.

Parameters Θ_a and Θ_c have, rather, informal nature and are not well defined. The ability of classifying, to which of changes type: $\mathcal{S}_1, \mathcal{S}_2$ or $\mathcal{S}_3$, a given problem

belongs, allows to choose a class of optimization methods to solve the problem and choose a measure of a given methods effectiveness. If a given problem belongs to the class $\mathcal{S}_1$, global optimum is usually moved to such a point which is close enough to be found again without a risk of becoming trapped in a local optimum. Then it is possible to use standard optimization methods, like gradient methods, to follow the optimum point during all the processing time. Here, evolutionary computation method is computationally rather too expensive to use.

From an evolutionary point of view, $\mathcal{S}_2$ is the most interesting class. The changes are too difficult and therefore computationally too expensive for the classical optimization methods, but not too difficult for evolutionary methods, which may solve the problem because of their softness and concurrent searching, especially when we are satisfied even if it's suboptimal only.

The turbulent changes $\mathcal{S}_3$ are usually unable to controlled (e.g. the changes of square error function in the on-line neural network training process where a sequence of training patterns is randomly chosen from a training set). Any optimization process can not keep up the optimum peak track. Applied adaptation algorithms usually find hills of a form of objective function averaged over searching time

3 Quality Rates for Adaptation Algorithms

Before a form of a quality rate for searching algorithm in the non-stationary environments is chosen, a researcher has to decide what kind of results will be satisfying, what type of searching process should be applied. Four main types of searching processes can be distinguished [13, 14].

$\mathcal{C}_1$: **A tracing process.** — This type of the searching process is dedicated mainly to adiabatic problems ($\mathcal{S}_1$). The goal of the searching process of the type $\mathcal{C}_1$ is to keep solutions closed to the optimum as well as possible. Most of publications of the non-stationary optimization consider such a type of searching process. Applied measures of searching algorithms are usually based on measures for stationary environments [5, 9, 10]. Although authors did not use these measures to non-stationary optimization evaluation, the closeness to the optimum during the search process is an interesting value which seems to be helpful in comparisons between applications and is easy to control in experiments. The evolutionary approach to non-stationary optimization presented in [12] uses the following tracing measure:

$$\mu_{tr} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \rho\big(\boldsymbol{x}_{opt}(t), \boldsymbol{x}_0(t)\big), \tag{4}$$

where $\boldsymbol{x}_0(t)$ is the best point of population in the time t and $\boldsymbol{x}_{opt}(t) = \arg\max_{\boldsymbol{x}\in\mathcal{D}} \Phi(\boldsymbol{x}, t)$, $\rho(\boldsymbol{a}, \boldsymbol{b})$ is a distance measure in $\mathcal{D}$, e.g. if $\mathcal{D} \subset \mathbb{R}^n$ then $\rho(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|$.

$\mathcal{C}_2$: **An optimization in a mega-epoch.** — This type of searching process concerns tasks, in which consecutive changes in the environment are significant but occur rather seldom (ones in a 'mega-epoch'). The goal is to find

the optimum before a new change occurs. For estimations of non-stationary optimization results, the two measures of *accuracy* and *adaptability* were proposed in [15, 17].

$\mathcal{C}_3$: **Keeping solutions on an acceptable level.** — In many real technological problems, e.g. in the *on-line* training of a dynamic neural networks [8, 13], in control systems [3] or in many problems of the operational research, the optimal solution is not so necessary as the solution of an acceptable quality. This problems usually are of the type $\mathcal{S}_2$. One has to be sure that the fitness of the actual best known solution will not be worse than a given assumed level during the whole length of a searching process. This acceptable level may, for example, describe non conflict run of concurrent processes, guarantee the stability of the controlled dynamic system [18].

$\mathcal{C}_4$: **A process with averaged acceptability.** — This type of searching process is dedicated to turbulent problems ($\mathcal{S}_3$). A searching process is unable to follow the optimum as well as to guarantee the acceptable solutions during the algorithm processing. The only measure of the adaptation process is its ability to find the solution with the best average fitness over all realization of $\Phi(\boldsymbol{x}, t)(t = 1, 2, \ldots, t_{\max})$. This measure can be expressed in the following form [13]

$$\mu_{avac} = \frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \rho\big(\boldsymbol{x}^\star, \boldsymbol{x}_0(t)\big), \tag{5}$$

where

$$\boldsymbol{x}^\star = \arg\max \left(\frac{1}{t_{max}} \sum_{t=1}^{t_{max}} \Phi(\boldsymbol{x}, t) \right).$$

4 Illustrative Examples

Let us consider the following two-dimensional non-stationary adaptation landscape composed by two Gaussian peaks

$$\begin{aligned} \Phi(\boldsymbol{x}, t) = (0.5 + 0.5\cos\big(\pi t/s\big))e^{-10\|\boldsymbol{x}-\boldsymbol{a_1}\|^2} \\ +(0.5 - 0.5\cos\big(\pi t/s\big))e^{-10\|\boldsymbol{x}-\boldsymbol{a_2}\|^2}, \end{aligned} \tag{6}$$

where t denotes time, s is a given positive parameter controlling the rate of change of peaks highs, $\boldsymbol{a}_1 = (0.25, 0.25)$ and $\boldsymbol{a}_2 = (0.75, 0.75)$ are locations of Gaussian peaks' centres.

Adaptation process is controlled by the Evolutionary Search with Soft Selection (ESSS) algorithm [6, 13]. A real, n-dimensional, searching space (an adaptation landscape) $\mathbb{R}^n$ is given. A fitness function Φ to be maximized is also defined on this adaptation landscape. Previously, an initial population $P(0)$ of η elements is randomly generated. If the ESSS algorithm is used to solve the optimization problem in $\mathbb{R}^n$ without constrains, the concept that an initial population has to be 'uniformly distributed' in the search space has no sense. One

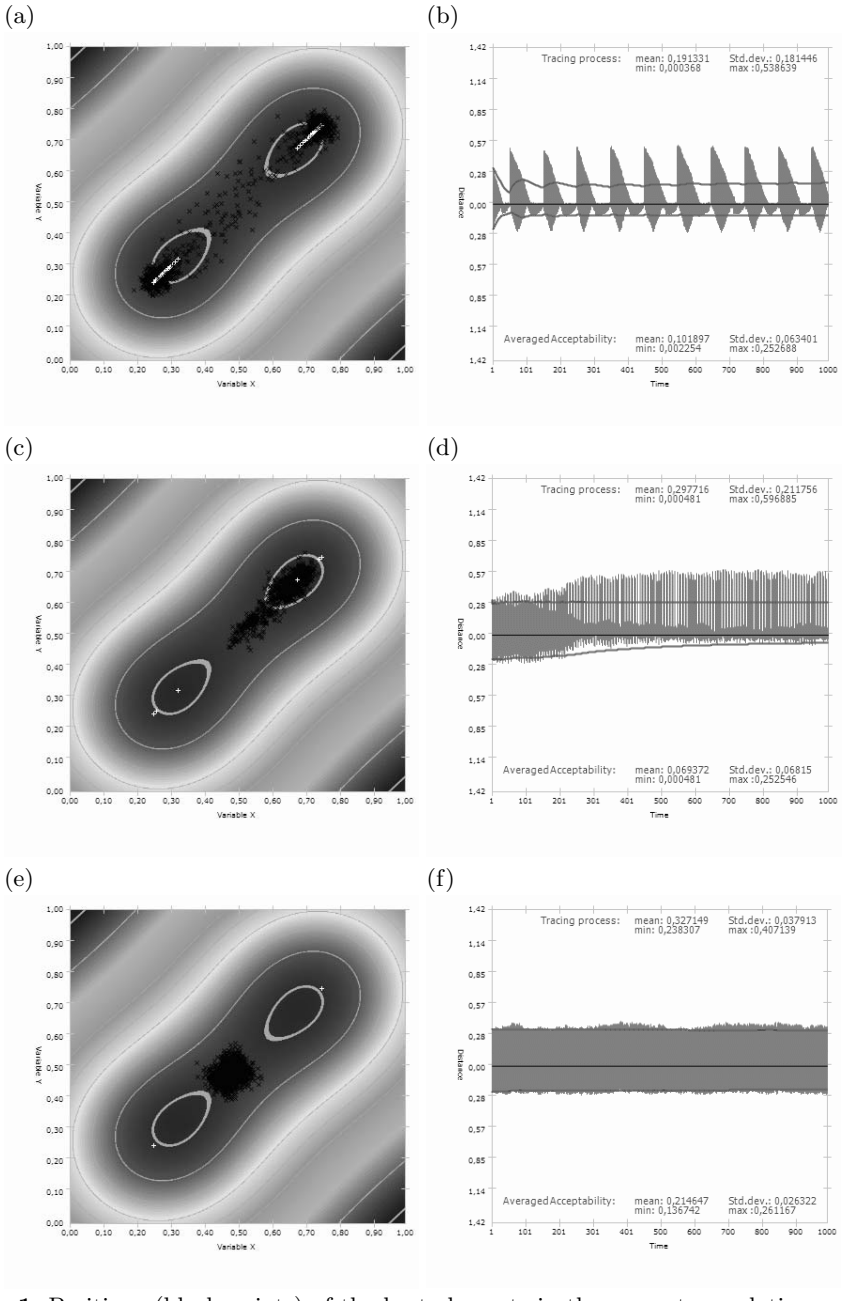


Fig. 1. Positions (black points) of the best elements in the current population on the map of mean values of the fitness funtion taken over one period of changes for adiabatic (a), indirect (c) and turbulent (e) cases. The comparison of two measures: the tracing μ_{tr} (4) (the upper graph) and the average acceptability μ_{avac} (5) (the lower inverse graph) ones for adiabatic (b), indirect (d) and turbulent (f) cases.

Table 1. The comparison of two measures of the ESSS adaptation quality for landscapes belonging to different severity classes

function severity class		evolutionary process	
s	$M([[0,1],[0,1]],1000,1000)$	μ_{tr}	μ_{avac}
100	0.00021 (adiabatic)	0.19133	0.10190
10	0.02071 (indirect)	0.29772	0.06937
1	1.27840 (turbulent)	0.32715	0.03124

of the possible and rational solution is to create an initial population by adding η times a normally-distributed random vector to a given initial point $\boldsymbol{x}_0^0 \in \mathbb{R}^n$. The fitness $\phi_k^0 = \Phi(\boldsymbol{x}_k^0)$ is calculated for each element $\boldsymbol{x}_k^0$ of the population $(k = 1, 2, \ldots, \eta)$. The searching process consists in generating a sequence of η-element populations. A new population $P(t+1)$ is created based only on the previous population $P(t)$. In order to generate a new element $\boldsymbol{x}_k^{t+1}$, a *parent* element is selected and mutated. Both selection and mutation are random processes. Each element $\boldsymbol{x}_k^t$ can be chosen as a parent with a probability proportional to its fitness ϕ_k^t (the *roulette method*). A new element $\boldsymbol{x}_k^{t+1}$ is obtained by adding a normally-distributed random value to each entry of the selected parent:

$$\left(\boldsymbol{x}_k^{t+1}\right)_i = \left(\boldsymbol{x}_{h_k}^t\right)_i + N(0,\sigma) \quad i = 1, \ldots, n, \tag{7}$$

where the standard deviation σ is a parameter to be selected.

Figure 1 presents results for the ESSS algorithm with following parameters: $\eta = 20$ and $\sigma = 0.01$. Figure 1(a) and 1(b) illustrates the adiabatic function changes $s = 100$. The best point of the population follows the global optimum during all time. In the case of the indirect changes (Fig.1(c) and 1(d)), population converges to one of the local optimums (which is higher in the begining of the evolutionary process) and fluctuates around it. The turbulent case (Fig.1(e) and 1(f)) is characterized by the fact that the global optimum is not monitored. The population fluctuates around the indirect point between local peaks. This feature is well known in the *on-line* training process of the artificial neural networks (cf. [8]).

The comparison of two measures: the tracing measure μ_{tr} (4) and the average acceptability μ_{avac} (5), of the ESSS adaptation process quality is presented in Tab. 1. All values of measures are averaged over 500 algorithms runs. It is easy to see that the effectiveness of the evolutionary adaptation has to be described using different measures for landscapes for landscapes of different severity class. Obtained results confirm the observation of behavior of the considered evolutionary process presented in Fig. 1.

5 Summary

For years, the evolutionary algorithms were applied mostly to the group of static problems. However, nowadays a wider group of non-stationary problems is optimized with evolutionary algorithms. In this paper, the properties of the simple

evolutionary process in a non-stationary environment are analyzed using a set of simulating experiments. Obtained results shows that behavior of the population strongly depends on the severity of landscape changes. In the adiabatic case the population follows the current global optimum localization. In the turbulent case the population tries to localize the averaged solution taken over a given (relatively wide) time period. Both evolutionary adaptation properties can be quantitatively monitored using two measures: the tracing and the average acceptability ones. The first measure μ_{tr} (4) gives an information how closed to the optimum the current solution is. The second one μ_{avac} (5) represents the evolutionary adaptation process ability to find the solution with the best average fitness taken over all realization of fitness function during a given wide time period.

References

1. Bäck, T., On the Behaviour of Evolutionary Algorithms in Dynamic Environments, ICEC'98, IEEE Publishing, 1998, pp 446-451.
2. Branke, J., Memory Enhanced Evolutionary Algorithm for Changing Optimisation Problems, CEC'99, IEEE Publishing, 1999, pp 1875-1882.
3. Bryson, A.E., Ho, C., Applied Optimal Control, New York: A halsted Press Book, 1975.
4. De Jong, K., A., An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, (Doctoral Dissertation, University of Michigan), Dissertation Abstract Int. 36(10), 5140B. (University Microfilms No 76-9381), 1975.
5. Feng, W., Brune, T., Chan, L., Chowdhury, M., Kuek, C., K., Li, Y., Benchmarks for Testing Evolutionary Algorithms, The Third Asia-Pacific Conf. on Measurement & Control, Dunhuang, China, 1998.
6. Galar, R., Evolutionary search with soft selection,Biological Cybernetics, Vol.60, 1989, pp.357–364.
7. Grefenstette, J., J., Genetic Algorithms for Changing Environments, 2PPSN, Elsevier Science Publishers B. V., 1992, pp 137-144.
8. Korbicz, J., Obuchowicz, A., Patan, K., Network of Dynamic Neurons in Fault Detection Systems, IEEE Int. Conf. System, Man, Cybernetics, IEEE Publishing, 1998, pp. 1862–1867.
9. Mori, N., Imanishi, S., Kita, H., Nishikawa, Y., Adaptation to a Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm, VII ICGA'97, Morgan Kauffman, 1997, pp. 299–306.
10. Mori, N., Kita, H., Nishikawa, Y., Adaptation to a Changing Environments by Means of the Thermodynamical Genetic Algorithm, 4PPSN, vol. 1141 in LNCS, Springer, 1996, pp 513-522.
11. Mothes, J., Incertitudes et décisions industrielles, Dunod, 1967.
12. Obuchowicz, A., Adaptation in time-varying landscape using an evolutionary search with soft election, 3KAEiOG 1999, Warsaw University of Technology Press, pp.245–251.
13. Obuchowicz, A., Evolutionary Algorithms for Global Optimization and Dynamic System Diagnosis, Lubuskie Scientific Society, Zielona Góra, 2003.
14. Obuchowicz, A., Trojanowski, K., Some Aspects of Evolutionary Adaptation in Non-Stationary Environments, AEiOG Workshop 2002, Warsaw University of Technology Press, pp. 69–78.

15. Trojanowski, K., and Michalewicz, Z., Extensions of Evolutionary Algorithms for Non-Stationary Environments, 3KAEiOG 1999, Warsaw University of Technology Press, pp 317-328.
16. Trojanowski, K., and Michalewicz, Z., Evolutionary Algorithms for Non-Stationary Environments, 8IIS, 1999, ICS PAS Press, pp. 229–240.
17. Trojanowski,K., and Michalewicz, Z., Searching for Optima in Non-Stationary Environments, CEC'99, IEEE Publishing, pp.1843-1850.
18. Trojanowski,K., and Obuchowicz, A., Measures for Non–Stationary Optimization Tasks, ICANNGA'2001, Springer–Verlag, 2001, pp. 244–248.
19. Vavak, F., Fogarty, T., C., Comparison of Steady State and Generational Genetic Algorithm for Use in Nonstationary Environments, ICEC'96, IEEE Publishing, Inc., 1996, pp 192-195.
20. Vavak F., Fogarty T.C., Jukes K., Learning the Local Search Range for Genetic Optimisation in Nonstationary Environments, ICEC'97, IEEE Publishing, Inc., 1997, pp 355-360.

Hierarchical Representation and Operators in Evolutionary Design

Barbara Strug

Institute of Computer Science, Jagiellonian University,
Nawojki 11, Cracow, Poland
strug@softlab.ii.uj.edu.pl

Abstract. This paper deals with the possibilities of applying evolutionary methods in computer aided design[1]. As the representation is one of the main issues in any computer aided design system here an approach using hierarchically organized data is presented. It is based on traditional graph structures extended to be able to represent different types of relations within a designed object. Genetic operators working on such graphs and other elements of an evolutionary system are also presented. The method is illustrated by examples from two different domains. One of them is a chair design system based on proposed method. The other one is derived from the design of skeletal structures of transmission towers.

1 Introduction

The human design process, computer aided or traditional, is an iterative one consisting of several steps [1]. Firstly a preliminary or conceptual design is created, then it is analysed or tested in order to find out which of its components must be redesigned or refined. The process of evaluation and optimisation is repeated until an acceptable solution is found. Still, majority of computer aided design systems focuses on refining parameters defining design, thus on its optimisation and they usually work on a single design at a time. Since designing can be treated in computer science as a search process, with a space of all possible designs being a search space, it is possible to use search techniques used in other domains.

There is a number of search methods well established in computer science that can also be used in the space of designs. [14]. One of them is evolutionary technique. Instead of one solution at a time a larger subset of the search space, known as a population, is considered. As evolutionary search consists in evaluating and refining possible solutions it can be seen as analogous to a human design iterative process of analysis, testing and optimisation [1, 3]. Similarly to the refinement step in human design, which is based on earlier analysis and testing, in evolutionary search designs to be transformed are determined according to their evaluation (so called fitness). The refinement step is often performed

[1] The work reported in this paper was partially supported by the State Committee for Scientific Research grant 8T07A01621.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 447–454, 2006.

not on actual solutions (called phenotypes, in this paper - designs) but on their coded equivalents (called genotypes). Traditionally a binary coding is used most often in which each solution is transformed to a binary string [1, 10, 4, 13].

In design problems it is very often insufficient as not only geometrical properties of an object has to be represented but also other ones (like colour, material etc.) as well as relations between object components.

The methods used in CAD problems like boundary representations, sweep-volume representation, surface representations or CSG (constructive solid geometry) [11, 9, 12] allow only for the "coding" of geometry of an object being designed and do not take into account the inter-related structure of many design objects i.e. the fact that parts (or components) of an object can be related to other parts in different ways. In this paper a representations based on hierarchically organized data is used.

Different types of graphs have been researched and used in this domain, for example composition graphs [5, 6, 7]. In this paper an extension of composition graphs - hierarchical graphs is presented. They can represent an artifact being designed at different levels of detail at different stages of the design process thus hiding unnecessary low-level data and presenting the designer only an outline of the object or showing a detailed view of the whole object (or of its part).

Using hierarchical graphs as the representation in an evolutionary search requires the adaptation of traditional evolutionary operators like cross-over and mutation. As the graphs selected by some selection methods to be transformed by the evolutionary operators at subsequent stage of the evolution and their structure are not known a priori the operator must be defined in such a way as to allow for an "online" computation of new graphs. Thus the operator has to be specified by an algorithm rather then a set of rules.

An example of application of this method is also shown and some advantages and disadvantages of this approach as well as possible future research directions are also briefly discussed.The method is illustrated by examples from two different domains. One of them is a chair design system based on proposed method. The other one is derived from the design of skeletal structures of transmission towers [2, 8, 15].

2 Representation

Hierarchical graphs (HGs) are an extension of traditional graphs. They consist of nodes and edges. What makes them different from standard graphs is that nodes in HGs can contain internal nodes. These nodes, called children, can in turn contain other internal nodes and can be connected to any other nodes with only exception being their ancestors.

A node in a hierarchical graph may represent a geometrical object or a groups of objects. It may also be used to hide certain details of a designed object that are not needed at a given stage of design or to group object having some common features (geometrical or functional). A node that represents a single object is called an object node. Nodes that do not represent actual geometric entities but

are used to represent hierarchical structure or other relations are called group nodes. It is important to note that the edges between nodes are by no means limited to edges between descendants of the same node. In other words there may exist edges between nodes having different ancestors.

Nodes and edges in hierarchical graphs can be labelled and attributed. Labels are assigned to nodes and edges by means of node and edge labelling functions respectively, and attributes - by node and edge attributing functions. Attributes represent properties (for example size, position, colour or material) of a component represented by a given node.

A labelled attributed hierarchical graph may represent a potentially infinite number of designs. The given graph G being a hierarchical graph can represent a, potentially, infinite subset of designs having the same structure. To represent an actual design we must define an instance of a graph. An instance of a hierarchical graph is a hierarchical labelled attributed graph in which to each attribute a value has been assigned from the set of possible values of a given attribute.

As such a hierarchical graph defines only a structure of a design to create a visualisation of an object an interpretation is necessary. Such interpretation determines the assignments of geometrical objects to nodes, its fragments to bonds and establishes a correspondence between edges and sets of relations between objects (components of a design). The objects assigned to nodes are usually called primitives. The geometry of these objects may be internally represented by means of any known representation that allows for easy application of similarity transformations. Geometrical objects used depend on the domain of application, for example when designing chairs the set of geometrical objects could contain boxes and cylinders, or some predefined objects like legs, seats and other parts of a chair and a set of relations could consist of an adjacency relation. For the design of transmission towers a set of primitives may consist of bars used to build trusses [15].

An example of a hierarchical graph and a corresponding object are depicted in fig.1a and 1b respectively. The nodes depicted as black filled circles are object nodes and other nodes are group nodes. The object shown (a simple chair) is actually only one of many possible designs this graph can represent. The chair was obtained after choosing an instance of this graph and then interpreting it.

3 Evolutionary Design System

To use such a representation in an evolutionary design system a number of elements of this system must be defined.

Firstly the method of initialization must be chosen. One of the possibilities is to generate a population of random graphs consisting of nodes and edges from a given set. Although this method is easiest to implement in any design system it is usually very slow in producing acceptable or feasible designs as many designs are rejected. Another possible mechanism is known as graph grammars and it has been successfully used in many domains to generate graphs [16]. Graph grammars are systems of graph transformations consisting of symbols, an initial graph (an

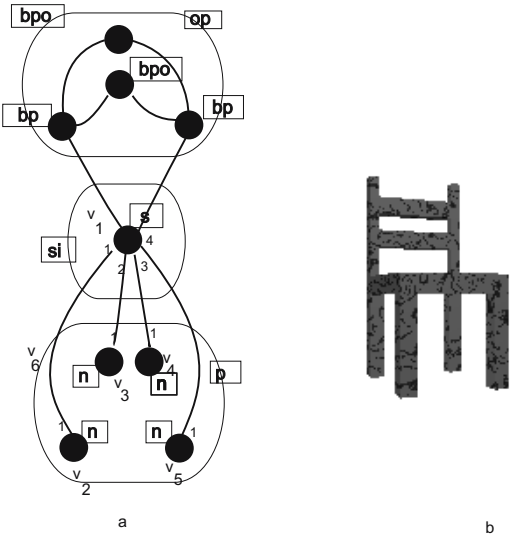


Fig. 1. A HG-graph and one of its possible interpretations

axiom) and sets of rules called productions. Each production is composed of two graphs named left side and right side of the production.

A family of HG-graphs can be generated by means of a graph grammar. Such a grammar describes all formally correct solutions, for example chairs or layouts of transmission towers. To produce a graph we apply a sequence of productions. We start with an initial graph as a current one. Applying an appropriate production consists in finding a subgraph of the current graph such that it is isomorphic with the left side of the production to be applied. Then this subgraph is replaced by the right side of the production. This two steps yield a new current graph and they are repeated until we reach a final graph. This process is called a derivation.

It also possible to allow the user to generate an initial population of graphs or to use graphs generated by another program.

The genetic operators (usually a crossover and a mutation) constitute the next element of an evolutionary algorithm. As in this paper a nonstandard representation is used new genetic operator had to be devised.

The graph based equivalent of a standard crossover operator requires establishing subgraphs that would be then exchanged. When a crossover is performed on two selected graphs, G_1 and G_2 the subgraphs g_1 and g_2 respectively are selected in these graphs. Then each subgraph is removed from a graph and inserted into the second one. As a result two new graphs are generated. However there may exist edges connecting nodes belonging to a chosen subgraph with nodes not belonging to it. Such edges are called embedding of a subgraph. So removing a subgraph from a graph and inserting it into another requires a method allowing for proper re-connection of these edges. The underlying idea is that all edges should be re-connected to similar nodes to those they were connected to

in the graph from which they were removed. There is probably more than one possibility of defining which nodes are similar.

In this paper a similarity-like relation is used. This relation is called homology. The name was inspired by the gene homology in biology. This relation is responsible for establishing subgraphs of selected graphs that are homologous - or similar in some way- and thus can be exchanged in the crossover process. The definition of this relation is based upon the assumption that both graphs selected for crossover code designs consisting of parts having similar or even identical functions (even if these parts have different internal structure, material or/and geometrical properties).

In other words both graphs are assumed to belong to the same class. The homology relation is defined on three levels that differ in terms of requirements put on graphs to be in the relation. The weakest of these relations is called context free homology and it only requires two subgraphs to have the same number of top-level nodes (i.e. nodes that do not have ancestors) with identical labels (without considering the number and labels of their children-nodes or ancestors). It is the least restrictive of the three relations and it allows for higher variety of new graphs to arise from a crossover but at the same time it is able to produce the least meaningful graphs or, in other words, the most "disturbed" ones.

On the opposite side the strongly context dependent homology is defined. It requires the top-level nodes in both subgraphs to have not only identical labels but also to have identically labeled ancestors up to the top-most level of the graph hierarchy. Nevertheless the internal structure of a node and its attributes are not taken into account so even exchanging strongly homologous subgraph may still produce considerably different new graphs. When the context free relation is to weak i.e. it results in too many graphs being unacceptable (rejected by fitness function) and the strong homology is too restrictive or results in designs that are very similar or even identical to its parents the weakly context dependent homology may be useful. It takes into consideration direct ancestors of a given node but not any ancestors of higher level in graph hierarchy.

Formally a crossover operator cx is defined as a 6-tuple $(G_1, G_2, g_1, g_2, T, U)$, where G_1, G_2, g_1, g_2 are hierarchical graphs and their subgraphs respectively. The crucial elements of this operator are T and U that are called embedding transformations i.e. they describe how edges of the embedding are to be re-connected. They are sets of pairs of the form (b, b'), where b denotes a bond (a predefined part of a node) to which an edge was connected originally and b' - the one to which it will be re-connected in a new graph.

It is important to notice however that the graphs to be crossed over and their respective subgraph are selected during the execution of the evolutionary algorithms so the embedding transformations can not be defined a priori (as it is in graph grammars [16, 5]. Hence probably the most difficult problem is to find a method allowing us to establish these transformations. The algorithm generating these transformations requires only the subgraphs being exchanged

to be homologous. For each level of homology a crossover operator is defined thus we have three crossover operators having different level of context dependence.

As the second genetic operator mutation is usually used. This operator is much easier to be defined for hierarchical graph-based representation.

The mutation operators may be divided into structure changing mutations and attributes changing. The second group can be further divided into local and global mutation operators.

The attribute changing operators are executed as changing values of attributes of a selected node (local mutation) or nodes (global mutation). As a result it changes geometrical properties of objects assigned to this node or nodes by the interpretation. However it is also possible to define mutation operators introducing structural changes in an artifact being designed what would not be possible in binary representation. Such mutations could consist in adding or removing nodes from a hierarchical graph. In the chairs design system these mutations may for example result in obtaining chairs with changed number of legs or different number of components of a back.

So while crossover allows us to generate artifacts being combinations of previously existing designs mutation may introduce wholly new elements into object being designed.

4 Results

The method presented in this paper was implemented. It was tested in the domain of graphical design - as a chair design system It was also used as a tool for the design of transmission towers. To select genotypes to be crossed over or mutated a fitness proportional selection is used in both applications. The examples of objects obtained from both systems are presented in fig. 2 and fig. 3 respectively.

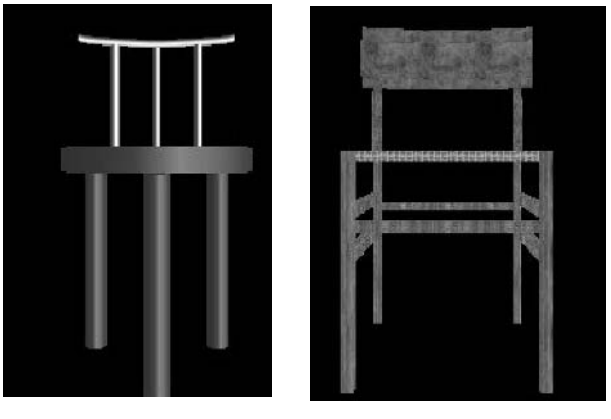


Fig. 2. Examples of chairs generated by an evolutionary system

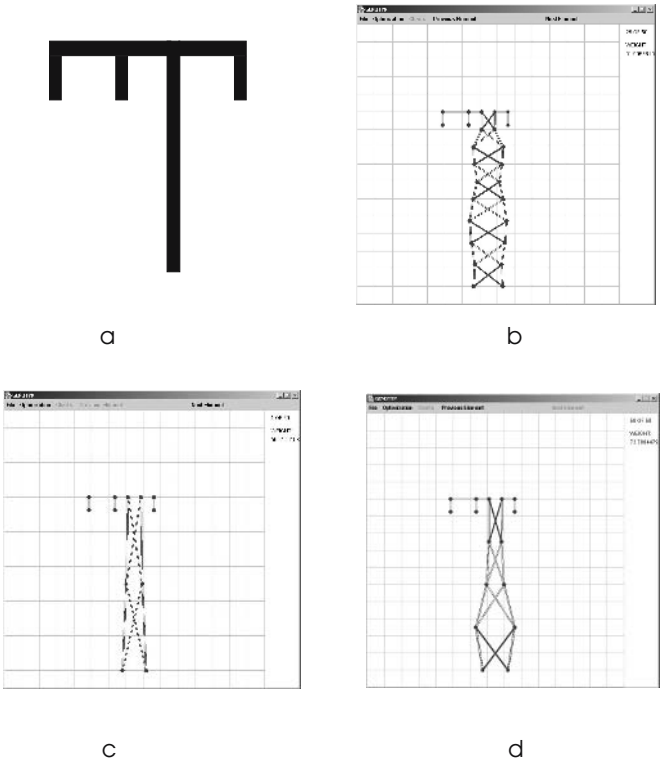


Fig. 3. Examples of first and second level towers design by an evolutionary system

The chairs presented in fig.2a and 2b were obtained by a graph-based evolutionary algorithm from an initial population of 10 graphs representing chairs defined by the user.

The evolutionary design system for transmission towers is a two level system [2]. In the first step a layout of the tower is evolved from the initial population generated by a graph grammar. Then one layout is selected and its internal structure is evolved. In fig 3a a layout selected from the first level is depicted. F ig. 3b depicts an element from the initial population of the second level evolution based on the selected layout. Figs. 3c and d show the best element evolved in the second level evolution and a poor one respectively.

5 Conclusions

Applying evolutionary methods to the design domain poses many problems. One of the main problems concerns coding designs in such a way that they can be easily transformed during an evolutionary process. The solution presented in this paper is a hierarchical graph used as a genotype and equivalents of standard genetic operators. Graph-based operators are usually more complex than

those used in binary representation but in author's opinion the benefits of using graph representation (possibility of coding relationships between components of an artifact and ability to introduce structural changes) compensate for it. The strongest point in a graph-based representation is its ability to represent in a uniform way all types of relations and objects and to produce highly fit individuals.

The use of graph grammars makes it possible to generate an initial population of graphs represented designs belonging to a desired class. Thus the graph grammar and fitness function are the only elements of the evolutionary design system that has to be changed in order to design different objects.

References

1. P. J. Bentley, Generic Evolutionary Design of Solid Objects using a Genetic Algorithm, PhD thesis, UCL London 1), 3-38. (1997)
2. Borkowski A., Grabska E., Nikodem P, and Strug B., Searching for Innovative Structural Layouts by Means of Graph Grammars and Evolutionary Optimization,, Proc. 2nd Int. Structural Eng. and Constr. Conf, Rome (2003)
3. De Jong K, Arciszewski T, and Vyas H, An Overview of Evolutionary Computation and its Applications, in. Artificial Intelligence in Engineerig,9-22, Warsaw (1999)
4. D.E.Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Reading, MA, Addison-Wesley. (1989)
5. E.Grabska,Theoretical Concepts of Graphical Modelling. Part one: Realization of CP-graphs. Machine GRAPHICS and VISION, 2(1993)
6. Grabska, E.. Graphs and designing. Lecture Notes in Computer Science, 776 (1994).
7. E.Grabska, W. Palacz, Hierarchical graphs in creative design. MG&V, 9(1/2), 115-123. (2000)
8. Hajela P. and Lee, J, genetic Algorith in Truss Topological OpJournal of Solids and Structures vol.32, no 22 , 3341-3357, (1995)
9. Hoffman, C. M.,Geometric and Solid Modeling: An Introduction, Morgan Kaufmann, San Francisco, CA, (1989)
10. Holland, J. H. Adaptation in Natural and Artificial Systems, Ann Arbor,(1975)
11. Mantyla, M.,An Introduction To Solid Modeling, Computer Science Press, Rockville,MD,vol.87,(1988)
12. Martin, R R and Stephenson, P C Sweeping of Three-dimensional Objects Computer Aided Design Vol 22(4) (1990), pp223-234.
13. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs.. Springer-Verlag, Berlin Heidelberg New York (1996)
14. Michalewicz, Z. Fogel, D. B.: How to Solve It: Modern Heuristics.. Springer-Verlag, Berlin Heidelberg New York (2000)
15. P. Nikodem and B. Strug. Graph Transformations in Evolutionary Design, Lecture Notes in Computer Science,vol 3070, pp. 456-461, Springer, 2004.
16. Rozenberg, G. Handbook of Graph Grammars and Computing by Graph. Transformations, vol.1 Fundations, World Scientific London (1997)

Improving Parallelism in Structural Data Mining

Min Cai[1], Istvan Jonyer[1], and Marcin Paprzycki[2]

[1] Department of Computer Science, Oklahoma State University,
Stillwater, Oklahoma 74078, U.S.A
{cmin, jonyer}@cs.okstate.edu
[2] Computer Science Institute, SWPS, 03-815 Warsaw, Poland
marcin.paprzycki@swps.edu.pl

Abstract. Large amount of data collected daily requires efficient algorithms for its processing. The SUBDUE data mining system discovers substructures in structurally complex data, based on the minimum description length principle. Its parallel implementation, MPI-SUBDUE, was created in 2001 to reduce computation time and/or to deal with larger datasets. In this paper, a new, more efficient implementation of MPI-SUBDUE is introduced. The experimental results show that, for the mutagenesis dataset, the new implementation outperforms the original one by up to 33% and that the performance gain increases with the number of processors used.

1 Introduction

Large amounts of data are collected on a daily basis and added to the existing repositories. The need to extract valuable information from this data challenges researchers to develop efficient techniques to discover and interpret interesting patterns in it. In this paper we are interested in discovering concepts in structural data, for which a number of algorithms have been proposed [1, 6, 8]. One of them, SUBDUE, discovers substructures on the basis of the minimum description length principle [8]. Working with graph-based data representation and utilizing graph-algorithms, when applied to real-world problems, SUBDUE takes a very long time to execute. In 2001, a parallel version of SUBDUE (MPI-SUBDUE) was implemented [1, 8]. MPI-SUBDUE partitions the graph representing the dataset into parts that are analyzed independently first, and then partial results are communicated so that the globally-best substructure is selected. Parallelization was achieved through MPI, while communication between processes consisted of a sequence of point-to-point messages. However, even a superficial analysis of MPI-SUBDUE indicated a number of possible inefficiencies, which not only could have resulted in poor performance, but also constitute a problem form the point of view of programming simplicity and expressiveness [7].

Since SUBDUE is one of the best existing graph-based data mining tools, we have decided to develop its more efficient version and in the process to improve it from the point of view of programming simplicity and expressiveness. To accomplish these goals, we have proceeded with the following modifications of the MPI-SUBDUE:

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 455–462, 2006.

- the initial graph partitioning was parallelized,
- multiple point-to-point communications (MPI_Send and MPI_Recv) have been replaced by an MPI collective communication (MPI_Allgatherv),
- parallel summation has been applied to deciding the globally best discoveries.

In reporting our work we proceed as follows. In the next section we present an overview of the SUBDUE system. Both the original and the NEW-MPI-SUBDUE are described in Section 3. We follow with the description and analysis of experimental results of testing both algorithms on three datasets. In Section 5 we summarize our results and outline our future work.

2 SUBDUE

SUBDUE is a data mining system working through substructure discovery aimed at finding interesting and recurring subgraphs in a labeled graph. This goal is achieved by applying the minimum description length principle (MDL). SUBDUE has been successfully applied to molecular biology, image analysis and computer-aided design, as well as other domains [8]. During SUBDUE's execution two basic steps (1) substructure discovery and (2) replacement are performed. Structural data (input for SUBDUE) is represented as a labeled graph. Objects in the data correspond to vertices, while relationships between them correspond to edges. A substructure is a connected subgraph (a subset of vertices and edges from the input graph) and represents a structural property of the dataset. In the first step SUBDUE discovers the best (according to the MDL principle) substructure in the complete data-graph. This substructure could be the final answer, but it is also possible to repeat the process to discover a hierarchy of important structures. To achieve this goal, all instances of the best substructure at a given level are compressed to a single vertex and SUBDUE is invoked on the "reduced graph". Hierarchy of substructures obtained in this way is then used for various levels of interpretation; depending on goals of data analysis (see [1, 6, 8] for a complete description of SUBDUE and examples of its application to real-life problems).

3 Parallel SUBDUE

Data parallel approach and functional (or control) parallelization are the two main ways of achieving algorithm parallelization. Both approaches can be used to parallelize SUBDUE. Functional parallel SUBDUE has been introduced in [1]. In this paper, we focus on the data parallel approach. In parallel SUBDUE, the input graph is first partitioned into n partitions that are distributed among n processors. Each processor finds the best substructure in its partition and communicates it to all other processors. Each processor uses these substructures to compare them with its own structures, using the MDL principle (structure that is very good for the data in partition k may be bad in partition l and thus all "local champions" must be compared vis-*á*-vis structures in each partition).

Results of local comparison are then combined and exchanged between processors and as a result, the "globally best" substructure is found. This process can be repeated to obtain a hierarchical decomposition [1, 6, 8].

Let us note that this approach to algorithm parallelization may come at a price. Since the original graph is divided into subgraphs it is possible that some structures that existed in the original dataset will be lost. This happens when the input is partitioned by removing some edges, thus potentially cutting important substructures and dividing them between multiple processors. To maximally offset this problem a high-quality graph partition program (METIS) is used, which eliminates only a minimal number of edges; thus reducing possibility of splitting important substructures.

3.1 MPI-SUBDUE

In the original version of MPI-SUBDUE [8], the input graph was partitioned into n subgraphs using the METIS package [2]. In METIS 4.0, two partitioning programs *pmetis* and *kmetis* can be used to partition a graph. The *pmetis* is based on multilevel recursive bisection [5], whereas the *kmetis* is based on multilevel k-way partitioning [4]. Both routines produce high quality partitions. However, as specified in [2], *kmetis* is expected to be considerably faster than *pmetis* when the number of partitions is larger than $n = 8$.

A series of point-to-point communications (MPI_Send and MPI_Recv) were used to exchange information between processors. There is a total of three situations in MPI-SUBDUE, when inter-processor communication takes place: (1) when the best local substructures are exchanged, (2) when the results of local comparisons are propagated, and (3) when the final best substructure is established (see above and [8]).

3.2 NEW-MPI-SUBDUE

We made three improvements to the original version of MPI-SUBDUE. First, we have explored parallelism in the graph partitioning. The input graph is now partitioned using the PARMETIS (version 3.1) graph partitioning package [3]. Second, instead of using point-to-point communications (MPI_Send and MPI_Recv routines), the MPI collective communication (MPI_Allgatherv routine) is used. Obviously, point-to-point communication involves a pair of processors and use of MPI_Send and MPI_Recv puts excessive communication demands on the system, which deteriorates the program's performance. Collective communication, on the other hand, involves every process in a group, and allows all machine resources to be exploited more efficiently. Furthermore in many cases, vendors offer machine-tuned, efficient implementations of collective communication [7]. Third, hierarchical (binary-tree based) summation is used in finding globally best substructures. After being evaluated on all partitions, the "scores" of substructures are propagated up the hierarchy, where at each internal node of the tree these "scores" are added for substructure reported from two different partitions, and their sum is passed further up the hierarchy. The algorithm of the

```
NEW-MPI-SUBDUE(Graph)
Global variables: n, j, Value[0...(n-1)], Pos_Instances[0...(n-1)], Neg_Instances[0...(n-1)]
begin
  spawn(P0, P1, P2, P3, ..., Pn);
  apply PARMETIS to partition the graph into n partitions;
  for all Pi where 1 ≤ i ≤ n do
    each processor discovers the best substructure in its graph partition;
    each processor broadcasts its best substructure to all other processors;
    each processor evaluates its best substructure and broadcasts the results to all other processors;
    each processor stores the value, number of positive instances, number of negative instances of
      the best substructures in Value[0...(n-1)],Pos_Instances[0...(n-1)],
      and Neg_Instances[0...(n-1)], respectively;
    for j ← 0 to log(n) do
      if (i-1) mod 2^j = 0 and i-1+2^j < n then
        Value[i-1] ← Value[i-1] + Value[i-1 + 2^j];
        Pos_Instances[i-1] ← Pos_Instances[i-1] + Pos_Instances[i-1+2^j ];
        Neg_Instances[i-1] ← Neg_Instances[i-1] + Neg_Instances[i-1+2^j ];
        each processor broadcasts Value[i-1], Pos_Instances[i-1], Neg_Instances[i-1]
          to all other processors;
      endif
    endfor
    each processor updates the values of its best substructure with its corresponding elements
      in Value[0...(n-1)], Pos_Instances[0...(n-1)], and Neg_Instances[0...(n-1)] and
      sends the updated best substructures to P0;
    endfor
  P0 finds and outputs the global best substructures
end
```

Fig. 1. The NEW-MPI-SUBDUE algorithm

NEW-MPI-SUBDUE is summarized in Figure 1. Interestingly, as we will see in the next section, set of changes that could be considered relatively insignificant, resulted in a very significant performance improvement.

4 Experimental Results

Let us now illustrate the performance of the NEW-MPI-SUBDUE in experiments performed on three input graphs that represent mutagenesis data and are derived from OxUni [9]. These datasets were collected in order to predict the mutagenicity of aromatic and heteroaromatic nitro compounds. Here, mutagenicity is denoted by positive or negative real numbers. There are four levels of background knowledge in the database: atoms in the molecules, bonds in the molecules, and two attributes describing the molecule as a whole and higher level submolecular structures. The atom and bond structures were obtained from a standard molecular modeling package called Quanta [9]. In our current experiments, we were concerned only with performance improvements of the

Table 1. Partition time for Graphs 1 and 2

	Graph 1		Graph 2	
N	Execution time METIS (seconds)	Execution time PARMETIS (seconds)	Execution time METIS (seconds)	Execution time PARMETIS (seconds)
2	0.145	0.081	0.147	0.082
4	0.164	0.053	0.169	0.073
8	0.164	0.076	0.172	0.071
16	0.182	0.075	0.204	0.070

NEW-MPI-SUBDUE, and due to space limitations and since such results are outside of scope of this paper, we will not report on patterns found in the data.

First, we experimented with two small data graphs: Graph 1 had 2844 vertices and 2883 edges, while Graph 2 had 2896 vertices and 2934 edges. These two graphs are similar in size to these used in the original report [8] and we have found that the computer hardware has progressed so fast, that problems that were computationally challenging in 2000 are now much too small for an interesting comparison. We have therefore moved to the complete mutagenesis dataset (Graph 3 had 22268 vertices and 22823 edges), where the real performance gain could be observed.

Our experiments were performed on a cluster composed of 20 compute nodes, each with two AMD Athlon MP 1800+ (1.6GHz) CPUs. Each compute node has 2 GB of DDR SDRAM, and switching between compute nodes is provided by a 24-port full-backplane Gigabit Ethernet switch. All nodes run RedHat 9.0 operating system. In our work we used Portland Group C compiler version 5.0-2 and MPICH. All experiments were performed on an empty or lightly loaded machine. Results reported here, for all numbers of processors, represent best times obtained for multiple runs.

Since the available cluster had 40 processors and since we have used a binary summation algorithm, it was natural to utilize up to $2^5 = 32$ processors. Since the NEW-MPI-SUBDUE was implemented on the basis of the MPI-SUBDUE, we used the same master-slave approach and processor P_0 coordinated the execution of the program. Therefore, a total of 33 processors were used in the largest experiment. To obtain a complete performance picture each input data-graph was partitioned into $2, \ldots, 32$ partitions. First, in Tables 1-2 we present the time of graph partition of the original MPI-SUBDUE (sequential partition) and the NEW-MPI-SUBDUE (parallel partition). When experimenting the original MPI-SUBDUE, since there are two programs in the sequential partition, *pmetis* and *kmetis*, and *kmetis* is considerably faster than *pmetis* when the number of partitions is larger than 8 [2], we used *kmetis* for partitioning the graph into more than 8 subgraphs and *pmetis* for partitioning the graph into less than or equal to 8 subgraphs. In the NEW-MPI-SUBDUE, we used PARMETIS for graph partitioning.

Two observations can be made. First, as expected, parallel performance of PARMETIS is not very impressive, as we observe almost no speedup. However,

Table 2. Partition time for Graph 3

N	Execution time METIS (seconds)	Execution time PARMETIS (seconds)
2	1.009	0.700
4	1.268	0.621
8	1.269	0.597
16	1.425	0.599
32	1.626	0.601

thanks to its parallelism the partition time remains almost flat, whereas the time for the sequential partition increases with the number of partitions required. Therefore, second, when the total execution time of parallel SUBDUE is the shortest (for the largest number of processors), the sequential partition time is the longest and thus its effect (approached from Amdahl's Law perspective) is the most significant. However, third, the overall partition time is very small in comparison with the total execution time.

In Tables 3-5 we depict performance of the complete execution of the two versions of parallel SUBDUE obtained for running 1 iterations (finding the best substructure only). We also present the sizes of the largest and smallest partition and the actual performance gain.

The performance improvement of NEW-MPI-SUBDUE over the original MPI-SUBDUE is significant. While the performance gains of up to 68% in Graph 1 and 67% in Graph 2 were obtained, they are not very significant due to the short total execution time of the code. The realistic picture of performance gains is illustrated in the case of the large Graph 3. Here the gain ranges between 21% and 39% and slowly increases with the number of processors used.

The single-processor execution time of SUBDUE applied to Graph 3 is 2149 seconds and this means that a total speedup of approximately 268 is obtained on 32 processors. This results is easily explained when one considers the fact that when the data-graph is divided, then each processor operates only on a subgraph. Since formulas that express complexity of SUBDUE, while known to be polynomial in terms of the number of vertices and edges, depend also on the density of the graph, and thus we cannot easily utilize them in the case of

Table 3. Experimental results for Graph 1 for MPI-SUBDUE & NEW-MPI-SUBDUE

		Number of vertices		Execution time		
N	P	Minimum	Maximum	MPI-SUBDUE (seconds)	NEW-MPI-SUBDUE (seconds)	Improvement
2	3	1376	1468	28	9	68%
4	5	586	790	6	4	33%
8	9	244	412	2	2	0%
16	17	96	236	1	1	0%

Table 4. Experimental results for Graph 2 for MPI-SUBDUE & NEW-MPI-SUBDUE

		Number of vertices		Execution time		
N	P	Minimum	Maximum	MPI-SUBDUE (seconds)	NEW-MPI-SUBDUE (seconds)	Improvement
2	3	1404	1492	21	7	67%
4	5	614	790	6	5	17%
8	9	218	496	2	1	50%
16	17	84	248	1	1	0%

Table 5. Experimental results for Graph 3 for MPI-SUBDUE & NEW-MPI-SUBDUE

		Number of vertices		Execution time		
N	P	Minimum	Maximum	MPI-SUBDUE (seconds)	NEW-MPI-SUBDUE (seconds)	Improvement
2	3	10914	11354	653	514	21%
4	5	5362	5756	178	132	26%
8	9	2580	2932	58	45	22%
16	17	1218	1560	23	14	39%
32	33	458	870	12	8	33%

our graph. However, we can see in our final experiment (Graph 3) that increase of the number of processors by a factor of 2 results in each case in decrease of time by a factor of 4 and this gives us some indication about the complexity of the SUBDUE when applied to this particular data-graph. One could therefore suggest that even when 4 processors are used, graph could be divided into 32 partitions (8 partitions per processor) and in this way the total execution time reduced. Unfortunately, this approach may be counter-productive as it may lead to problems described above as by splitting the data-graph into unnecessary sub-graphs we may be loosing information about important substructures.

It is worth noting that the even though the mutagenesis dataset leads to a very sparse graph the results of applying PARMETIS are not very good (especially for large number of processors). Let us consider Graph 3 and $n = 32$ processors. In this case an optimal partition should be into subgraphs of size 696, whereas the actual minimum partition size was 458, while the maximum 870. This immediately points to a serious workload imbalance. If parallel SUBDUE is to be successful for even larger datasets, balancing sizes of subgraphs needs to become one of research priorities.

5 Conclusions

In this paper, by exploring data parallelisms and applying MPI collective communication, a NEW-MPI-SUBDUE algorithm was implemented and tested with three mutagenesis datasets (two small ones and one large). The experimental

results show that the NEW-MPI-SUBDUE algorithm provides performance that is approximately 20-30% better that the original MPI-SUBDUE.

References

1. Cook, D.J., Holder, L.B., Galal, G., Maglothin, R.: Approaches to Parallel Graph-Based Knowledge Discovery. Journal of Parallel and Distributed Computing, 61(3) (2001) 427-446
2. Karypis, G., Kumar, V.: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN (1998)
3. Karypis, G., Schloegel, K., Kumar, V.: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Ver. 3.1. University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN (2003)
4. Karypis, G., Kumar, V.: Multilevel K-way Partitioning Scheme for Irregular Graphs. Journal of Parallel and Distributed Computing, 48(1) (1998) 96-129
5. Karypis, G., Kumar, V.: A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing (1998)
6. Galal, G.M., Cook, D.J., Holder, L.B.: Improving Scalability in a Knowledge Discovery System by Exploiting Parallelism In the Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (1997) 171-174
7. Gorlatch, S.: Send-Receive Considered Harmful: Myth and Realities of Message Passing. ACM Transaction on Programming Languages and Systems, Vol. 26, No. 1. (January 2004)
8. http://cygnus.uta.edu/subdue/
9. http://www-ai.ijs.si/∼ilpnet2/apps/pm.html

Parallel Query Processing and Edge Ranking of Graphs

Dariusz Dereniowski* and Marek Kubale**

Department of Algorithms and System Modeling,
Gdańsk University of Technology, Poland
{deren, kubale}@eti.pg.gda.pl

Abstract. In this paper we deal with the problem of finding an optimal query execution plan in database systems. We improve the analysis of a polynomial-time approximation algorithm due to Makino et al. for designing query execution plans with almost optimal number of parallel steps. This algorithm is based on the concept of edge ranking of graphs. We use a new upper bound for the edge ranking number of a tree to derive a better approximation ratio for this algorithm. We also present some experimental results obtained during the tests of the algorithm on random graphs in order to compare the quality of both approximation ratios on average. Both theoretical analysis and experimental results indicate the superiority of our approach.

1 Introduction

A parallel query execution is a way for improving performance in large database systems. In general, there are three types of parallelism in database systems: *inter-query parallelism*, *inter-operator parallelism* and *intra-operator parallelism*. In all cases the goal is processor allocation but at different levels. In inter-query parallelism we have a set of queries (some of them can be executed concurrently) and we have to find their parallel scheduling. If more than one processor is assigned to a query then the two other mentioned types of parallelism can be exploited as well. In inter-operator parallelism we schedule operations within a single query. In this paper we consider joins only as they are the most time consuming operations. Thus, we schedule join operations by assigning processors to joins and determining which operations can be computed independently. In the last type of parallel scheduling, intra-operator parallelism, we split one join operation among several processors in order to compute it efficiently. In this paper we present results which can be applied in inter-operator parallelism.

In the inter-operator parallelism we have two main problems to consider: join sequence selection and allocation of processors to joins. In this paper we deal with the first issue. Furthermore, the algorithm under consideration does not deal with the join execution costs. To avoid large intermediate relations we can

* Supported in part by KBN grant 3 T11C 01127.
** Supported in part by KBN grant 4 T11C 04725.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 463–469, 2006.

use a technique described in [8], where the graph ranking approach can still be used.

Finding an optimal parallel *query execution plan* (QEP) is NP-hard. Thus, for complex queries the problem is intractable and most approaches are based on heuristics which generally have no formal analysis (such as worst case bounds) on their optimality [12]. Makino, Uno and Ibaraki [9] gave an algorithm with a worst case performance guarantee for solving this problem. In the sequel we will call their approach as the *MUI algorithm*. In the next section we describe the concept of edge ranking and the MUI algorithm. In Section 3 we improve on the prior analysis of MUI. Finally, Section 4 gives some experimental results.

2 Preliminaries

The *join graph* of a query is defined as a simple graph G such that the vertices of G correspond to the base relations and two vertices are adjacent in G if the corresponding relations have to be joined during the execution of a query [4, 8, 11]. In this paper we assume that the join operations are executed in steps. We are interested in finding parallel scheduling of the joins, so we assume that in each step many join operations can be performed. If a base relation is used in two different join operations (which means that in the query graph the edges which correspond to these operations are incident to a common vertex) then these joins cannot be computed in the same step. The scheduling of the join operations can be represented as a rooted tree T', where nodes of this tree represent the join operations and a particular join can be evaluated only if all its descendants have been already computed. Furthermore, if two nodes are unrelated in T' then the corresponding joins can be computed in parallel. The above data structure is known as the *operator tree*. For more detailed description of operator trees see e.g. [1, 5].

If an operator tree T' has been computed the join can be scheduled in such a way that the tree is processed in a bottom-up fashion – in the ith step all joins which correspond to the nodes in the ith level of T' are computed, where the last level is the root of T'. Note that the height of T' (the length of the longest path from the root to a leaf) is the number of parallel steps required to process the query. So, it is desirable to find a tree T' of small height. Note that if some relations x and y have been joined in the ith step (let z be the resulting intermediate relation) then in all joins in the later steps we use z instead of x and y.

In the following we assume that a join graph $G = (V(G), E(G))$ is given, where $|V(G)| = n$ and $|E(G)| = m(G) = m$. We denote by $\Delta(G)$ the maximum vertex degree of G, in short the *degree* of G. An *edge k-ranking* of G is a function $c : E(G) \to \{1, \ldots, k\}$ such that each path between edges x, y satisfying $c(x) = c(y)$ contains an edge with a greater color. The smallest integer k such that there exists an edge k-ranking of G is the *edge ranking number* of G and is denoted by $\chi'_r(G)$. Makino et al. [9] introduced the *minimum edge ranking spanning tree* (MERST) problem. In this problem the goal is to find a spanning tree whose ranking number is as small as possible.

Spanning trees and their edge rankings can be used to find good QEPs (which in our case is equivalent to finding low height separator trees) in the following way. In order to complete the execution of the query, we have to compute joins which correspond to edges forming a spanning tree in the query graph G. Indeed, if C is a cycle with l edges in G then computing any $l-1$ joins from that cycle results in an intermediate relation obtained by joining all base relations corresponding to the vertices of C. Furthermore, if we have selected some spanning tree T of G, then the task is to find an optimal parallel scheduling of joins corresponding to the edges of T. We find an edge ranking c of T and schedule the operations in such a way that for any two edges $e_1, e_2 \in E(T)$ the join corresponding to e_1 is the father of the join corresponding to e_2 in an operator tree T' if and only if $c(e_1) > c(e_2)$ and every path connecting e_2 to an edge e with $c(e) > c(e_1)$ passes through the edge e_1. So, the number of colors used by c is the height of the operator tree T' obtained in this way. Furthermore, this approach is optimal in the sense that if the spanning tree of the query graph is given then the height of an operator tree of minimum height equals to the edge ranking number of T [9].

Let us illustrate the above concepts in the following example. Consider a database query of the form

$$\begin{aligned}\text{Select * from } A, B, C, D, E, F \text{ where } A.x = C.x \text{ and } A.y = D.y \text{ and} \\ A.z = E.z \text{ and } A.t = B.t \text{ and } B.u = E.u \text{ and } B.v = F.v \ldots\end{aligned} \tag{1}$$

Fig. 1(a) depicts the corresponding query graph G containing edges $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$, $\{B, E\}$, $\{B, F\}$ which correspond to the join operations that have to be computed in order to complete the above query. The edges of a spanning tree T of G are marked with heavy lines in Fig. 1(a). Note that the degree of T is minimum over degrees of all spanning trees of G. Furthermore, this is an optimal solution to the MERST problem for the graph G. We compute an edge ranking of T in order to create an operator tree. An example of such a coloring is given in Fig. 1(a) (the numbers labeling the edges of T are the colors). The largest color used is equal to 3. Thus, this query can be computed in parallel in three steps. The operator tree T' created on the basis of this edge ranking is given in Fig. 1(b). Nodes of T' are labeled with the corresponding

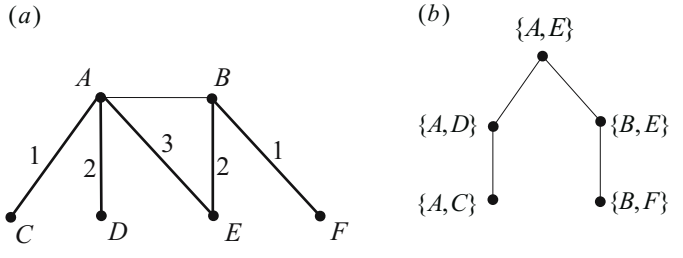


Fig. 1. (a) a query graph G, its spanning tree T and an edge ranking of T; (b) the corresponding operator tree T'

join operations. Finally, the operator tree can be used to create a parallel schedule as described above. In this case the join operations $\{A, C\}$ and $\{B, F\}$ are computed independently in the first step. In the second step we have operations $\{A, D\}$ and $\{B, E\}$. However, instead of using the relation A (B) we use the intermediate relation $\{A, C\}$ $(\{B, F\})$ created previously. In the final step we join the intermediate relations corresponding to the sons of the root. The first relation is the result of joining A, C, D and the other was obtained by joining B, E and F.

Though the MERST problem is polynomially solvable for threshold graphs [8], it is NP-hard for general graphs [9]. As we described above, the difficulty lies in finding a required spanning tree of the query graph. The MUI algorithm solves MERST for general graphs with sublinear approximation ratio

$$\frac{\min\{(\Delta^* - 1)\log n/\Delta^*, \Delta^* - 1\}}{\log(\Delta^* + 1) - 1},$$

where Δ^* is the degree of a spanning tree whose Δ is as small as possible, and log stands for $\log_2$ [9].

The algorithm MUI can be described as follows:

1. Find a spanning tree T of G with $\Delta(T) \leq \Delta^* + 1$.
2. Find an optimal edge ranking of T.

In the first step we use a 1-absolute approximation polynomial-time algorithm given in [3]. The problem of finding an optimal edge ranking of a tree is polynomially solvable [6].

3 Worst-Case Analysis of MERST

The following upper bounds for the edge ranking number of a tree are used to derive the corresponding approximation ratios of MUI. In the following $\Delta = \Delta(T)$.

Theorem 1. ([9]) *If T is a tree with degree equal to Δ then*

$$\chi'_r(T) = \lceil \log n \rceil, \quad \text{if } \Delta = 0, 1, 2$$

$$\chi'_r(T) \leq \frac{(\Delta - 2)\log n}{\log \Delta - 1} = B_1(T), \quad \text{if } \Delta > 2.$$

Theorem 2. ([2]) *If T is a tree with degree equal to $\Delta > 1$ then*

$$\chi'_r(T) \leq \Delta \log_\Delta m = B_2(T).$$

The following simple lower bounds are also used

Lemma 1. ([9]) *For any tree T, $\chi'_r(T) \geq \max\{\Delta, \lceil \log n \rceil\}$.*

If T_o is an optimal solution to MERST for G, and T is a tree produced in the first step of algorithm MUI, then on the basis of Lemma 1 and Theorem 1 we have [9]

$$\frac{\chi'_r(T)}{\chi'_r(T_o)} \leq \frac{(\Delta^* - 1)\log n/(\log(\Delta^* + 1) - 1)}{\max\{\Delta^*, \lceil \log n \rceil\}}$$
$$\leq \frac{\min\{(\Delta^* - 1)\log n/\Delta^*, \Delta^* - 1\}}{\log(\Delta^* + 1) - 1} = R_1(G). \quad (2)$$

Our approximation ratio for MUI, obtained on the basis of Lemma 1 and Theorem 2 is

$$\frac{\chi'_r(T)}{\chi'_r(T_o)} \leq \frac{(\Delta^* + 1)\log_{\Delta^*+1}(n-1)}{\max\{\Delta^*, \lceil \log n \rceil\}} = R_2(G). \quad (3)$$

Since $B_1 - B_2 = \Theta(\log n)$ for $\Delta \geq 4$ [2], by (2) and (3) we have $R_2(G) \leq R_1(G)$ for $\Delta^* \geq 3$. Note, that for a fixed value of Δ^* we have $R_1 = \Theta(\log n)$ and $R_2 = \Theta(1)$.

4 Experimental Results

Below we present some experimental results gained while testing MUI on random graphs. Each chart shows the values of parameters as functions of n, the size of graph. We generated graphs for each $n \in \{50, 60, \ldots, 150\}$. Each point denoted in the chart is the average of 100 values. All graphs created during the tests were connected. Note that if G is not connected then solving the MERST problem for G reduces to solving the problem for all connected components of G separately. Then a solution for G is a spanning forest and its edge ranking number is the maximum over the edge ranking numbers of the trees forming the forest. In order to create a random graph we constructed its random spanning tree and then, according to the graph density g, more edges were added. This means that for small g (i.e. $g = 0.005$) the random graphs were trees. Fig. 2 shows the results of the computer experiments.

Fig. 2(a) depicts the relation between the edge ranking number of a tree and the bounds B_1 and B_2. The degree of all trees generated was equal to 10. Two types of inaccuracies are included in the approximation ratios considered in this paper. The first follows from the fact that the MUI algorithm uses a heuristic for finding a minimum degree spanning tree. However, it is possible to create examples of graphs for which a spanning tree with minimum vertex degree is not the one with minimum edge ranking number. Then, there is some inaccuracy in bounding the edge ranking number when only n and Δ are given. Thus, the purpose of comparing the edge ranking number with its bounds is that the quality of the approximation ratio directly depends on the bounds used.

The second test is presented in Fig. 2(b), where we tested only the first part of MUI, i.e. we generated random graphs G with $\Delta(G) = 10$ and densities

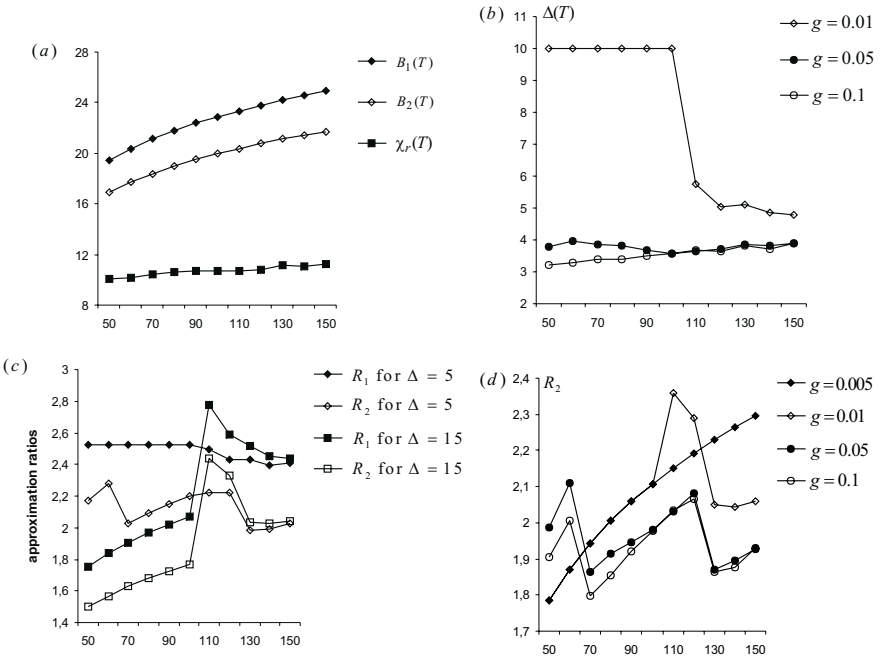


Fig. 2. Experimental results

$g = 0.01, 0.05, 0.1$, respectively. We used only small values of g, because as the experiments show, already for these values of g, the degree of the spanning trees is very small. From the theoretical analysis it follows that once we obtain a spanning tree which is a path, the problem has been solved optimally, so we want to avoid such cases. Some values of the first function for $g = 0.05$ are equal to 10, because random graphs obtained in these cases were trees.

Fig. 2(c) compares approximation ratios R_1 and R_2. In this test the graph density $g = 0.01$. We used a small value of g, because as the previous experiment shows, for larger g the average over degrees of T was smaller than 4 and in that case we would rather use the theoretical analysis from Section 3 to compare R_1 and R_2.

Fig. 2(d) presents the approximation ratio R_2. In this test, for each value of n and g we generated random graphs G with $\Delta(G) = 10$. Then, we computed R_2 for the spanning tree obtained in the first part of MUI. As before, each point is the average of 100 values (i.e. 100 different random graphs G were created). The first function, because of the small value of g, presents R_2 for trees with $\Delta = 10$. For smaller values of n and $g = 0.01$ graphs G are also trees. Larger graphs G are not acyclic. The approximation ratio R_2 is clearly not monotonic in Δ^*, which explains why the second function presented in Fig. 2(d) can get smaller and bigger values than the first one.

5 Summary

In Section 3 we gave a theoretical analysis which indicates that the new approximation ratio better describes the worst case behavior of the MUI algorithm. On then other hand, the experimental results presented in Section 4 compare both bounds in the average case. The mean value of R_1 (taken over all test cases) was 15% bigger than the mean value of R_2.

References

1. S. Chaudhuri, An overview of query optimization in relational systems, *Proc. PODS* (1998), Seattle (WA), USA.
2. D. Dereniowski, M. Kubale, Efficient parallel query processing by graph ranking, *Fundamenta Informaticae* (submitted).
3. M. Furer, B. Raghavachari, Approximating the minimum-degree Steiner tree to within one of optimal, *J. Algorithms* **17** (1994) 409-423
4. T. Ibaraki, T. Kameda, On the optimal nesting order for computing N-relational joins, *ACM Transactions on Database Systems* **9** (1984) 482-502.
5. M. Kremer, J. Gryz, A survey of query optimization in parallel databases, *Technical Report* CS-1999-04 York University (1999).
6. T.W. Lam, F.L. Yue, Optimal edge ranking of trees in linear time, *Proc. of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (1998) 436-445.
7. H. Lu, M.-C. Shan, K.-L. Tan, Optimization of multi-way join queries for parallel execution, *Proc. of the 17th Conference on Very Large Data Bases*, Barcelona, Spain, September 1991.
8. K. Makino, Y, Uno, T. Ibaraki, Minimum edge ranking spanning trees of threshold graphs, *LNCS* **2518** (2002) 428-440.
9. K. Makino, Y, Uno, T. Ibaraki, On minimum edge ranking spanning trees, *J. Algorithms* **38** (2001) 411-437.
10. P. de la Torre, R. Greenlaw, A.A. Schaffer, Optimal edge ranking of trees in polynomial time, *Algorithmica* **13** (1995) 529-618.
11. J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, Vol. 1. Computer Science Press, Maryland, 1990.
12. P.S. Yu, M.-S. Chen, J. L. Wolf, J. Turek, Parallel query processing In: N.R. Adam, B.K. Bhargava, editors, *Advanced Database Systems*, *LNCS* **759** Springer-Verlag, 1993.
13. X. Zhou, M.A. Kashem, T. Nishizeki, Generalized edge-rankings of trees, *LNCS* **1197** (1997) 390-404.

Online Balancing of aR-Tree Indexed Distributed Spatial Data Warehouse

Marcin Gorawski and Robert Chechelski

Silesian University of Technology,
Institute of Computer Science,
Akademicka 16, 44-100 Gliwice, Poland
{Marcin.Gorawski, Robert.Chechelski}@polsl.pl

Abstract. One of the key requirements of data warehouses is query response time. Amongst all methods of improving query performance, parallel processing (especially in shared nothing class) is one of the giving practically unlimited system's scaling possibility. The complexity of data warehouse systems is very high with respect to system structure, data model and many mechanisms used, which have a strong influence on the overall performance. The main problem in a parallel data warehouse balancing is data allocation between system nodes. The problem is growing when nodes have different computational characteristics. In this paper we present an algorithm of balancing distributed data warehouse built on shared nothing architecture. Balancing is realized by iterative setting dataset size stored in each node. We employ some well known data allocation schemes using space filling curves: Hilbert and Peano. We provide a collection of system tests results and its analysis that confirm the possibility of a balancing algorithm realization in a proposed way.

1 Introduction

A distributed spatial data warehouse system (DSDW) gathers telemetric information about media utility consumption (gas, energy, water, heat) [5]. The data warehouse is a part of a decision support system that is one of two layers in an infrastructure built to provide a forecasts of media consumption. The second layer is a telemetric system of integrated meters reading.

The telemetric system of integrated meters reading is based on AMR technology (Automated Meter Reading). The telemetric system enables data to be transferred from media consumption meters (localized on a huge geographical area) to the telemetric server using a GSM cellular phone network with GPRS technology. The distributed spatial data warehouse system is a part of a decision support system for making tactical decisions about the media production amount based on a short-term prognosis of its use. Forecasts are made exploiting the analysis of the data stored in the data warehouse (supplied with data from a telemetric server). The telemetric server is a source of measured data which are loaded into the DSDW database during an extraction process.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 470–477, 2006.

1.1 Architecture of Distributed Spatial Data Warehouse, the Description of Used Data Model

The DSDW system has a shared nothing type distributed architecture, where every node has its own processor, a memory area, and a disc [9]. All the communication between the nodes is done only through network connections (messages passing) with RMI technology (Remote Method Invocation). Shared nothing architecture offers unlimited scaling possibilities and almost linear speedup coming along with system enlargement. One of more important problems during exploiting this architecture in DBMS systems is data allocation amongst nodes that has strong influence on final performance.

The data warehouse system is composed of a set of nodes. Every node is equipped in an independent Oracle 9i RDBMS. Data in each node is organized in cascaded star model and indexed with an aR-tree. The data model and the index have the same structure across the system nodes.

Telemetric data gathered in the system has a spatial dimension and a time dimension. The spatial dimension consists of three attributes of space location (X, Y, Z) describing the localization of meters [6]. The time dimension contains dates of measurements. The system supports a few types of meters, each of them has a different frequency of readings, that cause the granularity in time dimension to vary.

An aR-tree with a virtual memory mechanism was added to the system to deal with summarized data, which are the most important in the DSDW system. Hence, aggregates are computed as they arrive during query execution (unless they were computed in a response to previous queries). Such construction of the index causes that the query determines which data will be drawn to an aggregation [5]. The next advantage of this index structure is that the time taken to draw data already aggregated is very small in comparison to the time taken for aggregation, when large amounts of detailed data are pulled out from the data warehouse.

1.2 The DSDW System Balancing Fundamentals

Our previous solutions for balancing the DSDW system concerned a case when the system consisted of nodes having the same parameters or the same performance characteristics. The essence of the problem considered in this work was the DSDW system balancing which is based on computers with different performance characteristics. The differences concern the computational capacity, size of operational memory, and I/O subsystem throughput.

The rest of the paper is organized as follows. Section 2 contains the motivation for our algorithm. Section 3 provides a detailed description of the algorithm of balancing, and section 4 presents and discuses the results of performed systems tests. Conclusions and discussion on future work are contained in section 5.

2 Algorithm Motivation

To improve the efficiency of the data warehouses realized in shared nothing architecture all the nodes have to be involved in realization of every query. Optimal speedup, and the same minimal response time, is achieved when the work time of all nodes is the same. Each node in such a system computes its answer only on contained piece of data. Huge amounts of data makes impossible to relocate data during query execution. The source data for the DSDW system are partitioned among N nodes, so that every node possesses the same data schema. The fact table, which usually takes more than 90% of the space occupied by all the tables, is split into N partitions loaded into nodes. The dimension tables are replicated into all nodes in the same form [1].

In this paper we present a balancing method based upon setting the size of partitions that are loaded into the system nodes (the fact table is partitioned). The following questions need to be addressed to effectively balance the data warehouse: how to manage nodes differences, how to mark efficiency of a node, how to link between them.

To manage nodes differences we decided to arrange data allocation amongst nodes and to set partition sizes. To this end we exploited some well known data allocation schemes using space filling curves. The curve visits every point of space exactly once and never crosses itself and is used for the linear ordering of k-dimensional space [4]. In these schemes the curve selects the next grid block which is in turn assigned to a node in round-robin fashion [2, 7]. Two types of curves: Peano and Hilbert [3] were implemented, and schemes fashion was changed in a way which enables to control the dataset sizes stored in all nodes.

Node efficiency is marked by an execution of a single range query. The realization of such query must draw and aggregate all detailed data stored in system nodes. The measure of efficiency is the obtained aggregation time.

As a link mechanism we propose a simple adjusting of dataset size, based on imbalance values of particular nodes. During balancing we use the test set that is a subset of the fact table and is chosen to the query marking nodes performance. The algorithm iteratively calculates partition sizes by executing a series of size computation, loading and aggregation (query execution). The algorithm is applicable only before the main warehouse run, when system is tuned up. We have made some additional presumptions: characteristics of the nodes are constant and communication costs are very low (omitted).

3 System Balancing Algorithm

In order to catch a real differences between system nodes we use the concept of a fact table division factor – p_i. It represents a value that is a ratio of the fact table size stored into an i node to total size of fact table. Using the aggregation times of the test set, obtained by every node, the algorithm iteratively computes values of p_i factors in order to obtain a work time of a node similar to the mean work time [10].

1. Load dimension tables to all nodes in the same form,
2. Set all fact table division factors as $p_i = 1/N$,
3. Load test subset of fact table, partitioned according to division factors, to all nodes,
4. Aggregate all test subset data,
5. Maximum imbalance is less than assumed ? Yes – go to 7, No – go to 6,
6. Correct division factors p_i using aggregation times, go to 3,
7. Load the whole fact table, partitioned according to last computed division factors, into all nodes.

In the first step the algorithm loads dimension tables to all nodes; the same copy of dimension tables is loaded into each node. Initially, all factors p_i are set to the value 1/N where N indicates the total number of nodes. Next step of the algorithm is a fact table partition plan calculation (first part of step 3 of balancing algorithm). A code from [8] was used to compute h-values (the transformation of k-dimensional space into linear with the use of the Hilbert curve).

The fact table partition plan making algorithm:

1. Calculate h/z-value for a localization of each meter,
2. Sort all meters by h/z-value ascendant,
3. Allocate grid blocks to nodes using the round-robin method with skipping – to preserve the value of divide factor for i node.

Then fact table is sent to all nodes using a previously prepared partition plan (splitting of the fact table into partitions loaded into particular nodes). Moreover, this step respects the precalculated dataset sizes for each node.

The next step is an aggregation that is performed using a range query supplied by the user. Every iteration uses the same query. On the basis of obtained aggregation times, the algorithm calculates the imbalances of all system nodes in relation to the fastest one – according to the formula (1)

$$imbalance_i = \frac{T_i - T_{fast}}{T_{fast}} \tag{1}$$

where: T_i and T_{fast} are aggregation times of i^{th} and the fastest node.
If the maximum imbalance (amongst all nodes) is greater than assumed, then a correction of fact table division factors (p_i) is performed (according to algorithm presented below). Beside the aggregation times, the correction algorithm takes into account two additional factors: $corrP$, used when the division factor is increased, and $corrN$, used when the factor is decreased.

The division factors correction algorithm:

1. Calculate an average aggregation time of all nodes – avg
2. For each node calculate its imbalance of aggregation time with relation to avg. The imbalance is used next to correction of p_i factors. Imbalance is calculated according to formula (2).

$$imbalance_i = \frac{T_i - avg}{avg} \tag{2}$$

(a) if imb > 0 then make correction using formula (3)

$$p_i = p_i \cdot (1 - corrN \cdot imb) \tag{3}$$

(b) if imb < 0 then make correction using formula (4)

$$p_i = p_i \cdot (1 - corrP \cdot imb) \tag{4}$$

(c) if imb $= 0$ then do not make correction

3. Correct all factors to 100%

When the maximum imbalance is below the assumed value, the whole fact table is partitioned and loaded into nodes using division factors calculated during the last iteration of the balancing algorithm (by exploiting the previously presented the making fact table partition plan).

4 Tests

The test platform was composed of 6 nodes (5 servers and one server/maintainer). Source data was stored in the maintainer node which was responsible for realizing the program of the tests, especially balancing (implementing balancing algorithm described in section 3). Servers were performing operations ordered by the maintainer on their copy of data. Nodes were configured as follows: 3 nodes – 2.8 GHz Intel Pentium 4, 512 MB RAM, 120GB 7200 RPM IDE HDD, 2 nodes – 3.2 GHz Intel Pentium 4, 1 GB RAM, 120GB 7200 RPM IDE HDD, 1 node – 1.7 GHz Intel Pentium 4, 256 MB RAM, 80GB 7200 RPM IDE HDD. The maintainer software was run on one of the three early mentioned nodes. Each node worked under Windows XP, had an installed Oracle9i server and Java Virtual Machine v. 1.4.2_04. All the nodes were connected by a 100Mbit Ethernet network.

The fact table stored in the system had 9.4 millions rows. During balancing two test subsets were used, first having 0.93 millions and second having 1.87 millions rows (later called respectively the large test subset and the small test subset). The tests consist of a dozen or so balancing processes, after each of them two queries were performed in order to evaluate the quality of the balancing process. The first query draws out data from a few large regions (later called large-region-query), while second query draws data from a dozen small regions (later called small-region-query). In descriptions of the tests we are using the following signs: h – balancing with use of Hilbert curve, p - with use of Peano curve; 2d – ordering in two dimensions, 3d - ordering in three dimensions; a/b – corrN and corrP factors respectively. Stop condition of each one balancing process was decreasing imbalance below 10% or reaching ten process iteration.

The analysis of maximum imbalance graphs (maximum imbalance amongst all nodes, in every iteration of the balancing process) presented in figure 1 shows that the time taken for all processes of balancing using the Peano curve were shorter than corresponding processes when the Hilbert curve was used. The

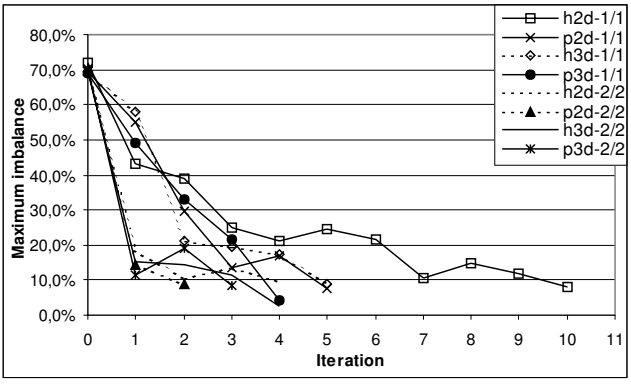


Fig. 1. Charts of maximum imbalance for the balancing processes with the use of the small test subset

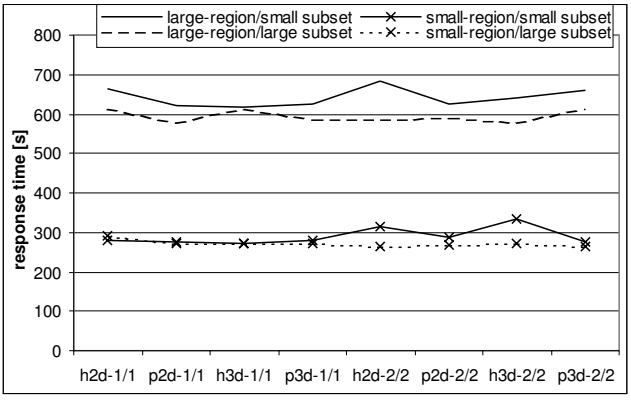


Fig. 2. System response times on large-region- and small-region-queries after balancing with the use of small and large test subset

use of bigger correction factor values in all cases resulted in the shortening of balancing process duration.

The next analysis concerns the realization of queries sent to system after finishing each balancing process. The charts presenting response time for large and small-region-queries and for both cases of test subset are shown in figure 2, while corresponding charts of maximum imbalance in figure 3. Comparing the response times on particular queries – the system balanced with use of either a small or large test subset, either a large- or small-region-query submitted to process – we see that the response time does not depend on correction factor values used during balancing. The next observation is the fact that system response times on region queries when the Hilbert curve was used to balance are generally higher than response times on queries after analogical balancing with the use of

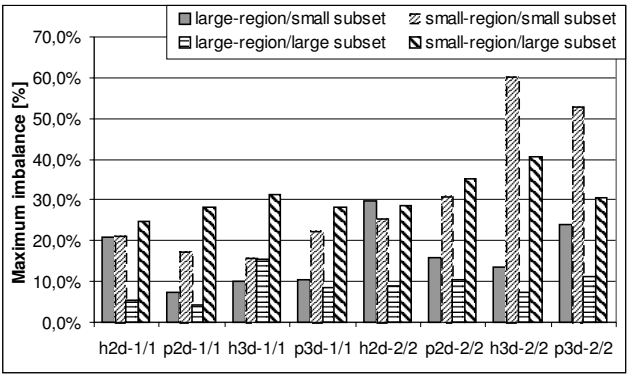


Fig. 3. Maximum imbalances of response times on large-region- and small-region-queries after balancing with the use of small and large test subset

the Peano curve. Furthermore, there is no influence of ordering type (two or three dimensional) on the system's response time and imbalances of submitted region-queries realization.

While analyzing the charts of maximum imbalance (fig. 2) we see that large-region-queries result in a low mean imbalance and a smother graph (imbalances are similar). The reason for this is the fact that small queries could receive regions stored in a part of system nodes, so only part of the nodes are involved in its realization. For larger queries there is higher probability that all system nodes will participate in computations.

The comparison of response times and imbalance charts for the small and large test subset shows that system imbalance after processing large-region-queries was lower when larger test subset was used for balancing. Moreover, the mean response time after that balancing is lower than after balancing with the second test subset.

In response to small-region-queries system had similar mean imbalances in the cases of both subsets, nonetheless the mean response time is significantly lower for queries sent to process after balancing with larger test subset.

5 Conclusions

In this paper we have presented a spatial telemetric data warehouse balancing algorithm. The considered data warehouse is made of PC computers having different computational characteristics. The presented results confirmed the possibility of warehouse balancing by using the method of setting node data set sizes. Described algorithm iteratively selects dataset sizes using the query realization time on test subset of the fact table. The comparison of two space filling curves used to ordering – Hilbert and Peano – shows that both of them give similar results, but Peano is slightly better to employ in our system.

The set of results presented in this work shows that exploiting larger data subsets to system's balancing, improves the mean time of system's response and decreases the imbalance of the system. The improvement does not depend on curve type and used correction factor values. The values only have an impact on the balancing process duration time.

The presented algorithm could be developed in the following directions:

- balancing of the data warehouse during data updating,
- looking for other mechanisms of nodes efficiency evaluation and linking with partition size,
- improvement of response time for small range queries.

References

1. Bernardino J., Madeira H.: Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing. CAiSE*00 Conference on Advanced Information Systems Engineering Stockholm, 2000
2. Dehne F., Eavis T., Rau-Chaplin A.: Parallel Multi-Dimensional ROLAP Indexing. 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan 2003.
3. C. Faloutsos and P. Bhagwat: Declustering using fractals in Proc. of the Int'l Conf. on Parallel and Distributed Information Systems, San Diego, California, January 1993, pp. 18-25.
4. Faloutsos C., Roseman S.: Fractals for Secondary Key Retrieval. Technical Report UMIACS-TR-89-47, CS-TR-2242, University of Maryland, Colledge Park, Maryland, May 1989
5. Gorawski M., Malczok R.: Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree, 5th Workshop on Spatial-Temporal DataBase Management (STDBM_VLDB'04), Toronto, Canada 2004.
6. Han, J., Stefanovic, N., Koperski, K. Selective Materialization: An Efficient Method for Spatial Data Cube Construction. In Research and Development in Knowledge Discovery and Data Mining, Second Pacific-Asia Conference, PAKDD'98, Melbourne, Australia, 1998.
7. Hua K., Lo Y., Young H.: GeMDA: A Multidimensional Data Partitioning Technique for Multiprocessor Database Systems. Distributed and Parallel Databases, 9, 211-236, University of Florida, 2001
8. Moore D. Fast hilbert curve generation, sorting, and range queries. `http://www.caam.rice.edu/~dougm/twiddle/Hilbert`
9. Papadias D., Kalnis P., Zhang J., Tao Y.: Efficient OLAP Operations in Spatial Data Warehouses. Spinger Verlag, LNCS 2001.
10. Zeng Z., Bharadwaj V.: Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks. Computer Communications, 27, pp. 679-694, 2004

Resumption of Data Extraction Process in Parallel Data Warehouses

Marcin Gorawski and Pawel Marks

Silesian University of Technology, Institute of Computer Science,
Akademicka 16 street, 44-101 Gliwice, Poland
{Marcin.Gorawski, Pawel.Marks}@polsl.pl

Abstract. ETL processes are sometimes interrupted by occurrence of a failure. In such a case, one of the interrupted extraction resumption algorithms is usually used. In this paper we present a modified Design-Resume (DR) algorithm enriched by the possibility of handling ETL processes containing many loading nodes. We use the DR algorithm to resume a parallel data warehouse load process. The key feature of this algorithm is that it does not impose additional overhead on the normal ETL process. In our work we modify the algorithm to work with more than one loading node, which increases the efficiency of the resumption process. Based on the results of performed tests, the benefits of our improvements are discussed.

1 Introduction

Processing large amounts of data for data warehouse during ETL process (*Extraction Transformation and Loading*) takes a very long time, several hours or even days. There is usually a relatively small time window fixed for a whole extraction. The more data to be processed, the longer the ETL process. When for some reason an extraction is interrupted, for example by hardware failure or no power supply, the extraction must be restarted. Such a situation is not rare. Sagent Technologies reports that every thirty extraction process is interrupted by a failure [9]. After an interruption there is usually no time left for running the extraction from the beginning. In this case, the most efficient solution is to apply one of the interrupted extraction resumption algorithms. In this paper we analyze the standard and our modified Design-Resume [6] algorithm (DR). The modified DR handles extraction graphs containing many extractors and many inserters.

Most commercial tools [2, 8] do not consider the internal structure of transformations and the graph architecture of ETL processes. Exceptions are researches [10, 11], where the authors describe the ETL ARKTOS (ARKTOS II) tool. To optimize ETL process, there is often designed a dedicated extraction application adjusted to requirements of a particular data warehouse system. Our experience prompted the decision to build developmental ETL environment using JavaBeans components [4]. In the meantime, a similar approach was proposed in

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 478–485, 2006.

paper [1]. Presented therein was J2EE architecture with an ETL and an ETL-Let container, providing efficient execution, controlling, and monitoring of ETL tasks during continuous data propagation.

Further speeding up of the ETL process forced us to abandon JavaBeans platform. The ETL-DR environment succeeds the previous ETL/JB [4] (JavaBeans ETL environment) and DR/JB [5] (ETL environment with DR resumption support). The new ETL-DR environment is a set of Java object classes, used by a designer to build extraction and resumption applications. This is analogous to JavaBeans components in the DR/JB environment. In the DR/JB we implemented an estimation mechanism detecting cases where the use of DR resumption is inefficient. Unfortunately, the model we used did not take into account many significant external factors, like virtual memory usage. In the ETL-DR the situation changed. We improved the implementation of the DR filters, which resulted in reduction of resumption inefficiency. Now the resumption is almost always faster than restarting the extraction from the beginning. Moreover, a lot of problems appeared in graphs containing many loading nodes. Hence, we decided to stop research on this mechanism. Another direction of our research is combining the DR resumption with techniques like staging and checkpointing. The combination of DR and staging, we named it hybrid resumption. This approach performs slightly better than pure DR algorithm but its efficiency depends on the extraction graph structure. Research on checkpointing is still in progress.

The main goal of all the extensions of the DR algorithm is to resume the ETL processes containing many loading nodes as efficiently as possible. Briefly described in Sect. 2 are the basic and modified DR algorithms. In Sect. 3 is our test environment; its resumption tests with results and short comments are described in Sect. 4. In Sect. 5 the obtained results are discussed and the paper is summarized. References are included in the last section.

2 Design-Resume Algorithm

2.1 Basic Version of the Algorithm

The Design-Resume [6, 7] algorithm (DR) recovers extraction processes from system failures. It works using properties assigned to each node of the extraction graph and data already loaded to a destination prior to a failure. This algorithm belongs to the group of redo algorithms [6], but during resumption it uses additional filters that remove from the data stream all tuples contributing to the tuples already loaded. An extraction graph is a directed acyclic graph (DAG), whose nodes process data and whose edges define data flow directions.

The algorithm is divided into two phases. In phase 1 the extraction graph is analyzed and additional filters are assigned (Design procedure). In phase 2 the filters are initialized with data already loaded into a data warehouse (Resume procedure). The most important feature of the DR algorithm is that it does not impose any additional overhead on the uninterrupted extraction process. Algorithms like staging or savepointing usually increase processing time a lot (even several times). The drawback of the DR algorithm is that it cannot be

applied if before a failure no output data were produced. Design procedure is usually run once to assign filters that are needed during resumption. The Resume procedure is run each time the ETL process must be resumed. It recovers data already loaded from a destination, and uses the data to initialize additional filters assigned in phase 1.

The basic DR algorithm, denoted later as DR(1), can resume only single-inserter (an inserter is a loading node) extraction graph. This limitation lowers the resumption efficiency, and encouraged us to modify the algorithm.

2.2 The Algorithm Modifications

The DR algorithm's limitation disallowing the resumption of ETL processes with many loading nodes is quite burdensome. The extraction graph must be divided into segments containing one inserter each. Usually the data loaded by one of the inserters are transformed by the same set of transformations as the data loaded by another inserter. It is 100% true for parallel warehouses where some of the tables are duplicated on the machines the warehouse consists of. Dividing the graph into smaller parts and running them independently forces some transformations to be run twice or more to do the same job. This waste of time would be eliminated if the original extraction graph was used for resumption. To improve the algorithm we modified both Design and Resume procedures.

The first modification was analysis of which inserters are necessary during resumption, and which have already completed the loading process. It is possible because each inserter after loading, writes its ID into the list of inserters that finished the processing. The list is kept in an external disk file. Before resumption the list is loaded from the file, and all inserters found on the list are removed from the extraction graph. After removing the inserters, the transformations that lost its destination nodes are removed also. This way we obtain the resumption graph containing only the nodes producing missing data. This modification has the biggest impact on the overall resumption efficiency. It also forces DR filters assignation routine to be run before each resumption, because the graph structure changes.

Next we introduced a new transitive property *sameSuffix*. It was mentioned by DR algorithm authors in [6] but it was not developed. This property gives a chance to assign DR filter more optimally, and closer to extractors. It is held when the tuple stream differs from the one before a failure but still contains contributors of the last loaded tuple and order of tuples is unchanged. Its influence is mostly visible when processing path divides into many paths performing similar operations but loading data by separate inserters. Such a situation occurs in parallel data warehouses.

After defining the new property we changed filter assignation rules. Now data stream can be filtered in a common part of the graph even if it is targeted to many inserters. We also had to modify the definition of the redundant filter, and change the DR filters structure [6].

We also added another small extension, namely a *subsetNotRecommended* hint for the Design procedure. The hint is applied to a particular inserter and it

denotes that using the subset filtration based on the data taken from a hinted inserter is not suggested. Subset filtration [6] is one the two possible filtration methods used in the DR resumption. It uses a complete tuple set taken from the inserter. Each tuple provided to the DR filter input is checked if it contributes to any tuple from the set taken from inserter. This method can be inefficient when the set to be searched is very large. In such a case it is sometimes better not to filter anything this way and do the filtration later when prefix filtration is possible. Prefix filters [6] needs to know only the last loaded tuple. They compare tuples from the input with the one taken from the inserter. All the tuples are rejected until the one corresponding to the last loaded is found. Then filter stops rejecting tuples, and the rest of the tuple stream remains unchanged. Obviously it is not always possible not to use subset filters. In such a case, the hints for the inserters that cannot use prefix filtration are ignored.

3 ETL-DR Extraction Environment

The ETL-DR environment is written in Java; it is a successor to the DR/JB environment (Sect. 1). However, in the new environment we do not use JavaBeans technology. We gambled on simplicity, efficiency, and ease of further development. The ETL-DR is a set of classes used by the graph designer to create an extraction application. These are analogous to JavaBeans components in the DR/JB environment. In comparison to the DR/JB, we significantly improved the processing efficiency and the complexity of the most important transformations, namely DR filters, grouping and joining.

Besides increasing the processing efficiency of the ETL-DR environment, the most important fact is that we implemented a modified DR algorithm within it (Sect. 2.2). It enables the resumption of interrupted extraction processes described by graphs containing many loading nodes. Moreover, to take advantage of all the DR algorithm features, we use a modified algorithm of loading data into a database. Tuples are loaded in a way making the inserter suffix-safe [6], which means that the last loaded tuple is known. This can speed up the resumption process significantly.

4 Tests

4.1 Test Conditions

The base for our tests is an extraction graph containing 4 extractors and 15 inserters (loading nodes). The graph consists of three independent parts, but it is seen by the extraction application as a single ETL process. To load data into a distributed data warehouse consisting of 5 PC machines we had to increase the number of inserters in the graph. Each inserter loads data into a single database table. Finally we obtained the graph containing 75 inserters.

The ETL process generates a complete data warehouse structure (Fig. 1). It is a distributed spatial data warehouse system designed for storing and analyzing

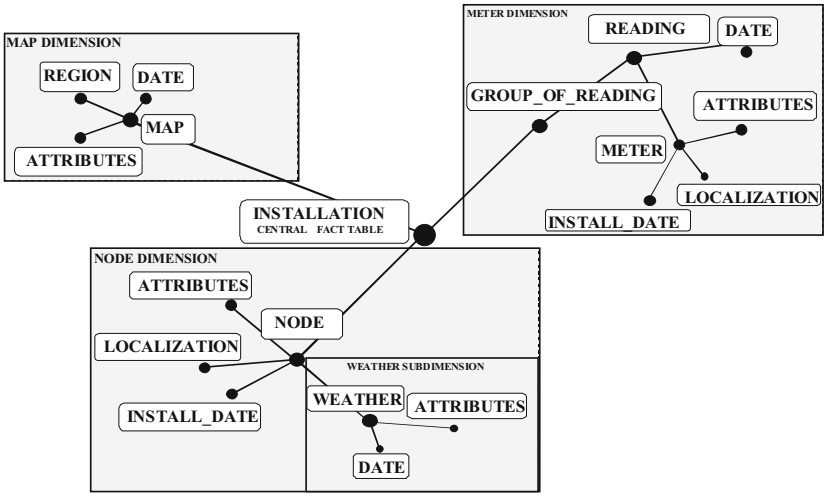


Fig. 1. Schema of the generated data warehouse

a wide range of spatial data [3]. The data is generated by media meters working in a radio-based measurement system. All the data is gathered in a telemetric server, from which it can be fetched to fill the data warehouse. The distributed system is based on a new model called the cascaded star model. The test input data set size is 500MB.

Data set is distributed over the 5 PC machines. All the dimension tables are replicated and each machine gathers the same data set. The fact table containing measurements from remote meters is divided into 5 more or less equal parts. The meter ID determines the target machine. The inserters loading into fact tables are preceded by additional filter nodes. Their task is to filter the tuples according to the distribution criteria (meter IDs).

Due to space limitations we cannot present the structure of a whole graph. We can only try to describe it shortly. Extractors read the data from four source files:

- meters parameters (meterID, nodeID, locX, locY, locZ, type, scope, monDate),
- weather changes around collecting nodes (nodeID, temperature, humidity, clouds, measDate),
- collecting nodes parameters (nodeID, locX, locY, locZ, monDate),
- set of meter measurements records (meterID, date, time, zone1, zone2).

Basing of the four source files, the complete data warehouse containing 15 tables is generated. METERS, NODES and WEATHER dimensions consists of the main table containing record describing particular meters, nodes or weather states, and also additional sub-dimension tables with necessary attributes. The required identifiers are created during ETL process. Fact table contains measurements from gas, energy and water meters. Some of them measure values in

two separate zones. Such records are transformed into records containing one measured value each with the zone number. One more thing that ETL process creates is a set of measurement groups, meters of the same type belonging to the same collecting node are put into the same group. Such a division is useful during analysis performed by the DSS system that uses the warehouse.

During each test the extraction process was interrupted in order to simulate a failure. The resumption process was then run and the time was measured. The collected measurement results permitted us to prepare resumption charts showing the resumption efficiency depending on the time of a failure.

For the tests we used the following machines:

1. for ETL software:
   - 1x PIV 2.8GHz 512MB RAM, Windows XP Prof, J2SDK 1.4.2_06
2. for distributed data warehouse based on Oracle 9i database:
   - 1x PIV 2.8GHz 512MB RAM, Windows XP Prof
   - 2x PIV 2.8GHz 512MB RAM, Windows 2000
   - 2x PIV HT 3.2GHz 1GB RAM, Windows XP Prof

Communication with the database was implemented using JDBC interface and Oracle OCI drivers. Oracle SQL*Loader was also used for fast data loading into database tables. Portions of data were loaded into temporary tables by SQL*Loader, and simple INSERT queries moved the packets into target tables. A single uninterrupted extraction process lasts about 65 minutes.

4.2 Modified DR Algorithm Tests

The goal of the test is to stress the difference between the efficiency of the standard version of the Design-Resume algorithm marked as DR(1), and our modified version, named DR(m). The test is based on the extraction graph presented in Sect. 4.1.

As can be seen in Fig. 2, the resumption based on the DR(1) algorithm can be very inefficient. This is due to the necessity of running the resumption for each inserter separately. Thus many transformations have to be run several times, each time doing the same job. Such waste of time does not take place during DR(m) resumption. In this case, the transformations are run only once, and they produce tuples for all the inserters at the same time. The figure show, that in our test, DR(1) resumption can be even eight times slower than the DR(m) method, that performs much faster.

Normal uninterrupted extraction process takes up to 65 minutes (about 4000 seconds). DR(m) resumption does not exceed this time, and we can say that no matter when the failure occurs, it is always profitable to run the resumption. Unfortunately DR(1) resumption efficiency is unacceptable. It lengthens the processing about eight times in the worst case, only for failures at the end of the extraction process it can save some extraction time but it is still no more than 10-15%. The reasons for this inefficiency are:

- running the same transformations for each inserter separately,
- when one of the inserters loads data to its destination machine, the other machines do nothing wasting time and the computation power.

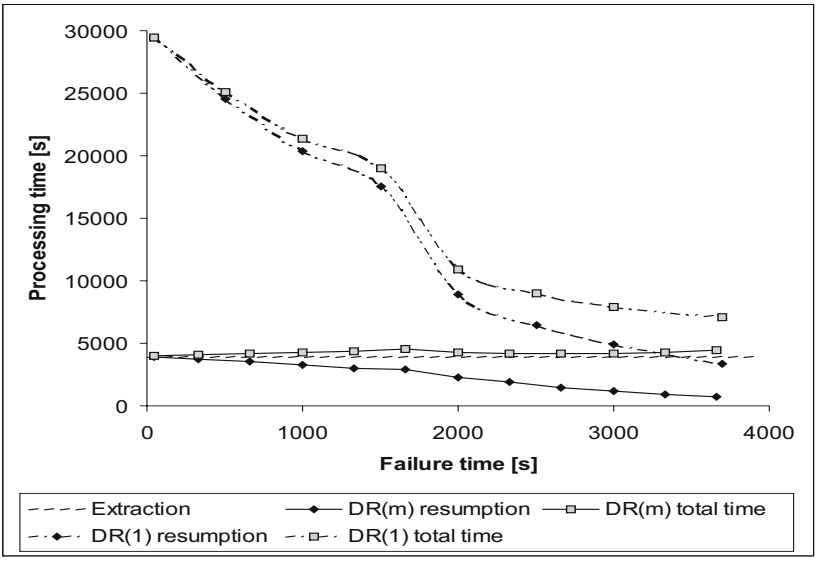


Fig. 2. Resumption efficiency. Resumption is the time needed to end the extraction after a failure. Total processing time is the sum of the time before occurrence of failure and the resumption time.

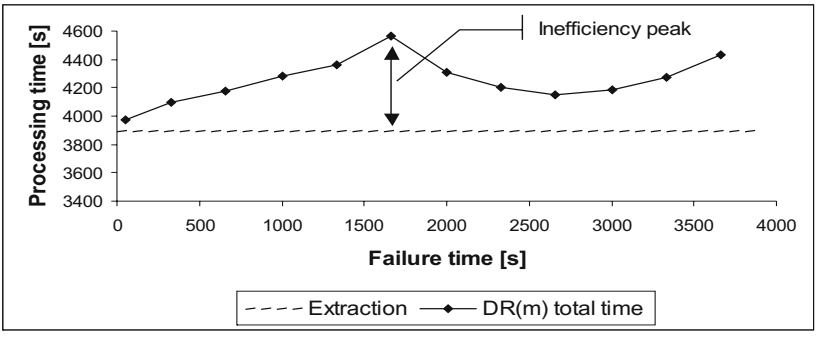


Fig. 3. More detailed analysis of DR(m) resumption efficiency

Figure 2 compares also the total processing times depending on the resumption algorithm. As one may observe, the curve for DR(m) resumption stays close to the normal extraction line (NE). Its peak can be found near the middle of the extraction, but the distance to NE does not exceed 12% (Fig. 3), which is a very good result. The overall time of DR(1) resumption varies between 200% and 800% of NE time. This causes that DR(1) cannot be applied to ETL processes containing many loading nodes (like in our parallel data warehouse system).

5 Summary

The paper describes our extensions to the resumption of the interrupted extraction process. The Design-Resume algorithm [6] designed by Labio et al was briefly explained and its general idea was presented. We focused on the problem of resumption, when the extraction graph contains many loading nodes, and an example of such a case was described in Sect. 4.1. Mentioned therein was the ETL process loading data to a parallel spatial data warehouse collecting telemetric measurements [3]. As shown in the tests, running the resumption for each inserter separately may be very inefficient. That is why we proposed a modification to the algorithm. First we modified the Design procedure. During the resumption, those inserters which completed the loading prior to a failure, do not have to be run again. The proposed solution resulted in increasing the efficiency of the resumption (Fig. 2). The efficiency almost reached the highest possible level (Fig. 3). It is impossible to make the total processing time (extraction + resumption) shorter than the uninterrupted extraction process. If it was possible it would mean than it is better to interrupt extraction and run resumption process rather than running single extraction process.

References

1. Bruckner R., List B., Schiefer J.: Striving Towards Near Real-Time Data Integration for Data Warehouses. DaWaK 2002.
2. Galhardas H., Florescu D., Shasha D., Simon E.: Ajax: An Extensible Data Cleaning-Tool. In Proc. ACM SIGMOD Intl. Conf. On the Management of Data, Teksas (2000).
3. Gorawski M., Malczok R.: Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree. 5th Workshop on Spatial-Temporal DataBase Management (STDBM_VLDB'04), Toronto, Canada 2004.
4. Gorawski M., Piekarek M.: Development Environment ETL/JavaBeans. Studia Informatica, vol. 24, 4(56), 2003.
5. Gorawski M., Wocaw A.: Evaluation of the Efficiency of Design-Resume/JavaBeans Recovery Algorithm. Archives of Theoretical and Applied Informatics, 15(1), 2003.
6. Labio W., Wiener J., Garcia-Molina H., Gorelik V.: Efficient resumption of interrupted warehouse loads. SIGMOD Conference, 2000.
7. Labio W., Wiener J., Garcia-Molina H., Gorelik V.: Resumption algorithms. Technical report, Stanford University, 1998.
8. Oracle. WarehouseBuilder10gOracle. Available at: `http://otn.oracle.com/products/warehouse/index.html.`
9. Sagent Technologies Inc.: Personal correspondence with customers.
10. Vassiliadis P., Simitsis A., Skiadopoulos S.: Modeling ETL Activities asGraphs. In-Proc. 4th Intl. Workshop on Design and Management of Data Warehouses, Canada, (2002).
11. Vassiliadis P., Simitsis A., Georgantas P., Terrovitis M.: A Framework for the Design of ETL Scenarios. CAiSE 2003.

An Efficient Parallel Solution of Complex Toeplitz Linear Systems*,**

Pedro Alonso and Antonio M. Vidal

Universidad Politécnica de Valencia, cno. Vera s/n, 46022 Valencia, Spain
{palonso, avidal}@dsic.upv.es

Abstract. A parallel algorithm for solving complex non–hermitian Toeplitz linear systems is presented. The parallel algorithm exploits the special structure of Toeplitz matrices to obtain the solution in a quadratic asymptotically cost. Despite of the low cost of the sequential algorithm we have obtained an efficient parallel algorithm. Our parallel algorithm is based on transforming the Toeplitz matrix into Cauchy–like matrix leading to a reduction in the communication cost. We use a message–passing programming model and the experimental tests are carried out on a distributed memory computer architecture.

1 Introduction

Fast algorithms for solving Toeplitz linear systems are based on the *displacement rank* property of the Toeplitz matrices. There exist a great amount of the so called *fast* algorithms in the literature but all of them are based on a little set of classical ideas. In addition, there exists a lack of parallel versions of these *fast* algorithms for the solution of Toeplitz linear systems using general purpose distributed memory computer architectures like clusters of personal computers. This fact is mainly due to the difficulty in obtaining a good efficiency in parallel *fast* algorithms. There exist a great dependency among operations that causes a low computational cost with a large number of point–to–point and broadcast–type communications if a message passing model programming is used [1].

On the other hand, if the Toeplitz matrix is not strongly regular, *fast* algorithms can break down or can produce poor results regarding the accuracy of the solution. In [2] it can be found a parallel algorithm that uses a refinement technique in order to improve the precision of the solution.

The parallel algorithm presented in this paper transforms the Toeplitz matrix into an another type of structured matrix called Cauchy–like. This transformation allows to avoid all the point–to–point communications appearing in other parallel algorithms that work directly with the Toeplitz matrix. Furthermore, we use a blocking parallel algorithm that minimizes the execution time by balancing and overlapping the computational time and the communication cost.

* Supported by Spanish MCYT and FEDER under Grant TIC 2003-08238-C02-02.
** Funded by Secretaría de Estado de Universidades e Investigación del Ministerio de Educación y Cultura of Spain.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 486–493, 2006.

Although the non–hermitian represents the most general class of Toeplitz matrices, there exist several improvements that must be apply to the sequential algorithm before building a parallel one (see [3] i. e. for the real symmetric case).

We make use of the standard libraries LAPACK [4] and ScaLAPACK [5] in order to achieve a more easy and portable implementation. We use the BIHAR library for performing the DFT [6, 7] together with our own implementation of the DFT by using the *Chirp-z* factorization [8].

The next section includes a brief revision of the mathematical background. Afterward, the sequential (Sect. 3) and parallel (Sect. 4) algorithms are described. The experimental analysis is shown in Sect. 5.

2 Rank Displacement and Cauchy–Like Matrices

The concept of *rank displacement* [9] describes the special structure of *structured* matrices. The definition uses the *displacement equation* of a given $n \times n$ matrix. If the rank r of the *displacement equation* is considerably lower than n ($r \ll n$), it is said that the matrix is *structured*.

Given a Toeplitz matrix $T = (t_{ij}) = (t_{i-j})_{0 \le i,j < n} \in \mathbb{C}^{n \times n}$, its displacement equation can be defined in several ways. A useful form for our purposes is

$$\nabla_T = Z_1 \ T - T \ Z_{-1}^T = G \ H^* \ , \tag{1}$$

where $Z_1 = Z + e_1 e_n^T$ and $Z_{-1} = Z - e_1 e_n^T$, being Z the one position down shift matrix, e_i the ith column of the identity matrix and $G, H \in \mathbb{C}^{n \times 2}$ are the *generators*. The rank of ∇_T is 2 so matrices G and H has only 2 columns. An explicit form for G and H can be found in [10, 11].

It is said that a matrix $C = (c_{ij})_{1 \le i,j \le n}$ is a Cauchy matrix, if for some complex vectors $\omega = (\omega_i)_1^n$ and $\lambda = (\lambda_i)_1^n$, the matrix

$$\nabla_{\omega,\lambda} C = ((\omega_i - \lambda_j) c_{ij})_1^n \ , \ (\omega_i - \lambda_j) c_{ij} = 1 \ ,$$

has very low rank with respect to n. If $(\omega_i - \lambda_j) c_{ij} \neq 1$, matrix $\nabla_{\omega,\lambda} C$ is said to be a Cauchy–like matrix. Cauchy–like matrices can also be defined as the unique solution of the displacement equation

$$\nabla_{\Omega,\Lambda} C = \Omega \ C - C \ \Lambda^* = \hat{G} \ \hat{H}^* \ , \tag{2}$$

being $\Omega = \mathrm{diag}(\omega_1, \ldots, \omega_n)$, $\Lambda^* = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, and

$$\hat{G}^* = \begin{pmatrix} g_1 \ \ldots \ g_n \end{pmatrix}, \hat{H}^* = \begin{pmatrix} h_1 \ \ldots \ h_n \end{pmatrix}.$$

If $\omega_i \neq \lambda_j$ for $1 \le i, j \le n$, Cauchy–like matrices have the following form

$$C = (C_{i,j})_{1 \le i,j \le n} = \left(\frac{g_i^* \ h_j}{\omega_i - \lambda_j} \right)_{1 \le i,j \le n} \ . \tag{3}$$

Both Toeplitz and Cauchy–like matrices as defined in (1) and (2), respectively, are structured matrices. There exists a direct relation between both classes of

matrices, because the circulant matrices Z_{-1} and Z_1 (1) can be diagonalized by using the Discrete Fourier Transform (DFT).

If F is the normalized DFT matrix of size n defined as $F = \frac{1}{\sqrt{n}}\left(e^{-\frac{2\pi i}{n}kj}\right)$, for $j,k = 0,\ldots,n-1$, where $i = \sqrt{-1}$ and $FF^* = F^*F = I$, being I the identity matrix, then $F\ Z_1\ F^* = \Omega$ and $F\ Z_{-1}\ F^* = \Lambda^*$, thus, applying the transformation $F(.)F^*$ to (1) can be obtained the displacement (2).

The solution of a Toeplitz linear system

$$Tx = b\ , \tag{4}$$

with $T \in \mathbb{C}^{n\times n}$ and $b, x \in \mathbb{C}^n$, can be approached by using F to transform the Toeplitz system into a Cauchy–like system

$$(FTF^*)(Fx) = (Fb) \rightarrow C\hat{x} = \hat{b}\ . \tag{5}$$

The Cauchy–like linear system (5) is solved by performing a LU factorization of C, $C = LU^*$, where L and U are lower triangular matrices. Solving the following two triangular linear systems

$$Ly = \hat{b}\ ,\ U^*\hat{x} = y\ , \tag{6}$$

the solution $\hat{x}$ of (5) is obtained. Finally, $x \leftarrow F^*\hat{x}$ is the solution of (4).

3 Triangular Factorization of Cauchy–Like Matrices

Given a Cauchy–like matrix C and the displacement equation defined in (2), Gohberg, Kailath and Olshevsky [12] proposed an algorithm (`GKO`) to factorize non-hermitian Cauchy–like matrices. The algorithm uses (3) in order to obtain certain entries of the Cauchy–like matrix or its Schur complements. For non–hermitian complex Toeplitz matrices and using the displacement matrices Z_1 and Z_{-1} (1), all the elements of C can be computed using (3).

Given the following partitions of the matrices C, Ω y Λ^*,

$$C = \begin{pmatrix} d & u \\ l & C_1 \end{pmatrix}, \Omega = \begin{pmatrix} \omega_1 & 0 \\ 0 & \Omega_1 \end{pmatrix} \text{ and } \Lambda^* = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \Lambda_1^* \end{pmatrix}\ ,$$

where $\begin{pmatrix} d^* & l^* \end{pmatrix}^*$ and $\begin{pmatrix} d & u \end{pmatrix}$ are one dimension vectors, and being

$$X = \begin{pmatrix} 1 & 0 \\ l/d & I_{n-1} \end{pmatrix} \text{ and } Y = \begin{pmatrix} 1 & u/d \\ 0 & I_{n-1} \end{pmatrix}\ ,$$

if the transformation $X^{-1}(.)Y^{-1}$ is applied to (2), we have

$$(X^{-1}\ \Omega\ X)(X^{-1}\ C\ Y^{-1}) - (X^{-1}\ C\ Y^{-1})(Y\ \Lambda^*\ Y^{-1}) =$$

$$\begin{pmatrix} \omega_1 & 0 \\ \Omega_1\ l/d - l & \omega_1/d\ \Omega_1 \end{pmatrix}\begin{pmatrix} d & 0 \\ 0 & C_{cs} \end{pmatrix} - \begin{pmatrix} d & 0 \\ 0 & C_{cs} \end{pmatrix}\begin{pmatrix} \lambda_1\ u & \Lambda_1^*/d - \lambda_1\ u/d \\ 0 & \Lambda_1^* \end{pmatrix} =$$

$$(X^{-1}\ \hat{G})\ (\hat{H}^*\ Y^{-1})\ .$$

From the element $(2,2)$ of the previous equation it is obtained

$$\Omega_1 \;\; C_{cs} - C_{cs} \;\; \Lambda_1^* = \hat{G}_1 \;\; \hat{H}_1^* \;\; . \tag{7}$$

Matrix C_{cs} is the Schur complement of C with respect to d and, therefore, (7) is the displacement representation of the Schur complement of C with respect to its first element $C_{1,1}$. Matrices $\hat{G}_1 = X^{-1}\hat{G}$ and $\hat{H}_1^* = \hat{H}^* Y^{-1}$ are the *generators* of C_{cs}. Here, the property that the Schur complements of a structured matrix are structured too, is used.

The computation of the first row and column of C by means of (3) and the computation of $\hat{G}_1$ and $\hat{H}_1^*$ defines the first step of the algorithm. Repeating this process n iterations on the successive arising Schur complements, the LU factorization of C is obtained. This algorithm is described in Alg. 1.

Algorithm 1 (LU factorization of a Cauchy–like matrix): *Given $\hat{G}$,$\hat{H}$, Ω and Λ^* of the displacement (2), this algorithm returns the lower unit triangular factor L and the lower triangular factor U such that $C = LU^*$.*

for $j = 1, \ldots, n$
  $d = \frac{\hat{G}_{j,:} \;\; \hat{H}^*_{:,j}}{\omega_j - \lambda_j}$
  $U_{j,j} = d$
  for $i = j+1, \ldots, n$
    $L_{i,j} = \frac{\hat{G}_{i,:} \;\; \hat{H}^*_{:,j}}{d \;\; (\omega_i - \lambda_j)}$
    $U_{i,j} = \frac{\hat{H}_{:,i} \;\; \hat{G}^*_{j,:}}{\omega_j - \lambda_i}$
    for $k = 1, \ldots, r$
      $G_{i,k} = G_{i,k} - G_{j,k} L_{i,j}$
      $H_{i,k} = H_{i,k} - H_{j,k} U_{i,j}/d$
    end for
  end for
end for

Algorithm 1 can be apply r, $r < n$, iterations. In this case, the algorithm also returns the generators updated.

4 The Parallel Algorithm

The central part of the parallel algorithm is the LU factorization of C using the procedure described in Sect. 3. For the parallel LU factorization of C, the generators are distributed between the processors using the ScaLAPACK model. Under this model a *logical* unidimensional mesh with $p \times 1$ processors is used for distributing $\hat{G}$ and $\hat{H}$ (2) cyclically by row blocks of size η as follows.

Let the following partition of the generators be

$$\hat{G} = \begin{pmatrix} \hat{G}_0 \\ \hat{G}_1 \\ \vdots \\ \hat{G}_{n/\eta - 1} \end{pmatrix} \text{ , and } \hat{H} = \begin{pmatrix} \hat{H}_0 \\ \hat{H}_1 \\ \vdots \\ \hat{H}_{n/\eta - 1} \end{pmatrix} \text{ , } \tag{8}$$

where η is the number of rows of each block. The blocks of $\hat{G}$ and $\hat{H}$ are cyclically distributed among the p processors such that $\hat{G}_b$ and $\hat{H}_b$, $b = 0, \ldots, n/\eta - 1$, completely belong to processor $P_{b \bmod p \times 1}$. The ith row of $\hat{G}$ and $\hat{H}$ belong to blocks $\hat{G}_{i/\eta}$ and $\hat{H}_{i/\eta}$, respectively. Matrices Ω and Λ^* are distributed in the same way. Algorithm 2 is the parallel version of Alg. 1.

Algorithm 2 (Parallel LU factorization of a Cauchy–like matrix): *Given $\hat{G}$, $\hat{H}$, Ω and Λ^* (2) distributed as shown in (8) for a given $\eta \geq 1$, this algorithm returns the unit lower triangular factor L and the lower triangular factor U, such that $C = LU^*$. Factors L and U are returned distributed as the generators (8). On each processor P_k, for $k = 0, \ldots, p-1$:*

for $b = 0, \ldots, n/\eta - 1$
  Let C_{cs}^b be the Schur complement of C with respect to the leading submatrix of order $b\eta$ partitioned as:

$$C_{cs}^b = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \ , \ C_{11} \in \mathbb{C}^{\eta \times \eta} \ .$$

  if *$\hat{G}_b \in P_k$ (and $\hat{H}_b \in P_k$)*
    1. *Compute factors L_b and U_b using Alg. 1 so $C_{11} = L_b U_b^*$.*
    2. *Broadcast $\hat{G}_b$ and $\hat{H}_b$, updated in step 1, and Ω_b and Λ_b^*.*
  else
    3. *Receive $\hat{G}_b$, $\hat{H}_b$, Ω_b and Λ_b^*.*
  end if
  for $i = b+1, \ldots, n/\eta - 1$
    if *($\hat{G}_i \in P_k$), update the blocks $\hat{G}_i$, $\hat{H}_i$, L_{ib} and U_{ib},* **end if**
  end for
end for

Updating blocks $\hat{G}_i$, $\hat{H}_i$, $b < i < n/\eta$, of the generators in the above algorithm can be easily obtained from Alg. 1. Applying only η ($\eta < n$) iterations of the loop with index j in Alg. 1, the following factorization is obtained

$$C = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} \begin{pmatrix} U_1^* & U_2^* \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & C_{cs}^1 \end{pmatrix} \ ,$$

and the generators for the displacement of C_{cs}^1 are obtained too. Applying a few modifications to Alg. 1 it is easy to obtain an algorithm to compute L_2, U_2^* and the generators for the displacement of C_{cs}^1, once L_1 y U_1^* have been computed. Each iteration of this algorithm is executed locally on each processor using the parameters received from the processor P_k.

Algorithm 2 performs one broadcast in each iteration of its main loop. The total number of broadcasts is n/η and each message contains 6η elements so the size and the number of messages are both a function of η. Different values of η changes the overlapping between communications and computations and also the weight of both factors in the total cost of the algorithm. The optimum value of η depends on the machine and is experimentally tuned.

The completely parallel algorithm is summarized in Alg. 3.

Algorithm 3 (Parallel solution of Toeplitz linear system): *Given a non–hermitian complex Toeplitz matrix* $T \in \mathbb{C}^{n \times n}$ *and a right hand side vector* $b \in \mathbb{C}^n$*, this algorithm returns the solution vector* x *of the linear system* $Tx = b$*. On each processor* P_k*, for* $k = 0, \ldots, p-1$*:*

1. *Every processor computes matrices* $\hat{G}$*,* $\hat{H}$*,* Ω*,* Λ^* *and the vector* $\hat{b}$*. The elements are distributed as in (8).*
2. *Apply Alg. 2 to obtain the triangular factors* L *and* U*.*
3. *Solve the triangular systems (6) in parallel by using PBLAS routines.* P_0 *gathers* $\hat{x}$ *from the rest of the processors and computes* $x = F^* \hat{x}$*.*

5 Experimental Results

The experimental results have been obtained in a cluster of 10 nodes, each of one is a two–processor board with 1 Gb. of RAM with two Intel Xeon at 2 GHz. The interconnection network is a SCI with a 2D torus topology. In the experiments, each MPI process is mapped onto one processor of each node.

The first test consists of tuning the block size η used in the partition (8). Figure 1 shows that there exists a range of values for η that can be used to get the best performance. We have chosen a fixed size of $\eta = 20$ for our machine. A more detailed study with other problem sizes shows that this is the best choice.

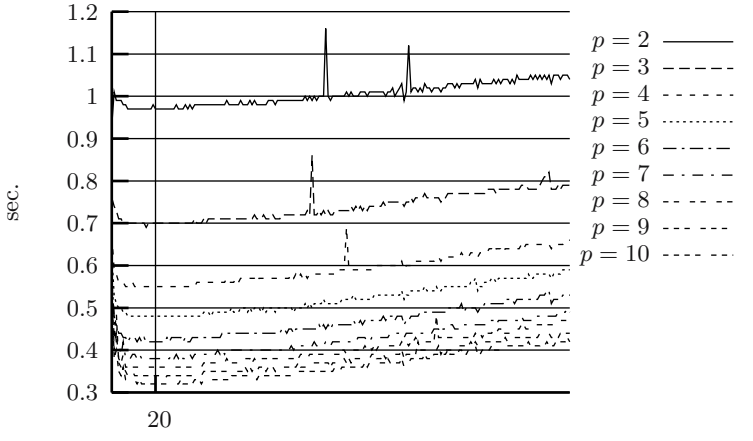


Fig. 1. Time versus different block sizes for $n = 4000$ and different processors

Another improvement carrying out in the algorithm deals with the use of the *Chirp-z* factorization for computing of the DFT. The time for transforming G and H (1) to $\hat{G}$ and $\hat{H}$ (2) involves two DFT's. The BIHAR library routines used highly depends on the prime decomposition of the problem size so if the problem size can be factorised in a product of low primes, the transformation is very fast; but if not, the transformation spends a great amount of time regarding the total

Table 1. Prime decomposition of different problem sizes

n	1999	2999	3999	4999	5999	6999	7999	8999	9999
p. d.	1999	2999	$3 \cdot 31 \cdot 43$	4999	$7 \cdot 857$	$3 \cdot 2333$	$19 \cdot 421$	8999	$3^2 \cdot 11 \cdot 101$
bihar	0.20	0.45	0.02	1.26	0.16	0.55	0.10	4.13	0.05
Chirp-z	0.01	0.02	0.03	0.06	0.06	0.07	0.07	0.15	0.15

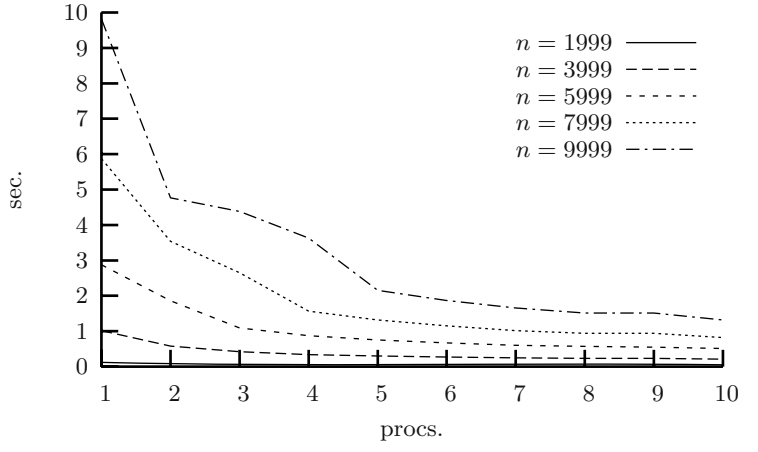


Fig. 2. Time for different size problems and different number of processors

Table 2. Efficiency of the parallel algorithm

n/p	2	3	4	5	6	7	8	9	10
1999	85%	77%	68%	61%	59%	51%	41%	37%	42%
2999	88%	81%	75%	68%	63%	58%	55%	49%	49%
3999	92%	86%	81%	75%	70%	68%	63%	56%	57%
4999	77%	88%	83%	77%	72%	69%	63%	58%	56%
5999	80%	92%	88%	83%	78%	75%	72%	64%	64%
6999	83%	74%	93%	89%	85%	82%	78%	69%	72%
7999	103%	76%	97%	93%	90%	86%	83%	75%	77%
8999	103%	75%	68%	91%	88%	85%	81%	72%	74%
9999	104%	100%	98%	95%	91%	88%	85%	76%	79%

execution time. Table 1 shows different problem sizes, its prime decomposition (p. d.) and the execution time in seconds without (bihar) and with the use of the *Chirp-z* factorization. If the prime factors are large, with the *Chirp-z* factorization we obtain a considerable lower time than only using BIHAR routines.

Figure 2 shows a large reduction in time achieved with the increment in the number of processors. This is specially significant because this result means the parallel algorithm can be useful in applications with real time constraints.

In addition, the results regarding the efficiency are quite good even up with 10 processors, taking into account that the asymptotically computational cost of the sequential algorithm is of $O(n^2)$ operations (see Table 2).

6 Conclusions

In this paper we have presented a parallel algorithm for solving Toeplitz linear systems of equations for non–hermitian complex matrices. The parallel algorithm exploits the rank displacement property the structured matrices. The transformation of a Toeplitz matrix into a Cauchy–like matrix lets to derive a parallel algorithm with a low communication cost so it can be obtained a large reduction in time and even a quite good efficiency in some cases. This fact represents an important improvement regarding other sequential algorithms traditionally used, like Levinson– or Schur–type algorithms, whose parallelization does not offer good results when run on distributed memory architectures.

References

1. Alonso, P., Badía, J.M., Vidal, A.M.: Parallel algorithms for the solution of toeplitz systems of linear equations. LNCS **3019** (2004) 969–976
2. Alonso, P., Badía, J.M., Vidal, A.M.: An efficient and stable parallel solution for non–symmetric Toeplitz linear systems. LNCS (to appear in 2005)
3. Alonso, P., Vidal, A.M.: The symmetric–Toeplitz linear system problem in parallel. LNCS (to appear in 2005)
4. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide. Second edn. SIAM, Philadelphia (1995)
5. Blackford, L., et al.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
6. Swarztrauber, P.: Vectorizing the FFT's. Academic Press, New York (1982)
7. Swarztrauber, P.: FFT algorithms for vector computers. Parallel Computing **1** (1984) 45–63
8. Loan, C.V.: Computational Frameworks for the Fast Fourier Transform. SIAM Press, Philadelphia (1992)
9. Kailath, T., Sayed, A.H.: Displacement structure: Theory and applications. SIAM Review **37** (1995) 297–386
10. Gallivan, K.A., Thirumalai, S., Dooren, P.V., Vermaut, V.: High performance algorithms for Toeplitz and block Toeplitz matrices. Linear Algebra and its Applications **241/243** (1996) 343–388 Proceedings of the Fourth Conference of the International Linear Algebra Society (Rotterdam, 1994).
11. Alonso, P., Badía, J.M., Vidal, A.M.: Parallel algorithms for the solution of Toeplitz systems of linear equations by means of the Cauchy–like method. Technical Report DSIC-II/26/2002, Departamento de Sistema Informáticos y Computación de la Universidad Politécnica de Valencia (2002)
12. Gohberg, I., Kailath, T., Olshevsky, V.: Fast Gaussian elimination with partial pivoting for matrices with displacement structure. Mathematics of Computation **64** (1995) 1557–1576

Monitoring the Block Conjugate Gradient Convergence Within the Inexact Inverse Subspace Iteration

Carlos Balsa[1], Michel Daydé[2], Ronan Guivarc'h[2],
José Laginha Palma[1], and Daniel Ruiz[2]

[1] FEUP, Porto, Portugal
{cbalsa, jpalma}@fe.up.pt
[2] ENSEEIHT–IRIT, Toulouse, France
{dayde, guivarch, ruiz}@enseeiht.fr

Abstract. We propose an algorithm called BlockCGSI to compute some partial spectral information related to the ill-conditioned part of a given coefficient matrix. This information can then be used to improve the solution of consecutive linear systems with the same coefficient matrix and changing right-hand sides.

The BlockCGSI algorithm combines the block Conjugate Gradient with the inverse Subspace Iteration. We indicate how to reduce the total amount of computational work by controlling appropriately the accuracy when solving the linear systems at each inverse iteration. We also improve the global convergence of the algorithm by means of polynomial filters.

1 Introduction

Partial spectral information associated with the smallest eigenvalues can be used to improve the solution of successive linear systems of equations. This is specially the case in the simulation of time dependent partial differential equations, where at each global iteration there are several systems with the same spectral properties to be solved. The idea is to perform some partial spectral decomposition in the first global iteration, and to exploit this information to improve the convergence in the iterative solution of the following linear systems. In previous work [1], we have experimented two techniques to improve the convergence of Conjugate Gradient (CG) algorithm, namely the deflation of the initial residual, and the Spectral Low Rank Update (SLRU) preconditioner [2]. We observed that the SLRU preconditioner, in contrast with the deflated starting residual, is efficient even when the precomputed spectral information is not very accurate.

In this work, we just focus on the computation of a *near*-invariant subspace associated with the smallest eigenvalues in the iteration matrix, and we propose an algorithm, called BlockCGSI, based on the inverse subspace iteration [3]. In this algorithm, the set of multiple solutions required for each inverse iteration is computed iteratively using a stabilized version of the block Conjugate Gradient algorithm (blockCG) [4, 5]. This combination, detailed in section 2, was proposed

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 494–504, 2006.

initially in [6], and used to deflate the initial residual in the consecutive runs of the CG algorithm.

In section 3, we analyze the convergence of the blockCG algorithm, and we propose a way to choose the threshold used to monitor the accuracy in the solution of the linear systems with respect to the global convergence of the inverse iteration outer loop. We devote section 4 to some numerical results illustrating the good behavior of our algorithm in general. Finally, in section 5, we conclude about the potential and limitations of the proposed technique.

2 BlockCG Coupled with Subspace Iteration (BlockCGSI Algorithm)

In this section, we present and detail partly the BlockCGSI algorithm used to compute an M-orthonormal basis W of a *near*-invariant subspace associated with the smallest eigenvalues in the preconditioned matrix $M^{-1}A$. If this basis incorporates, for instance, all the eigenvalues of $M^{-1}A$ in the range $[0, \mu]$, we can expect, when using it later as a second level of preconditioning, that the condition number of the coefficient matrix will be reduced to about $\kappa = \lambda_{\max}/\mu$ (where $\lambda_{\max}$ is the largest eigenvalue in $M^{-1}A$). In Algorithm 1, $\lambda_{\max}$ and μ are considered as input parameters (just a rough upper bound on $\lambda_{\max}$ is sufficient in general).

An other input concerns the choice of the block size s that defines the dimension of the working subspace at each inverse iteration. It also gives the number of right-hand sides and solutions vectors of the multiple linear systems solved by the blockCG algorithm, and consequently the amount of memory required as working space.

As a starting point, the algorithm requires the generation of an M-orthonormal basis of size s. The closer are these vectors to the targeted *near*-invariant subspace, the faster will be the convergence of the inverse iteration. The scope of steps 1 to 4 in Algorithm 1, is to generate an initial M-orthonormal set $V^{(0)}$ of s vectors with eigencomponents corresponding to eigenvalues in the range $[\mu_f, \lambda_{\max}]$ below some predetermined value $\xi \ll 1$ (denoted as the "filtering level"). This filtering technique is based on Chebyshev polynomials (step 3) and details about it can be found in [7]. The idea behind the use of these Chebyshev filters at the starting point is to put the inverse subspace iteration in the situation of working in the orthogonal complement of a large number of eigenvectors, e.g. all those associated with the eigenvalues in the range $[\mu_f, \lambda_{\max}]$. We can also expect that the resulting filtered right-hand sides will present more favorable spectral properties that can improve the convergence behavior of the blockCG. Obviously, there is some compromise to achieve, in the sense that very small values of μ_f and ξ will minimize the number of inverse and blockCG iterations but will also increase the computational efforts in the Chebyshev initial filtering step.

The essence of the inverse subspace iteration is the QR iteration. It consists in multiplying a set of vectors by $A^{-1}M$ and M-ortonormalizing it in turn.

ALGORITHM 1. BLOCKCGSI ALGORITHM

Inputs: $A, M = R^T R \in \mathbb{R}^{n \times n}, \mu, \lambda_{max} \in \mathbb{R}, s \in \mathbb{N}$
Output: a *near*-invariant subspace W associated with all eigenvalues in the range $]0, \mu]$

Begin

Generate the initial subspace (with filtering)

1. $Z^{(0)}$ =RANDOM(n, s)
2. $V^{(0)} = Z^{(0)}\Gamma$ such that ${V^{(0)}}^T V^{(0)} = I_{s \times s}$
3. $Q^{(0)}$=Chebyshev-Filter$(V^{(0)}, \xi, [\mu_f, \lambda_{\max}], A, R)$
4. $V^{(0)} = R^{-1}Q^{(0)}\Gamma$ such that ${V^{(0)}}^T M V^{(0)} = I_{s \times s}$
5. $W^{(0)}$ = empty
6. **For** $k = 1, ...,$ until convergence **Do:**

QR iteration

i. Solve $M^{-1}AZ^{(k)} = V^{(k-1)}$ with blockCG
ii. $P_k = Z^{(k)} - W^{(k-1)}{W^{(k-1)}}^T M Z^{(k)}$
iii. $Q^{(k)}\Gamma_k = P^{(k)}$ such that ${Q^{(k)}}^T M Q^{(k)} = I_{s \times s}$
iv. $Q^{(k)} = [W^{(k-1)}\ Q^{(k)}]$

Ritz acceleration

v. $\beta_k = {Q^{(k)}}^T A Q^{(k)}$
vi. Diagonalize $\beta_k = U_k \Delta_k U_k^T$
where $U_k^T = U_k^{-1}$
and Δ_k =Diag$(\delta_1, ..., \delta_{p+s})$ (Ritz Values)
vii. $V^{(k)} = Q^{(k)}U_k$ (Ritz Vectors)

Update the computational window

viii. $W^{(k)}$ = converged columns of $V^{(k)}$
ix. $V^{(k)}$ = non-converged columns of $V^{(k)}$
x. (n, p) = size$(W^{(k)})$
xi. Incorporate new vectors in $(V^{(k)})$

7. **EndDo**

End

If $W^{(k-1)}$ (initially empty) contains the set of vectors that have already converged at step $k-1$, the current subspace $Q^{(k)}$ should converge gradually to a *near*-invariant subspace that is M-orthogonal to $W^{(k-1)}$. In step i, the multiplication by $A^{-1}M$ is performed implicitly through the iterative solution of the system $M^{-1}AZ^{(k)} = V^{(k)}$ via the blockCG. In order to reduce the computational costs, this system is solved with a certain accuracy determined by the residual threshold ε. The appropriate choice of ε is detailed in section 3.2.

In step ii, the approximate solution vectors $Z^{(k)}$ are then projected onto the orthogonal complement of the converged vectors $W^{(k)}$, in order to remove the influence of eigencomponents associated with the converged eigenvalues. The set of projected vectors $P^{(k)}$ is then M-orthonormalized (step iii), and gathered together with $W^{(k)}$.

To improve the rate of convergence of the inverse subspace iteration, the QR iteration is followed by the Ritz acceleration (steps v to vii), as suggested by [3]. The spectral information contained in $Q^{(k)}$ is thus redistributed in the column vectors of $V^{(k)}$, that will contain each better approximations of individual eigenvectors. Steps v, vi, and vii, give the Ritz values, $diag(\Delta) = \delta_1, ..., \delta_{p+s}$, ranged in increasing order, and the associated Ritz vectors, $[v_1, v_2, ..., v_p, ..., v_{p+s}]$, where p is the dimension of $W^{(k-1)}$ and s is the current block size.

The end of the BlockCGSI algorithm consists in testing the convergence and updating the computational window. In step viii, all the Ritz vectors that are considered as *near*-invariant (with respect to the given accuracy) are assigned to $W^{(k)}$. More precise details about the monitoring of the convergence are given in section 3.1. Step xi consists in incorporating new vectors in the current set of vectors $V^{(k)}$. The operation that consists in introducing a set of ℓ new vectors, after some of the Ritz vectors have converged, is detailed in Algorithm 2. We denote this algorithmic issue in the BlockCGSI algorithm as *"sliding window"*. Its purpose is to enable the approximation of a number of eigenvectors greater than the block size s. Basically, we generate randomly the new vectors and filter them, as in the starting steps of the BlockCGSI algorithm. Then, these vectors are projected in the M-orthogonal complement of the converged ones, in order to remove the corresponding eigencomponents. Note that we can also opt to reduce or enlarge the block size s at this stage, when setting the value of ℓ (i.e. the number of newly incorporated vectors).

Algorithm 2. Incorporate New Vectors
a) P =RANDOM(n, ℓ)
b) $P = Q\Gamma$ such that $Q^TQ = I_{\ell\times\ell}$
c) P=Chebyshev-Filter$(Q, \xi, [\mu_f, \lambda_{\max}], A, R)$
d) $Q = R^{-1}P\Gamma$ such that $Q^TMQ = I_{\ell\times\ell}$
e) $P = Q - W^{(k)}{W^{(k)}}^TMQ$
f) $V^{(k)} = [V^{(k)}P]$

3 Convergence Analysis

The BlockCGSI algorithm involves two iterative loops: the first, that we also denote as the outer iteration, corresponds to the inverse iteration (at step 6 in the in Algorithm 1), and the second loop, or inner iteration, in the call to the blockCG algorithm (at step i in Algorithm 1) for the iterative solution of the linear system with multiple right-hand sides, $M^{-1}AZ = V$. These two iterations require some specific stopping criterion to monitor the global convergence of the

algorithm. Sections 3.1 and 3.2 are devoted to the analyze of the convergence properties at the two iteration levels. In section 3.2, we also propose a way to link the monitoring of the convergence in the inner loop with the measure of the convergence in the outer loop.

3.1 Inverse Subspace Iteration (Outer Loop)

At each inverse iteration k in Algorithm 1, the blockCG algorithm solves the s linear systems $M^{-1}Az_j^{(k)} = v_j^{(k-1)}$, $j = 1, ..., s$, where the matrix A is preconditioned with a symmetric and positive definite preconditioner, $M = R^T R$. The symmetrized system can be written as usual as

$$R^{-T}AR^{-1}Rz_j^{(k)} = Rv_j^{(k-1)} \Longleftrightarrow \tilde{A}\tilde{z}_j^{(k)} = \tilde{v}_j^{(k-1)}, \qquad j = 1, ..., s. \quad (1)$$

For simplicity, we will omit to repeat that j varies from 1 to s. We will consider that the subscript j refers to the position of the corresponding eigenvalue in the current working set. The superscript (k) denotes the inverse iteration number, and the tilde refers to the symmetrized system (1).

The outer iteration produces a sequence of Ritz vectors $\tilde{v}_j^{(1)}, \tilde{v}_j^{(2)}, ..., \tilde{v}_j^{(k-1)}$, that converge to the eigenvector $\tilde{u}_j = Ru_j$ corresponding to the eigenvalue λ_j of both matrices $\tilde{A}$ and A. At step k, the vectors $\tilde{v}_j^{(k)}$ are orthonormal, since the vectors $v_j^{(k)}$ (columns of matrix $V^{(k)}$) are M-orthonormal. The corresponding Ritz values (diagonal elements of matrix $\Delta^{(k)}$) are given by $\delta_j{}^{(k)} = {v_j}^{(k)^T} A v_j^{(k)} = \tilde{v}_j^{(k)T} \tilde{A} \tilde{v}_j^{(k)}$.

Provided the Ritz acceleration is incorporated in the algorithm, the convergence rate of the inverse subspace iteration, given by [8], is proportional to ratio $\lambda_j/\lambda_{p+s+1}$, where λ_{p+s+1} is the eigenvalue immediately following the current set of $(p + s)$ approximated eigenvalues. This property shows that, if the inner iteration is accurately enough and if λ_j is well separated from λ_{p+s+1}, we can have a good estimation of the eigenvalue λ_j in a few number of inverse iterations. In some cases, it can even be useful to increase the block size s just to benefit from a better gap in the relation above (*"Guard Vectors"* effect).

Finally, we introduce the error bound on each approximate eigenpair, given by [3]:

$$|\lambda_j - \delta_j^{(k)}| \leq \frac{||Av_j^{(k)} - \delta_j^{(k)} M v_j^{(k)}||_{M^{-1}}}{||Mv_j^{(k)}||_{M^{-1}}} = ||M^{-1}Av_j^{(k)} - \delta_j^{(k)} v_j^{(k)}||_M, \quad (2)$$

assuming that λ_j is actually the closest eigenvalue to $\delta_j^{(k)}$. Dividing (2) by $\delta_j^{(k)}$ instead of λ_j, we get an estimate of the relative residual upper bound and an estimate of the number of correct digits in $\delta_j^{(k)}$, which can be used to decide if a Ritz vector $v_j^{(k)}$ has converged or not.

3.2 BlockCG Iteration (Inner Loop)

The block Conjugate Gradient (blockCG) algorithm solves simultaneously the s linear systems from equation (1). For each system, $j = 1, .., s$, it produces a sequence of vectors $\tilde{z}_j^{[i]}$ (where the superscript $[i]$ stands for the blockCG iteration number) giving, after convergence, the approximate solution $\tilde{z}_j^{(k)}$ used in the k-th inverse iteration.

The residual vector associated with each iterate $\tilde{z}_j^{[i]}$ is $\tilde{r}_j^{[i]} = \tilde{v}_j^{(k-1)} - \tilde{A}\tilde{z}_j^{[i]}$. We also introduce another vector, which we will use to measure the proximity of the current iterate $\tilde{z}_j^{[i]}$ from the corresponding eigenvector $\tilde{u}_j$,

$$\tilde{S}_j^{[i]} = \tilde{A}\tilde{z}_j^{[i]} - \tilde{\delta}_j^{[i]}\tilde{z}_j^{[i]} = \tilde{v}_j^{(k-1)} - \tilde{\delta}_j^{[i]}\tilde{z}_j^{[i]} - \tilde{r}_j^{[i]},$$

where $\delta_j^{[i]} = \tilde{\delta}_j^{[i]} = \tilde{z}_j^{[i]T}\tilde{A}\tilde{z}_j^{[i]}/\tilde{z}_j^{[i]T}\tilde{z}_j^{[i]}$ is the Rayleigh quotient corresponding to the current iterate $\tilde{z}_j^{[i]}$. After some inverse iterations, we can assume that λ_j is the closest eigenvalue in the spectrum of $M^{-1}A$ from $\delta_j^{[i]}$, and the error bound (2) applied on the symmetrized system, at each inner iteration $[i]$, yields

$$|\lambda_j - \delta_j^{[i]}| \leq \frac{||\tilde{S}_j^{[i]}||_2}{||\tilde{z}_j^{[i]}||_2} = \frac{||\tilde{v_j}^{(k-1)} - \delta_j^{[i]}\tilde{z}_j^{[i]} - \tilde{r}_j^{[i]}||_2}{||\tilde{z}_j^{[i]}||_2}. \tag{3}$$

Additionally, if we start the blockCG iteration with $\tilde{z}_j^{[0]} = 0$, at each iteration the current residual $\tilde{r}_j^{[i]}$ remains orthogonal to both $\tilde{v_j}^{(k-1)}$ (the right-hand side) and $\tilde{z}_j^{[i]}$ (linear combination of the current Krylov vectors). Thus, $\tilde{r}_j^{[i]T}(\tilde{v_j}^{(k-1)} - \delta_j^{[i]}\tilde{z}_j^{[i]}) = 0$, and we can write

$$||\tilde{v_j}^{(k-1)} - \delta_j^{[i]}\tilde{z}_j^{[i]} - \tilde{r}_j^{[i]}||_2^2 = ||\tilde{v_j}^{(k-1)} - \delta_j^{[i]}\tilde{z}_j^{[i]}||_2^2 + ||\tilde{r}_j^{[i]}||_2^2. \tag{4}$$

Finally, if we translate the previous properties to the non-symmetrized system, we get

$$\begin{aligned} |\lambda_j - \delta_j^{[i]}| &\leq \frac{||M^{-1}Az_j^{[i]} - \delta_j^{[i]}z_j^{[i]}||_M}{||z_j^{[i]}||_M} \\ &= \sqrt{\frac{||v_j^{(k-1)} - \delta_j^{[i]}z_j^{[i]}||_M^2 + ||r_j^{[i]}||_M^2}{||z_j^{[i]}||_M^2}} \stackrel{def}{=} \sqrt{{\phi_j^{[i]}}^2 + {\omega_j^{[i]}}^2}. \end{aligned} \tag{5}$$

From (5), we can see that the error bound associated with each Rayleigh quotient depends on the relative residual measure $\omega_j^{[i]} = ||r_j^{[i]}||_M/||z_j^{[i]}||_M$ and on $\phi_j^{[i]} = ||v_j^{(k-1)} - \delta_j^{[i]}z_j^{[i]}||_M/||z_j^{[i]}||_M$. Even if we expect that the backward error measure $\omega_j^{[i]}$ will decrease down to a level of small magnitude, the value of $\phi_j^{[i]}$ is more likely to stagnate on a higher level, depending on the proximity

of the right-hand side $v_j^{(k-1)}$ from the correspondent eigenvector u_j. Therefore, the bound in (5) can be dominated by the value of $\phi_j^{[i]}$, and little improvement on the global convergence of the algorithm can be expected by further iterations in the blockCG.

We now investigate the asymptotic behavior of $\phi_j^{[i]}$, assuming that $z_j^{[i]}$ actually converges to $z_j^* = A^{-1}Mv_j^{(k-1)}$. Let us first introduce the asymptotic limit of $\delta_j^{[i]}$, $\delta^* = \langle z_j^*, v_j^{(k-1)} \rangle_M / ||z_j^*||_M^2$, and the angle θ_j in the M-norm between z_j^* and $v_j^{(k-1)}$, whose cosine is given by

$$\cos(\theta_j) = \frac{\langle z_j^*, v_j^{(k-1)} \rangle_M}{||z_j^*||_M ||v_j^{(k-1)}||_M} = ||\delta^* z_j^*||_M. \tag{6}$$

Since $v_j^{(k-1)}$ is M-orthonormal, we can write $\sin(\theta_j) = ||v_j^{(k-1)} - \delta^* z_j^*||_M$, which is also the asymptotic limit of $||v_j^{(k-1)} - \delta_j^{[i]} z_j^{[i]}||_M$. Consequently, the asymptotic limit of the component $\phi_j^{[i]}$ in (5) is

$$\phi_j^{[i]} \xrightarrow[i \to \infty]{} \frac{\sin(\theta_j)}{||z_j^*||_M} = \delta^* \tan(\theta_j), \tag{7}$$

which depends only on the two vectors $v_j^{(k-1)}$ and $z_j^* = A^{-1}Mv_j^{(k-1)}$. It is also clear that, if $v_j^{(k-1)}$ is close to an eigenvector, the angle θ_j should be very small, as well as the corresponding asymptotic limit of $\phi_j^{[i]}$. With respect to the bound in (5), this allows more room for decreasing the backward error $\omega_j^{[i]}$ in the blockCG iteration. The strategy suggested by this analysis is to decrease the value of the stopping criterion in the blockCG (inner loop) along with the convergence of the inverse iteration (outer loop). This is in agreement with the suggestions made in [9]. This basic idea is further developed to propose a stopping criterion for the blockCG.

The stopping criterion for the blockCG defines the approximation degree of all the solution vectors $\tilde{z}_j^{(k)} \approx \tilde{A}^{-1}\tilde{v}_j^{(k-1)}$ or equivalently of $z_j^{(k)} \approx A^{-1}Mv_j^{(k-1)}$. We propose to monitor only the convergence of the approximate solution $z_{p+1}^{[i]}$, corresponding to the smallest nonconverged Ritz value $\delta_{p+1}^{(k-1)}$ in the previous inverse iteration. The reason is that, in general, this system needs more efforts to be solved accurately. We use the relative residual measure

$$\omega_1^{[i]} = \frac{||v_1^{(k-1)} - M^{-1}Az_1^{[i]}||_M}{||z_1^{[i]}||_M}, \tag{8}$$

readily available in the blockCG iteration (see [5]). Notice also that $\omega_1^{[i]}$ is very close to the usual Rigal-Gaches [10] backward error measure.

In the outer loop, we consider the upper bound in (2) to monitor the accuracy of the approximated eigenvalues. A Ritz value $\delta_j^{(k)}$ has converged when it has at

least t correct digits, e.g. when $|\lambda_j - \delta_j^{(k)}|/\delta_j^{(k)} \leq 10^{-t}$. In order to satisfy this inequality, the stopping criterion in the blockCG is set as

$$\omega_1^{[i]} \leq \varepsilon, \qquad \text{with} \qquad \varepsilon = 10^{-t}\delta_1^{(k-1)}, \tag{9}$$

where $\delta_1^{(k-1)}$ is the smallest of the current set of nonconverged Ritz values. This stopping criterion is based on the upper bound (5), assuming that $|\lambda_1 - \delta_1^{[i]}|$ from the inner loop will be close to $|\lambda_1 - \delta_1^{(k)}|$ after the actual computation of the Ritz values in the outer loop.

Another assumption is that the value of $\phi_1^{[i]}$ in (5) is not dominant, and therefore that the value of $w_1^{[i]}$ governs the absolute error measure $|\lambda_1 - \delta_1^{[i]}|$ in the inner iteration. This is surely the case when the Ritz vector $v_1^{(k-1)}$ is close to an eigenvector because, in this case, the value of $\sin(\theta_1)$ should be small. However,this situation can be reversed when $\phi_1^{[i]}$ has reached its stagnation level defined in (7), in which case the blockCG iteration should be stopped and the next inverse iteration be launched. The risk of having $\omega_1^{[i]}$ much smaller than $\phi_1^{[i]}$ in the course of the blockCG iterations can anyway be limited with a relative precision 10^{-t} not too small ($t = 1$ or 2, for instance), which is in general enough for our purpose, e.g. building a *near*-invariant subspace for preconditioning the solution of consecutive linear systems with the same matrix. Another issue that helps in general to minimize this risk, is the use of $\delta_{p+1}^{(k-1)}$ (smallest Ritz value in the computational window) instead of the corresponding eigenvalue λ_{p+1} in (9). Indeed, when the Ritz vector is not close to be an eigenvector, the associated Ritz value is larger than the actual eigenvalue, all this resulting in the choice of a larger threshold ε in (9).

4 Numerical Experiments

In this section, we present some numerical results concerning the computation of the *near*-invariant basis W associated with the eigenvalues of $M^{-1}A$ in the range $]0, \mu[$. We have chosen, for illustration, a test matrix coming from the 2D heterogeneous diffusion equation discretized by finite elements in a L shape region, with size $n = 7969$ and non-zeros elements $nnz = 55131$. We also precondition the resulting linear system by means of the classical Incomplete Cholesky without fill-in ($IC(0)$). The spectrum is distributed from $\lambda_{\min} = 1.66e-08$ to $\lambda_{\max} = 1.55e+00$.

In figure 1, we show the convergence behavior of the values ω_1, ϕ_1 and the upper bound $\sqrt{\omega_1^2 + \phi_1^2}$ in (5) of $|\lambda_1 - \delta_1|$, as presented in section 3.2. The two plots in figure 1 illustrate the behavior of these three values in the blockCG run at the first inverse iteration ($k = 1$). The first plot corresponds to the case of non filtered starting vectors, and the second one to the case of starting vectors filtered with a level $\xi = 1e-2$ and a cut-off value $\mu_f = 5e-3$. We can observe the effect of the Chebyshev filtering of the starting vectors, which helps to make the value of ϕ_1 much smaller than what it can be with randomly generated

vectors. The consequence of that is that ω_1 becomes a good measure of the upper bound of $|\lambda_1 - \delta_1|$, even at the very beginning of the algorithm. Additionally, the filtering of the starting vectors changes the convergence behavior of the blockCG, because the filtered right-hand sides have more favorable spectral properties. It also enables to decrease substantially the asymptotic value of ϕ_1 in the first inverse iteration, which allows a larger range for the choice of the threshold ε in the blockCG.

In table 1, we present the total number of iterations performed at different levels in the BlockCGSI algorithm (see Algorithm 1) to compute a *near*-invariant subspace associated with all the eigenvalues in the range $]0, \mu[$. The requested relative precision in these eigenvalues has been set to one correct digit ($t = 1$), with respect to the convergence criterion for the outer loop given in section (3.1), and the stopping criterion for the blockCG set accordingly as in (9). The total number of inverse iterations is indicated by `InvIt` and the sum of all the iterations performed by the blockCG over all solves by `bCGIt`. The Chebyshev iterations count parameter `ChebIt` includes all the iterations spent in filtering the starting vector and the new vectors when the computational window is updated (see Algorithm 2). Finally we also include the total number of floating point operations performed by the BlockCGSI algorithm, in millions (`Mflops`). We have varied the filtering level from $\xi = 1e-6$ to $\xi = 1e-16$, including the case of no filtering. The block size haves been chosen to illustrate all the possible cases, e.g. when it is below, equal, or greater than the targeted number of eigenvectors (q). The two cut-off values of the filtering step μ_f correspond to the cases when it is greater or equal to the principal cut-off value μ in Algorithm 1.

The different results exposed in table 1 show that the algorithm manages to compute the targeted spectral information independently of the choice of the block size s. Of course it is optimal when s is correlated to the actual number

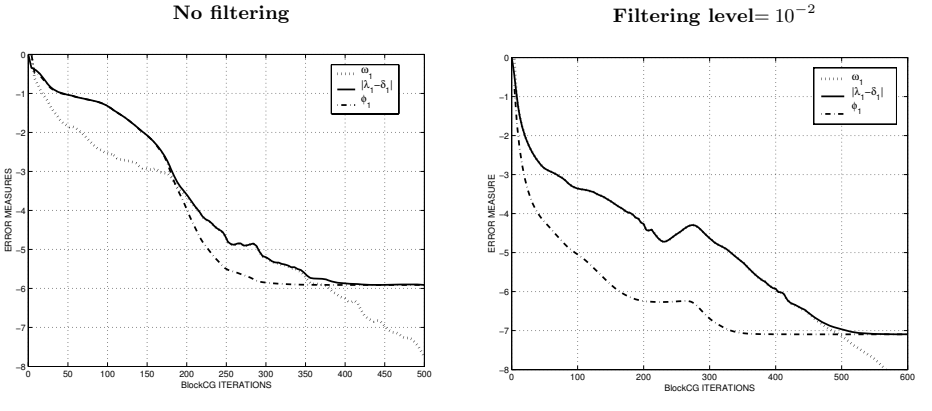


Fig. 1. Correlation between the residual measures ω_1, ϕ_1 and the upper bound on the spectral error $|\lambda_1 - \delta_1|$, in the blockCG with block size 4, and at the first inverse iteration

Table 1. Iteration and operation count as a function of the filtering level

	$\mu = 1.0e-2$ (3 eigenvalues)								$\mu = 3.0e-2$ (9 eigenvalues)							
Filter	$s = 3$, $\mu_f = 1.0e-1$				$s = 5$, $\mu_f = 1.0e-1$				$s = 5$, $\mu_f = \mu$				$s = 9$, $\mu_f = \mu$			
Level ξ	InvIt	bCGIt	ChebIt	Mflops	InvIt	bCGIt	ChebIt	Mflops	InvIt	bCGIt	ChebIt	Mflops	InvIt	bCGIt	ChebIt	Mflops
-	6	96	-	148	4	73	-	243	30	191	-	907	17	118	-	1310
$1e-6$	6	75	28	143	2	44	28	180	4	41	104	358	2	25	52	351
$1e-8$	6	63	37	134	2	35	37	165	3	20	138	352	2	11	69	274
$1e-10$	6	53	46	127	2	27	46	153	3	12	170	390	1	3	85	235
$1e-12$	3	35	55	99	2	19	55	140	3	6	204	434	1	3	102	275
$1e-14$	3	27	64	96	2	12	64	128	2	4	236	450	1	1	118	295
$1e-16$	2	19	75	88	1	10	73	130	2	4	270	556	1	1	135	335

of eigenvalues (q) in the range $]0, \mu[$. In this case, all the iterations count are minimized as well as the total number of operations. With larger block sizes s, the algorithm benefits from the *"guard vectors"* effect (see section 3.1), and the number of inverse iterations are reduced. A greater block size also improve the convergence of the block Conjugate Gradient. For these reasons, the increase of s does not necessarily imply increase of the total amount of work. When the block size is smaller than q the *"sliding window"* feature enables to obtain at any rate all the targeted vectors.

As we can observe in table 1, the filtering of the new vectors with Chebyshev polynomials improves very much the efficiency of the BlockCGSI algorithm. As the filtering level ξ decreases, the number of inverse iterations is reduced because the resulting filtered vectors are close to a *near*-invariant subspace, and the stagnation level of ϕ is very low (see also figure 1). This gives room for greater decrease of the error $|\lambda_i - \delta_i|$ at each inverse iteration. The number of blockCG iterations is also reduced, due to the better spectral properties of the right-hand. Obviously, decreasing the filtering level ξ also increases the number of Chebyshev iterations (`ChebIt`). In that respect, there is a compromise to reach in terms of total computational cost. The optimal value of ξ, that minimizes this computational work (`Mflops`), also depends on the other filtering parameter μ_f, and on the number of targeted eigenvalues q. We can observed that the closest is μ_f from μ, the higher is the optimal choice for ξ, and the smaller is q, the smaller is the optimal ξ.

5 Concluding Remarks

We have developed an algorithm that computes a *near*-invariant subspace, associated with the smallest eigenvalues in $M^{-1}A$, which combines the subspace inverse iteration and a stabilized version of the block Conjugate Gradient (blockCG) algorithm. The main focus in this work was the precise control of the accuracy when solving the system with multiple right-hand sides at each inverse iteration, and more precisely the good agreement of the stopping criterion used in the blockCG iteration with the measure of convergence in the inverse iteration itself. We have also investigated some particular techniques, like the

Chebyshev filtering of the starting vectors, and a form of dynamic adjustment of the dimension of the current subspace at each inverse iteration. The preliminary experiments indicate that Chebyshev filtering is useful to reduce the total amount of work through the reduction of both the inverse and blockCG iterations. The *"sliding window"* technique is helpful at any rate to make the algorithm more flexible and robust.

References

1. Balsa, C., Palma, J., Ruiz, D.: Partial spectral information from linear systems to speed-up numerical simulations in computational fluid dynamics. In Daydé, M., Dongarra, J., Hernandez, V., Palma, J., eds.: High Performance Computing for Computational Science, 6th Int. Meeting, VECPAR'04. LNCS 3402, Berlin, Springer-Verlag (2005) pp. 703–719
2. Carpentieri, B., Duff, I., Giraud., L.: A class of spectral two-level preconditioners. SIAM Journal on Scientific Computing **25** (2003) pp. 749–765
3. Parlett, B.N.: The Symmetric Eigenvalue Problem. SIAM, Philadelphia (1998)
4. O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra and its Applicattions (29) (1980) 293–322
5. Arioli, M., Duff, I., Ruiz, D., Sadkane, M.: Block Lanczos techniques for accelerating the block cimmino method. SIAM Journal on Scientific and Statistical Computing **16**(6) (1995) pp. 1478–1511
6. Arioli, M., Ruiz, D.: Block conjugate gradient with subspace iteration for solving linear systems. In: Iterative Methods in Linear Algebra, Second IMACS Symposium on Iterative Methods in Linear Algebra, Blagoevgrad, Bulgaria, S. Margenov and P. Vassilevski (eds.) (June, 1995) pp. 64–79
7. Giraud, L., Ruiz, D., Touhami, A.: A comparative study of iterative solvers exploiting spectral information for SPD systems. Technical Report RT/PA/04/40, CERFACS (2004) Also Technical Report TR/TLSE/04/03, ENSEEIHT-IRIT, Toulouse, France.
8. Golub, G.H., Loan, C.F.V.: Matrix Computation. The Johns Hopkins University Press, Baltimore and London (1983)
9. Golub, G.H., Ye, Q.: Inexact inverse iteration for generalized eigenvalue problems. BIT **40** (2000) pp. 671–684
10. Rigal, J.L., Gaches, J.: On the compatibility of a given solution with the data of a linear system. Journal of the ACM **14**(3) (July 1967) pp. 543–548

Parallel Schwarz Methods: Algebraic Construction of Coarse Problems, Implementation and Testing

Radim Blaheta, Petr Byczanski, Ondřej Jakl, and Jiří Starý

Institute of Geonics, Academy of Sciences CR,
Studentská 1768, 708 00 Ostrava – Poruba, Czech Republic
{blaheta, byczanski, jakl, stary}@ugn.cas.cz

Abstract. The paper describes domain decomposition methods of the Schwarz type with coarse problems constructed algebraically by aggregation of unknowns. The description includes a new method with no overlap of subdomains and interfaces on the coarse grid. Implementation issues are discussed for all the methods and their comparison is made on a model elasticity problem. Special attention is given to nonsymmetric hybrid preconditioners. A parallel implementation of the additive Schwarz method is tested on a 3D elasticity problem, employing a Beowulf cluster.

1 Introduction

Our aim is to solve discrete symmetric elliptic boundary value problems formulated as follows

$$\text{find} \quad u_h \in V_h : \quad a(u_h, v_h) \,=\, f(v_h) \quad \forall v_h \in V_h \tag{1}$$

where V_h is a finite element space, a is a bounded symmetric bilinear form and f is a bounded linear functional on V_h.

To be more specific, we solve a problem in a domain $\Omega \subset R^d\,(d = 2, 3)$ and the bilinear form in (1) takes one of the following forms

$$a(u, v) \,=\, \int_\Omega \sum_{ij} k_{ij}\, \frac{\partial u}{\partial x_i}\, \frac{\partial v}{\partial x_j}\, dx \tag{2}$$

$$a(u, v) \,=\, \int_\Omega \sum_{ijkl} c_{ijkl} \varepsilon_{ij}(u) \varepsilon_{kl}(v) dx \tag{3}$$

The bilinear form (2), corresponding to a *scalar boundary value problem* (e.g. heat conduction), is assumed to be symmetric and positive definite (SPD) on $\mathcal{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega_D\}$. The second form (3), which corresponds to an *elasticity problem*, is assumed to be SPD on $\mathcal{V} = \{v \in [H^1(\Omega)]^d : v = 0 \text{ on } \partial\Omega_D\}$. Above, $\varepsilon(u)$ is the small strain tensor induced by the displacement u, $H^1(\Omega)$ is the Sobolev space, $\partial\Omega_D$ is a part of the boundary of Ω, where the Dirichlet boundary conditions are imposed. For more details see e.g. [7].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 505–512, 2006.

The domain Ω is divided into a set $\mathcal{T}_h$ of finite elements. We shall assume triangular and tetrahedral elements in the case of $d = 2$ and $d = 3$, respectively. Then $V_h = \{v \in \mathcal{V} : v \mid_E \in P_1 \ \forall E \in \mathcal{T}_h\}$ with P_1 denoting the set of linear polynomials. Let us denote N_h the set of nodes of $\mathcal{T}_h$ and $\Phi_h = \{\phi_1, \ldots, \phi_n\}$ be the nodal FE basis of V_h. Then $u_h = \sum u_i \phi_i$, and the problem (1) can be transformed into the following algebraic problem:

$$\text{Find } u = (u_1, \ldots, u_n)^T, \quad Au = b, \quad u, b \in R^n \tag{4}$$

with A being symmetric positive definite $n \times n$ matrix.

Our main interest is to solve this algebraic system (4), assuming that the system can be very large with n being about 10^6 or even more. In such cases, the system should be solved iteratively, e.g. by the conjugate gradient (CG) method. For an efficient solution of large scale systems, we need a suitable preconditioner B to A as well as a possibility to perform the main operations in parallel. By main operations, we understand the matrix-vector multiplication with the matrix A and application of the preconditioner B.

For this purpose, we shall consider one and two-level additive Schwarz preconditioners

$$B_{1L} = \sum_1^m B_k, \quad B_k = R_k^T A_k^{-1} R_k \tag{5}$$

$$B_{2L} = B_{1L} + B_0, \quad B_0 = R_0^T A_0^{-1} R_0, \tag{6}$$

where A_k $(k = 1, \ldots, m)$ are local subproblems and A_0 is a coarse subproblem. The choice of subproblems is discussed in the next sections. We shall also consider nonsymmetric hybrid preconditioners defined by

$$B_H = B_{1L} + B_0(I - AB_{1L}), \tag{7}$$

$$B_H = B_0 + B_{1L}(I - AB_0). \tag{8}$$

More details can be found e.g. in [5], [4]. For the parallelization of matrix-vector multiplication and other necessary operations, we shall use a problem decomposition compatible with the local subproblems.

2 Local Subproblems and Parallel Implementation

We shall start with a FE triangulation $\mathcal{T}_h$ of the domain Ω and divide this triangulation into m parts $\mathcal{T}_k$. This division can be done in two steps: firstly $\mathcal{T}_h$ is divided into nonoverlapping sets $\mathcal{T}_k^0$, which are consecutively extended to overlapping sets $\mathcal{T}_k^\delta$. We shall denote

$$\Omega_k^0 = \cup\{E : E \in \mathcal{T}_k^0\}, \qquad \Omega_k^\delta = \cup\{E : E \in \mathcal{T}_k^\delta\}.$$

Practically, the division of $\mathcal{T}_h$ can be defined by exploiting specific features of the problem and the triangulation (easy for structured grids) or using general algorithms (see e.g. [6] for a survey).

Now, we can define the local matrices A_k and restrictions R_k. Let A_k^δ be the FE matrix arising from assembling the element matrices A_E for $E \in \mathcal{T}_h^\delta$. Then A_k will be the matrix arising from A_k^δ by incorporating homogeneous Dirichlet type boundary conditions on the inner boundary $\partial\Omega_k^\delta \setminus \partial\Omega$. Note that the boundary conditions on the outer boundary $\partial\Omega_k^\delta \cap \partial\Omega$ are given from the solved boundary value problem.

Let us denote

$$N_k^\delta = N_h \cap \Omega_k^\delta, \quad N_k^B = N_h \cap (\partial\Omega_k^\delta \setminus \partial\Omega), \quad N_k = N_k^\delta \setminus N_k^B,$$

then the restriction R_k can be represented by a Boolean matrix with nonzero elements only in the positions $(ii),\ i \in N_k$,

$$(R_k)_{ii} \ = \ I_{DOF},$$

where $I_{DOF} = 1$ for the scalar problem and I_{DOF} is 2x2 and 3x3 identity matrix for 2D and 3D elasticity, respectively. The dimensions of R_k are $n_{DOF}\cdot\mathrm{card}(N_k)\times n_{DOF} \cdot \mathrm{card}(N_h)$, where n_{DOF} is the number of degrees of freedom (DOF) per node.

The local triangulations can be distributed over m processors of a parallel computing systems and the construction and solution of the local problems with A_k can be done in parallel. Frequently, the solution of subsystems is simplified, e.g. by using an incomplete factorization of A_k.

For the parallel implementation of the CG method with any of the Schwarz preconditioners (5) – (8), it is desirable to parallelize also the other necessary operations, mainly the matrix-by-vector multiplication. To this end, we can use a nonoverlapping decomposition of the set of nodes N_h and consequently the set of the DOFs. Let

$$N_h = \bigcup_1^m N_k', \quad N_k' \subset N_k \subset N_k^\delta$$

be such decomposition. Then due to extension of $\mathcal{T}_k^0$ to $\mathcal{T}_k^\delta$, we can manage that all nodes, which are neighbours of some node in N_k', belong to N_k^δ. In this case, the full row/column information concerning DOFs corresponding to the nodes in N_k' can be found in A_k^δ. It enables a parallel implementation of the matrix-vector product. Note that the parallelization of the scalar products and vector updates can use the same decomposition.

Up to now, we assumed a certain distribution of the triangulation over the processors and parallel construction of the subproblems. Another approach, especially useful for the implementation of Schwarz methods in existing FE software, consists in assembling the full matrix A and construction of $\bar{A}_k^\delta$ according to

$$\bar{A}_k^\delta \ = \ R_k A R_k^T.$$

Note that A_k^δ and $\bar{A}_k^\delta$ differ only in diagonal terms corresponding to DOFs associated with the nodes in N_k^B.

Let δ express the size of overlap, $H_d = \max \operatorname{diam}\{\Omega_k^\delta\}$ and the subproblems are solved exactly. Then the efficiency of the preconditioner B_{1L} is characterized by

$$\operatorname{cond}\,(B_{1L}A) \,\leq\, C(1+\delta^{-2}) \tag{9}$$

with C independent on $H_d,\ h,\ \delta$. For the proof see e.g. [3] and the references therein.

3 A Coarse Problem by Aggregation

The preconditioner B_{1L}, involving only the local subproblems, is not numerically scalable, i.e. its efficiency deteorates with an increasing number of subdomains, cf. (9) with δ proportional to H_d. A remedy is to include a global subproblem into the preconditioners, see (6) – (8).

If the triangulation $\mathcal{T}_h$ is a refinement of a coarser triangulation $\mathcal{T}_H$, then it is easy to define A_0 as the FE matrix corresponding to the discretization of the solved boundary value problem on $\mathcal{T}_H$ and R_0^T as a natural interpolation given by the embedding $V_H \subset V_h$. The use of a coarser nested triangulation is efficient, but has also several drawbacks: in many cases, it is impossible to define such coarser triangulation; it is difficult to guarantee that the solution of systems with A_0 will be approximately as expensive as the solution of the local subproblems; and finally, it is difficult to implement such two-level Schwarz preconditioners in an existing FE software.

The above difficulties motivate an interest in coarse problems created algebraically from the information involved already in the FE matrix A corresponding to $\mathcal{T}_h$. One of the simplest and efficient ways is to create A_0 by *aggregation* of DOFs [1], i.e. by dividing the set of nodes N_h into nonoverlapping groups,

$$N_h \,=\, G_1 \cup \ldots \cup G_N \quad G_i \cap G_j = 0 \quad \text{for } i \neq j. \tag{10}$$

Then, we can define

$$A_0 \,=\, \left(a_{ij}^{(0)}\right)_{N\times N}, \quad a_{ij}^{(0)} \,=\, \sum_{k\in G_i} \sum_{l\in G_j} a_{kl}\,, \tag{11}$$

$$R_0 \,=\, (r_{ij})_{N\times n}\,, \quad r_{ij} \,=\, 1 \text{ if } j \in G_i, \quad r_{ij} = 0 \text{ otherwise}\,. \tag{12}$$

For elasticity problems, we shall use the nodal blocks of DOFs, i.e. $r_{ij} = I_{DOF}$ if $j \in G_i$ and a_{kl} will be the appropriate $n_{DOF} \times n_{DOF}$ block of A.

The necessary information about aggregation is usually stored in one destination vector $d \in R^n,\ d_i = j$ iff $j \in G_i$. Then the implementation of the prolongation and restriction is very easy and straightforward. It is also possible to develop efficient algorithm for computation of A_0 according to (11), even in the case when the sparse structure of A_0 is not known apriori and when A exists only in the distributed form of $A_k^\delta,\ k = 1, \ldots, m$.

If $\mathrm{diam}\{\bigcup E : \text{some of vertices of } E \text{ are in } G_i\} \leq H$ and some additional but natural assumptions are fulfilled, then

$$\mathrm{cond}\,(B_{2L}A) \leq C(1 + h^{-1}H + \delta^{-2}H^2)\,. \tag{13}$$

with C independent on $H_d,\ H,\ h,\ \delta$. For the proof see [3], [8], [9]. The estimate (13) indicates that the coarse problem created by aggregation can ensure the numerical scalability.

Aggregation can be easily created on regular structured grids by regular clustering. On unstructured grids, we can use several algorithms based on simple matrix graph or more sophisticated ones based on the strength of couplings between nodes, see e.g. [8].

4 Coarse Problems with Interfaces

The flexibility of aggregation technique allows also to introduce new variants of the Schwarz method. Let us describe one of them, see [3].

Let $\mathcal{T}_k^0$ $(k = 1, \ldots, m)$ define a nonoverlapping decomposition of $\mathcal{T}_h$, Ω_k^0 is the subdomain associated with $\mathcal{T}_k^0$ (see Section 2),

$$N_k^0 = N_h \cap \bar{\Omega}_k^0, \quad N_k^{0B} = N_h \cap (\partial\Omega_k^0 \setminus \partial\Omega), \quad N_h^{IB} = \bigcup_{k=1}^{m} N_k^{0B}.$$

Then we can define the local problems A_k by assembling the FE matrices for the elements from $\mathcal{T}_k^0$ and incorporating the homogeneous Dirichlet boundary conditions on $\partial\Omega_k^0 \setminus \partial\Omega$. The restriction R_k is defined by the inclusion $(N_k^0 \setminus N_k^{0B}) \subset N_h$.

The coarse global problem A_0 is defined by an aggregation $N_h = G_1 \cup \ldots \cup G_N$ which involves all nodes on N_h^{IB} as single-element groups (if $i \in N_h^{IB}$, then there is k such that, $G_k = \{i\}$). DOFs associated with the nodes on N_h^{IB} are presented only in the coarse problem.

The efficiency of the corresponding additive two-level preconditioner B_{2L} is characterized by

$$\mathrm{cond}\,(B_{2L}A) \leq 2(1 - \gamma)^{-1}\,, \tag{14}$$

where $\gamma = \cos(V_0, W_0)$ with V_0 being the coarse space created by aggregation and W_0 being any direct sum complement of V_0, i.e. $V_h = V_0 \oplus W_0$. See [3] for the proof.

The advantages of the new method are the following: the local subproblems are fully independent, the aggregation on the inner boundaries $\partial\Omega_k^0 \setminus \partial\Omega$ are determined in advance, the aggregation within $\partial\Omega_k^0$ is independent on the other subdomains and can be done independently. Moreover, this way of decomposition induces favourable properties for the use of nonsymmetric hybrid preconditioner, whose efficiency is independent of the number of local problems.

5 Nonlinear and Nonsymmetric Preconditioners

The preconditioners (5) – (8) require to solve subproblems for implementation of the operations $g_k = A_k^{-1} w_k$. Although these subproblems are smaller than the original problems, it is still inefficient to solve them exactly by direct methods. Based on evaluation of numerical tests, we adopted the following strategy.

The matrices of the local subproblems A_k are replaced by an incomplete factorization $F_k = L_k U_k$, and problems with F_k are solved instead of the original subproblems. For the coarse subproblem, this procedure is not sufficient. Therefore, we solve the problem $A_0 g_0 = w_0$ by inner CG iterations preconditioned by an incomplete factorization F_0 of A_0. The relative residual norm criterion with $\varepsilon_0 = 10^{-1}$ is used for stopping the inner iterations.

The described strategy provides an approximate two-level preconditioner $\tilde{B}_{2L}$, that is a nonlinear mapping which represents some approximation to the linear SPD preconditioner B_{2L}. The use of the nonsymmetric hybrid preconditioner B_H or its inexact form $\tilde{B}_H$ implies a further violation of the standard requirements for linear SPD preconditioner. Nevertheless, both B_{2L} and $\tilde{B}_H$ can be implemented to standard CG method or its flexible variant. The implementation in the flexible GPCG[s] with explicit orthogonalization to $s \geq 1$ previous search directions guarantees correctness and convergence of the procedure, see e.g. [2].

6 Numerical Results

The efficiency of various preconditioners arising from implementation of the described ideas can be compared by solving a model elasticity problem (plane deformation) in $\Omega = \langle 0, 2\rangle \times \langle 0, 3\rangle$ with pure Dirichlet boundary conditions ($\partial\Omega_D = \partial\Omega$). The problem can be written as follows: Find $u \in \mathcal{V}$,

$$a(u, v) = \int_\Omega (f_1 v_1 + f_2 v_2) dx \qquad \forall v \in \mathcal{V}\,,$$

where a is the bilinear form (3), $f_1(x_1, x_2) = f_2(x_1, x_2) = 7.5 + 2.5x_1 + 1.1x_2$. The elasticity tensor $C = (c_{ijkl})$ has the following nonzero elements

$$c_{iiii} = \lambda + 2\mu, \quad c_{iijj} = \lambda, \quad c_{ijij} = c_{ijji} = \mu$$

for $i, j \in \{1, 2\}$, $i \neq j$. If E and ν denote the elasticity modulus and Poisson ratio, then $\lambda = E\nu(1+\nu)^{-1}(1-2\nu)^{-1}$ and $\mu = E(1+\nu)^{-1}/2$, respectively.

The problem is discretized by linear triangular FE on a uniform grid with the mesh size $h = 1/30$. The local problems are given on subdomains $\Omega_k = \langle 0, 2\rangle \times \langle x_k, x_{k+1}\rangle$ with overlap $\delta = 2h$ or zero overlap for the method with interfaces on the coarse grid. The subproblems are solved exactly.

The required numbers of iterations for the accuracy $\varepsilon = 10^{-3}$ and various additive (AP) and hybrid (HP) Schwarz preconditioners can be seen in Table 1. The hybrid preconditioners are used in nonsymmetric form in combination with

Table 1. The numbers of iterations for the relative accuracy $\varepsilon = 10^{-3}$. AP denotes additive preconditioner (5,6) , HP denotes the hybrid preconditioner (7) + GPCG[1].

Type	δ	Coarse problem	Number of subdomains									
			2		4		8		16		24	
AP	2h	no coarse problem	20	47	25	56	34	77	47	111	56	138
AP	2h	nested, $H = 3h$	8	13	8	13	8	14	8	16	9	18
HP	2h	nested, $H = 3h$	6	10	6	11	6	12	7	13	8	16
AP	2h	regular aggr. 2×2	15	35	17	37	18	41	20	45	20	47
HP	2h	regular aggr. 2×2	11	23	12	26	12	27	13	27	14	30
AP	0	aggr. 2×2 + interf.	15	29	16	32	16	32	17	38	18	42
HP	0	aggr. 2×2 + interf.	8	16	8	18	9	18	9	21	9	22

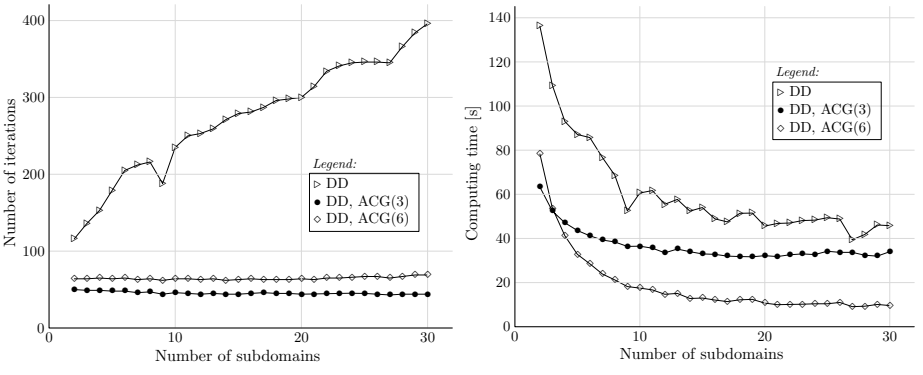


Fig. 1. Stabilization of the number of iterations due to coarse problems by aggregation (*left*). ACG(p) denotes regular clustering of p^3 nodes. Computing times for additive preconditioners (*right*). The coarse problem should be balanced with the local ones.

GPCG[1]. The coarse problem uses either nested coarse triangular grid with the mesh size $H = 2h$, aggregation with clustering 2×2 square macroelements or the same aggregation with interfaces. For each number of subdomains, two columns show the numbers of iterations for $\nu = 0.3$ and $\nu = 0.49$, respectively.

The efficiency of parallel solvers based on the CG method and additive Schwarz preconditioners without and with coarse problem constructed by aggregation can be seen from the diagrams in Figure 1.

The solved problem is a square footing benchmark. This 3D elasticity problem with isotropic elastic material ($E = 14.0\,\mathrm{MPa}$, $\nu = 0.3$, $\rho = 2.5\,\mathrm{g\,cm^{-3}}$) is defined on a cuboid $100 \times 100 \times 40\,\mathrm{m}$. The boundary conditions include zero normal displacements and zero tangential stresses on all sides with the exception of the top one. A constant pressure is given on the central part ($10 \times 10\,\mathrm{m}$) of the top side. One quarter ($50 \times 50 \times 40\,\mathrm{m}$) of the symmetric problem domain is discretized firstly into $80 \times 80 \times 80$ equal bricks that are subsequently divided into tetrahedral FE. The discretized problem has 1 594 323 DOFs and is solved with a relative accuracy $\varepsilon = 10^{-4}$.

The parallel solution uses the additive Schwarz method with one-dimensional decomposition of the domain up to 30 subdomains and a coarse problem created by regular aggregation of $3 \times 3 \times 3$ or $6 \times 6 \times 6$ nodes. The computations are performed on a Linux cluster consisting of 16 computing nodes with 2 × AMD Athlon/2600 GHz processors and 3 GB memory each. The nodes are interconnected via Myrinet L9 2 Gb/s and Fast Ethernet networks.

7 Concluding Remarks

Our experience shows that the Schwarz preconditioners and methods can be used for development of efficient and scalable parallel solvers at least when working on smaller parallel computing systems. These methods are flexible enough for balancing the computational load on the processors, adopting inexact solvers etc. On the other hand, a special care should be devoted to physical or numerical anisotropy, high Poisson ratio and similar difficulties.

The Schwarz technique with algebraic coarse problem created by aggregation can be easily implemented into existing FEM software. Another implementation, which starts with a FE grid decomposition and parallel assembling of the local problems, is useful if the solved system must be several times updated. It is e.g. the case of solving nonlinear problems.

Acknowledgement. The work is supported by the grant No. S3086102 of the Academy of Sciences of the Czech Republic.

References

1. Blaheta, R. : A multilevel method with overcorrection by aggregation for solving discrete elliptic problems. J. Comp. Appl. Math. **24** (1988) 227–239
2. Blaheta, R.: GPCG – generalized preconditioned CG method and its use with non-linear and non-symmetric displacement decomposition preconditioners. Numer. Linear Algebra Appl. **9** (2002) 527-550
3. Blaheta, R.: Space decomposition Preconditioners and Parallel Solvers. In: M. Feistauer et al (eds.), Numerical Mathematics and Advanced Applications, Springer-Verlag, Berlin (2004) 20–38
4. Toselli, A., Widlund, O.: Domain Decomposition Methods – Algorithms and Theory. Springer, Berlin (2005)
5. Smith, B. F., Bjørstad, P. E., Gropp, W. D.: Domain Decomposition – Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, Cambridge (1996)
6. Saad, Y.: Iterative methods for sparse linear systems. SIAM, Philadelphia (2003)
7. Brenner, S. C., Scott, L. R.: The Mathematical Theory of Finite Element Methods (Second Edition). Springer-Verlag, New York (2002)
8. Brezina, M.: Robust iterative methods on unstructured meshes. PhD. thesis, University of Colorado at Denver (1997)
9. Jenkins, E. W., Kelley, C. T., Miller C. T., Kees, C. E.: An aggregation-based domain decomposition preconditioner for groundwater flow. SIAM J. Sci. Comput. **23** (2001) 430–441

Direct Solution of Linear Systems of Size 10^9 Arising in Optimization with Interior Point Methods*

Jacek Gondzio and Andreas Grothey

School of Mathematics, University of Edinburgh,
JCMB, King's Buildings, Edinburgh, EH9 3JZ, UK
J.Gondzio@ed.ac.uk, A.Grothey@ed.ac.uk

Abstract. Solution methods for very large scale optimization problems are addressed in this paper. Interior point methods are demonstrated to provide unequalled efficiency in this context. They need a small (and predictable) number of iterations to solve a problem. A single iteration of interior point method requires the solution of indefinite system of equations. This system is regularized to guarantee the existence of triangular decomposition. Hence the well-understood parallel computing techniques developed for positive definite matrices can be extended to this class of indefinite matrices. A parallel implementation of an interior point method is described in this paper. It uses object-oriented programming techniques and allows for exploiting different block-structures of matrices. Our implementation outperforms the industry-standard optimizer, shows very good parallel efficiency on massively parallel architecture and solves problems of unprecedented sizes reaching 10^9 variables.

1 Introduction

Since their discovery [1] interior point methods (IPMs) have enjoyed well-deserved interest and have been subject of intensive study which led to a development of complete theory [2] and a thorough understanding of their implementation [3]. Interior point methods for optimization have a number of advantages. Depending on the algorithm used they guarantee finding a solution of the problem in not more than $\mathcal{O}(\sqrt{n})$ or $\mathcal{O}(n)$ iterations where n is the problem dimension. In practice they display a faster convergence suggesting that they enjoy $\mathcal{O}(\log n)$ complexity. But most of all, IPMs are reliable and can be implemented to provide unprecedented efficiency when applied to solve very large scale problems. We illustrate these features in this paper.

The bulk of work in every iteration of an interior point method is the solution of an indefinite system of equations. This system is regularized [4] to guarantee that an (indefinite) triangular Cholesky-like decomposition of it can be found. There exists a vast body of literature about parallel Cholesky decomposition.

* Supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/R99683/01.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 513–525, 2006.

Indeed, the method is often implemented to exploit block-operations and all independent operations are executed on different processors.

To increase the degree of parallelism in the implementation of Cholesky factorization one looks for such an ordering of a sparse matrix which concentrates nonzero entries in independent blocks and if possible limits the fill-in to these blocks. Very large scale optimization problems by their very nature display block-structure. It is a consequence of the way how these problems are modelled. Models of engineering problems commonly involve indexing variables over discretizations in several dimensions hence they replicate few generic blocks. Such blocks are usually loosely coupled, and the word "loosely" translates into a high degree of sparsity displayed by matrices involved. Examples of such models include features such as:

- dynamics: inter-temporal connections are spread over a long horizon,
- uncertainty: scenarios are induced by stochastic (event) tree, or
- spatial distribution: functions are discretized over their domains.

We have developed a structure-exploiting optimization code called OOPS (Object-Oriented Parallel Solver) [5, 6, 7]. OOPS is an implementation of the pri-mal-dual interior point method which uses all recent algorithmic advances (see `http://maths.ed.ac.uk/~gondzio/parallel/solver.html`).
It allows *any* block-structure of the optimization problem to be exploited by the linear algebra operations of the interior point method. In this paper we illustrate the parallel efficiency of this software. We apply it to a class of min-variance portfolio optimization problems [6, 8]. These models are quadratic (or nonlinear when higher order moments are used to measure risk). Stochastic programming modelling techniques [9] are used and this leads to challenging optimization problems which defy standard software.

The paper is organised as follows. In Section 2 interior point methods for optimization are briefly explained. In Section 3 the linear algebra operations involved by IPMs are discussed and exploiting block structure of matrices in these operations is addressed. In Section 4 the object-oriented implementation of linear algebra operations is briefly discussed and in Section 5 the formulation of min-variance portfolio optimization problems is given. In Section 6 the computational results are reported and in Section 7 the conclusions are given.

2 Interior Point Methods for Optimization

Developed over the last two decades, interior point methods have gained a strong position in the area of optimization. They easily generalise from linear, through quadratic to nonlinear programming and for all these classes of problems provide efficient algorithms. In this section we discuss IPMs applied to convex nonlinear programs and briefly comment on the simplified linear and quadratic models. Next, we show how their implementation can take advantage of three particular block-structures: primal and dual block-angular and bordered block-diagonal. The reader interested in the theory of IPMs is encouraged to consult [2]; aspects of their implementation for general problems are discussed in [3].

Consider the convex nonlinear optimization problem

$$\min \quad f(x) \qquad \text{s.t.} \qquad g(x) \leq 0,$$

where $x \in \mathcal{R}^n$, and $f : \mathcal{R}^n \mapsto \mathcal{R}$ and $g : \mathcal{R}^n \mapsto \mathcal{R}^m$ are convex, twice differentiable. Having introduced a nonnegative slack variable $z \in \mathcal{R}^m$ the inequality constraint can be rewritten as an equation $g(x) + z = 0$. The inequality $z \geq 0$ is "replaced" by the logarithmic barrier terms giving the following barrier problem

$$\min \quad f(x) - \mu \sum_{i=1}^{m} \ln z_i \qquad \text{s.t.} \qquad g(x) + z = 0$$

and the associated Lagrangian

$$L(x, y, z, \mu) = f(x) + y^T(g(x) + z) - \mu \sum_{i=1}^{m} \ln z_i.$$

The first order optimality conditions for the barrier problem (conditions for a saddle point of Lagrangian) have the following form:

$$\begin{array}{rl} \nabla_x L(x, y, z, \mu) = 0 \Rightarrow & \nabla f(x) + \nabla g(x)^T y = 0 \\ \nabla_y L(x, y, z, \mu) = 0 \Rightarrow & g(x) + z = 0 \\ \nabla_z L(x, y, z, \mu) = 0 \Rightarrow & YZe = \mu e \\ & (y, z) \geq 0, \end{array} \tag{1}$$

where $Y = \text{diag}\{y_1, y_2, \cdots, y_m\}$ and $Z = \text{diag}\{z_1, z_2, \cdots, z_m\}$. Interior point algorithm for nonlinear programming [2] applies Newton method to solve this system of equations and gradually reduces the barrier parameter μ to guarantee the convergence to the optimal solution of the original problem. The Newton direction is obtained by solving the system of linear equations:

$$\begin{bmatrix} Q(x,y) & A(x)^T & 0 \\ A(x) & 0 & I \\ 0 & Z & Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - A(x)^T y \\ -g(x) - z \\ \mu e - YZe, \end{bmatrix}, \tag{2}$$

where the matrices $Q(x, y) = \nabla^2 f(x) + \sum_{i=1}^{m} y_i \nabla^2 g_i(x) \in \mathcal{R}^{n \times n}$ and $A(x) = \nabla g(x) \in \mathcal{R}^{m \times n}$ are the Hessian of Lagrangian and the Jacobian of constraints, respectively. After substituting $\Delta z = \mu Y^{-1} e - Ze - ZY^{-1}\Delta y$ in the second equation we get

$$\begin{bmatrix} -Q(x,y) & A(x)^T \\ A(x) & \Theta_D \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = \begin{bmatrix} \nabla f(x) + A(x)^T y \\ -g(x) - \mu Y^{-1} e \end{bmatrix}, \tag{3}$$

where $\Theta_D = ZY^{-1}$ is a diagonal scaling matrix. The matrix involved in this set of linear equations is symmetric and indefinite. For convex optimization problem (when f and g are convex), the matrix Q is positive semidefinite.

If the variables x have a sign restriction ($x \geq 0$) the above system contains an additional primal scaling matrix $\Theta_P = XS^{-1}$ and the indefinite matrix used in (3) takes the form

$$H = \begin{bmatrix} -(Q(x,y) + \Theta_P^{-1}) & A(x)^T \\ A(x) & \Theta_D \end{bmatrix}. \tag{4}$$

In case of linear programming matrix A is constant and $Q = 0$ and in case of quadratic programming matrices A and Q are constant. LP and QP problems are usually formulated with the equality constraints and nonnegative variables hence the corresponding reduced systems take forms

$$H_{LP} = \begin{bmatrix} -\Theta_P^{-1} & A^T \\ A & \end{bmatrix} \qquad \text{and} \qquad H_{QP} = \begin{bmatrix} -(Q + \Theta_P^{-1}) & A^T \\ A & \end{bmatrix},$$

respectively. To simplify notation in the rest of this paper we will drop arguments of $A(x)$ and $Q(x,y)$ in (4) and use A and Q instead.

Standard approach [10, 11] requires 2×2 pivots be used in symmetric decomposition of matrix H. In our implementation we follow [4] and regularize matrix H to transform it to a quasi-definite matrix [12]. We add primal and dual regularizations and obtain

$$H_R = \begin{bmatrix} -Q - \Theta_P^{-1} & A^T \\ A & \Theta_D^{-1} \end{bmatrix} + \begin{bmatrix} -R_p & 0 \\ 0 & R_d \end{bmatrix},$$

where diagonal positive definite matrices $R_p \in \mathcal{R}^{n \times n}$ and $R_d \in \mathcal{R}^{m \times m}$ can be interpreted as adding proximal terms to the primal and dual objective functions, respectively. After this modification H_R becomes quasi-definite hence for any symmetric row and column permutation a triangular decomposition $H_R = LDL^T$ exists with diagonal matrix D. This matrix has exactly n negative and m positive pivots. Since 2×2 pivots do not have to be used, the symbolic and numerical factorization phases can be split as in the positive definite case. This is an important feature in the implementation of interior point methods and it is essential for the implementation of structure exploiting linear algebra techniques.

3 Linear Algebra for Block-Structured Matrices

There are many structures that could be exploited by an interior point solver. We restrict our attention to those in which Hessian and Jacobian matrices are built of blocks. The well-known elementary structures observed in Jacobian matrices A are:

$$\begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \\ B_1 & B_2 & \cdots & B_n & B_0 \end{bmatrix}, \begin{bmatrix} A_1 & & & C_1 \\ & A_2 & & C_2 \\ & & \ddots & \vdots \\ & & & A_n & C_n \end{bmatrix}, \begin{bmatrix} A_1 & & & C_1 \\ & A_2 & & C_2 \\ & & \ddots & \vdots \\ & & & A_n & C_n \\ B_1 & B_2 & \cdots & B_n & B_0 \end{bmatrix}, \tag{5}$$

and represent a primal block-angular, dual block-angular and row and column bordered structure, respectively. However, many real-life problems have more complicated nested structures that embed those and other elementary blocks. We assume that Hessian matrix Q has a closely related structure induced by the column partitioning of A.

The corresponding matrix H can be reordered leading to structures which can be exploited by a parallel factorization. Suppose, for example, that the augmented system matrix has the symmetric bordered block-diagonal structure.

$$H = \begin{bmatrix} H_1 & & & G_1^T \\ & H_2 & & G_2^T \\ & & \ddots & \vdots \\ & & & H_n \; G_n^T \\ G_1 & G_2 & \cdots & G_n \; H_0 \end{bmatrix}, \tag{6}$$

where $H_i \in \mathcal{R}^{n_i \times n_i}, i = 0, ..., n$ and $G_i \in \mathcal{R}^{n_0 \times n_i}, i = 1, ..., n$. Under the condition that H is quasi-definite (and we assume that it has been regularized to satisfy this condition), we can obtain a Cholesky-like block decomposition $H = LDL^T$, where

$$L = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_n \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} \; L_0 \end{bmatrix}, \quad D = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \ddots & \\ & & & D_n \\ & & & & D_0 \end{bmatrix}$$

and

$$H_i = L_i D_i L_i^T \tag{7a}$$

$$L_{n,i} = G_i L_i^{-T} D_i^{-1} \tag{7b}$$

$$S = H_0 - \sum_{i=1}^{n} G_i H_i^{-1} G_i^T \;=\; L_0 D_0 L_0^T. \tag{7c}$$

This decomposition can be used to compute the solution to the system $Hu = b$, where $u = (u_1, \ldots, u_n, u_0)^T$, $b = (b_1, \ldots, b_n, b_0)^T$ by the following sequence of operations:

$$z_i = L_i^{-1} b_i, \quad i = 1, \ldots, n \tag{8a}$$

$$z_0 = L_0^{-1}(b_0 - \sum_{i=1}^{n} L_{n,i} z_i) \tag{8b}$$

$$y_i = D_i^{-1} z_i, \quad i = 0, \ldots, n \tag{8c}$$

$$u_0 = L_0^{-T} y_0 \tag{8d}$$

$$u_i = L_i^{-T}(y_i - L_{n,i}^T u_0), \quad i = 1, \ldots, n \tag{8e}$$

Note that blocks $L_{n,i}$ do not have to be stored. This leads to obvious memory savings: blocks $L_{n,i}$ are computed because they contribute to the Schur complement matrix S in (7c) but they do not have to be stored any longer. This also results in time savings, since the multiplications with blocks $L_{n,i}$ and $L_{n,i}^T$ in equations (8b) and (8e) are executed as sequences of less expensive operations using the following formulae:

$$\begin{aligned} L_{n,i}z_i &= G_i L_i^{-T} D_i^{-1} z_i = G_i(L_i^{-T} D_i^{-1} z_i), \\ L_{n,i}^T u_0 &= D_i^{-1} L_i^{-1} G_i^T u_0 = D_i^{-1} L_i^{-1}(G_i^T u_0). \end{aligned}$$

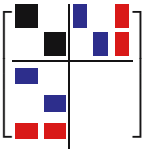
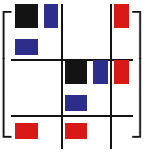
Consequently, these operations have complexity $\mathcal{O}(\mathrm{nz}(G_i)+\mathrm{nz}(L_i))$ rather than $\mathcal{O}(\mathrm{nz}(L_{n,i}))$ and it is usual to expect that $\mathrm{nz}(L_{n,i}) \gg \mathrm{nz}(G_i)+\mathrm{nz}(L_i)$.

The block factorization $H = LDL^T$ together with computations (7) and (8) are therefore an *implicit* representation of the inverse of H.

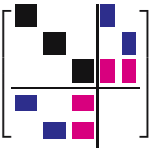
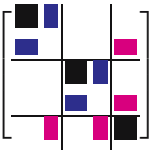
Apart from the above mentioned efficiency gains the use of a block implicit inverse facilitates the parallelisation of the calculation. Indeed most of the two operations: computing the symmetric decomposition (7) and using it for solving system of equations (8) will parallelise trivially. The sums in (7c) and (8b) require parallel communications, while operations involving L_0 and D_0 (namely factorization of S and the L_0^{-1}, L_0^{-T} operations in (8b, 8d)) have to be performed on all processors.

We conclude this section by giving examples of reordered matrices H for the three common structures mentioned earlier. The reordering preserves symmetry, that is we apply the same permutation to block-rows and block-columns of H. To simplify the presentation we have made three assumptions: (i) Jacobian matrices A in (5) have only two diagonal blocks, (ii) Hessian matrices Q have block-diagonal structures induced by the block-column partitions in A, and (iii) the $(2,2)$ block in H is zero.

Primal Block-Angular Structure
Blocks of H have been permuted following the reordering $\{1,3;\ 2,4;\ 5\}$.

$H =$  , $PHP^T =$ 

Dual Block-Angular Structure
Blocks of H have been permuted following the reordering $\{1,4;\ 2,5;\ 3\}$.

$H =$  , $PHP^T =$ 

Row and Column Bordered Block-Diagonal Structure
Blocks of H have been permuted following the reordering $\{1,4;\ 2,5;\ 3,6\}$.

$$H = \left[\begin{array}{ccc|ccc} \blacksquare & & & \blacksquare & & \blacksquare \\ & \blacksquare & & \blacksquare & \blacksquare & \blacksquare \\ & & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \hline \blacksquare & & & \blacksquare & & \\ & \blacksquare & & \blacksquare & & \\ & & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{array}\right], \qquad PHP^T = \left[\begin{array}{ccc|ccc} \blacksquare & & & \blacksquare & & \blacksquare \\ \blacksquare & & & & \blacksquare & \blacksquare \\ & \blacksquare & & \blacksquare & \blacksquare & \\ \hline & & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ & \blacksquare & & \blacksquare & & \\ \blacksquare & \blacksquare & & \blacksquare & \blacksquare & \blacksquare \end{array}\right]$$

4 Object-Oriented Implementation

As shown in the previous section block-structured matrices can be reordered by permuting blocks to such forms which offer an advantage for parallel computations. Many real life problems have more complicated structures which include a nesting of elementary block-structures. The nested block-structure of a matrix can be thought of as a tree. Its root is the whole matrix and every block of a particular sub-matrix is a child node of the node representing this sub-matrix. Leaf nodes correspond to the elementary sub-matrices that can no longer be divided into blocks. With every node of the tree we associate information about the type of structure this node represents. This tree determines the order and type of linear algebra operations needed by the interior point algorithm.

The design of OOPS follows object-oriented principles, treating the blocks (and sub-blocks) of matrices as objects [7, 5]. We use a `Matrix` interface that defines all linear algebra methods needed for an interior point algorithm such as `Factorize`, `solveL` or `solveLt`. The interface also provides all operations required for a given `Matrix` object to become a sub-matrix in the nested structured matrix. In such case `Matrix` object is accessed by the `Matrix` object corresponding to its ancestor in the tree defining the nested structure.

Several specialised classes provide concrete implementation of the `Matrix` interface, each exploiting a different possible structure such as for example *primal block-angular*, *dual block-angular*, *bordered block-diagonal*, *rank corrector* as well

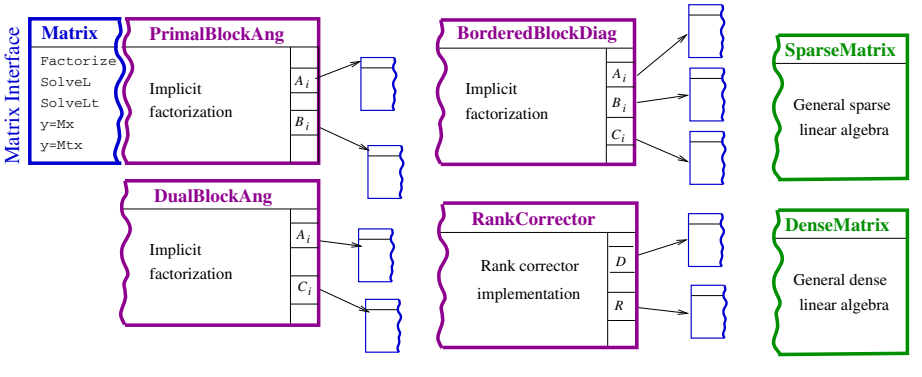


Fig. 1. The matrix interface and several implementations of it

as a standard *dense matrix* or *sparse matrix*. The implementing classes can be classified as either *leaf-node* classes such as `DenseMatrix` or `SparseMatrix` or the *complex* classes, such as `PrimalBlockAng`, `DualBlockAng`, `RankCorrector` or `BorderedBlockDiag` (see Figure 1). The design of OOPS library is based on the assumption that an efficient implementation of all methods for a *complex* class can be reduced to a sequence of methods performed on its constituents [7, 5]. The top-level class here does not need to know the exact type of its constituent objects nor whether they themselves are of *leaf-node* or *complex* type, it merely needs to know that they support the methods of the interface and assumes that they do so in a way most efficient for their particular structure. Summing up, OOPS re-creates the structured matrix tree with a tree of `Matrix` objects.

5 Large-Scale Portfolio Optimization Problems

To demonstrate the efficiency of OOPS we have applied it to solve a class of portfolio optimization problems. We follow a description of these problems given in [9] and consider extensions [13] which allow for higher order moments to be incorporated into the model (for more details see [6, 8]).

We consider investment of an initial wealth b into assets $j = 1, \ldots, J$ with uncertain returns. We allow the portfolio be be rebalanced at discrete times $t = 1, \ldots, T$ and we want (i) to maximize the expected final wealth of the portfolio at time T, and (ii) to minimize the associated risk. The standard formulation of this problem leads to Markowitz portfolio optimization problem [14, 15].

The stochastic process is approximated by a discrete distribution and modelled as an event tree. With each node of the tree i we associate the time stage t it belongs to, the ancestor node $a(i)$, and the list of successor nodes (children) which belong to the next time stage $t + 1$. The probability of reaching node i is denoted by p_i and it is equal to the product of probabilities associated with all arcs in the the event tree leading from the root node to node i.

At every node i and for each asset j we define three variables $x^h_{i,j}, x^b_{i,j}$ and $x^s_{i,j}$ which denote the amount of asset held, bought and sold, respectively. The *inventory constraint* for an asset i writes:

$$(1 + r_{i,j})x^h_{a(i),j} = x^h_{i,j} - x^b_{i,j} + x^s_{i,j}, \quad \forall i \neq 0, j, \tag{9}$$

where $r_{i,j}$ is the return associated with a branch (arc of event tree) connecting ancestor node $a(i)$ with i. The *budget constraint* imposes an equality of cash inflow from selling assets and cash outflow for buying new assets:

$$\begin{array}{l} \sum_j (1 + c_t)v_j x^b_{i,j} = \sum_j (1 - c_t)v_j x^s_{i,j} \quad \forall i \neq 0 \\ \sum_j (1 + c_t)v_j x^b_{0,j} = b, \end{array} \tag{10}$$

where v_j is a unit price of asset j and c_t is a transaction cost. This constraint takes a simplified form for the root node $i = 0$ where one can only purchase assets of total value equal to the initial budget b.

In the standard approach the wealth is measured by an expected value of the final portfolio converted into cash

$$y = I\!E((1-c_t)\sum_{j=1}^{J} v_j x^h_{T,j}) = (1-c_t)\sum_{i\in L_T} p_i \sum_{j=1}^{J} v_j x^h_{i,j}, \tag{11}$$

where L_T denotes the subset of nodes in the event tree which belong to the terminal stage. It is common to use the variance of return as a risk measure:

$$r \;=\; \mathrm{Var}((1-c_t)\sum_{j=1}^{J} v_j x^h_{T,j}) = \sum_{i\in L_T} p_i[(1-c_t)\sum_{j} v_j x^h_{i,j} - y]^2. \tag{12}$$

The min-variance portfolio minimizes the aggregate objective of the form $y-\lambda r$ which combines two criteria into a single one and uses an arbitrary parameter λ to express the wealth-risk trade-off. The larger the λ the more attention is paid to risk hence more of safe assets are selected into the portfolio.

Summing up, the standard multi-stage Markowitz portfolio optimization problem consists in minimizing $y - \lambda r$ subject to constraints (9), (10), (11) and (12). Models require all decision variables $x^h_{t,j}, x^b_{t,j}, x^s_{t,j}$ be nonnegative and may impose additional constraints on portfolio selection. This standard model leads to a quadratic programming problem. It is a challenging problem because the event tree corresponding to a multi-stage problem is usually very large (it grows exponentially with the number of stages). Various extensions to the standard model which do not change the underlying structure were given in [6, 8].

6 Numerical Results

In this section we present computational results of our approach.

We have compared the performance of OOPS with that of the commercial code CPLEX 9.1. Since we do not possess a parallel CPLEX license these results are from runs on a serial 3GHz Linux PC with 2GB of memory. We summarize our findings in Table 1. As can bee seen OOPS needs consistently less memory than CPLEX. CPLEX actually fails to solve problem C70 due to running out of memory (OoM). In this case we give an estimate solution time based on the number of flops reported from its symbolic Cholesky factorization. The smallest problem C33 is solved slightly faster by CPLEX, while for larger problems OOPS becomes much more efficient than CPLEX. We demonstrate the parallel efficiency of our code on a massively parallel environment. All computations

Table 1. Comparison of OOPS with CPLEX 9.1

prob	vars	cons	f.s.d.	CPLEX 9.1		OOPS	
				time	mem	time	mem
C33	168.451	57.274	33	292	497MB	344	156MB
C50	382.801	130.153	50	1361	1.3GB	828	345MB
C70	745.651	253.522	70	(5254)	OoM	1627	664MB

were performed on the BlueGene (BlueSky) service at Edinburgh Parallel Computing Centre (EPCC). This machine has 1024 nodes each of which comprises 2 IBM-PowerPC-440 processors running at 700Mhz and 512MB of RAM. In our experiments we run the machine in co-processor mode, that is the two processors on each node are split into a computing and a communicating unit, with all the memory available to the computing unit.

On the BlueGene service the memory is local to each node. A problem that just solves on the available memory on n processors is therefore likely to run out of memory on $n/2$ processors. To circumvent this difficulty we present our results in two series of problems. The first comprises the biggest problem we have been able to solve on BlueGene: an ALM problem with 6 stages arranged in an event tree of dimension 128x24x16x10x5x4 resulting in a total of 12.831.873 scenarios considered. This problem has just over 500 million variables. When solved on 1024 nodes, each node works with a decision tree of dimension 3x16x10x5x4 or 12532 scenarios, leading to a sparse system matrix of size 175.448×488.748. We have solved variations of this problem A16 through A1024 on $16 - 1024$ nodes where each node works with the same sub-tree as for the big problem, but the total problem has fewer first stage decisions (f.s.d.).

For completeness we have also included the details of our largest problem in the series solved at all (B1280). This example has a slightly larger event tree and almost twice as many variables per scenario as A1024. Due to larger memory requirements it could not be solved on BlueGene, but was solved on the 1600 1.7GHz-processor HPCx service instead. Due to limited allocation of computing resources on this system we are unable to provide further details for this problem. Problem sizes for this series are summarized in Table 2. Columns $\mathrm{nz}(A), \mathrm{nz}(L)$ are the numbers of nonzeros in the system matrix and the implicit inverse of the augmented system, respectively. Column *peak Mem* is the peak Memory used per node by our implementation. As can be seen the number of nonzeros in the implicit inverse grows linearly in the problem size and hence the memory per node stays roughly the same. The memory increase for higher number of processors is due to the local $\mathcal{O}(nodes^2)$ memory requirements of communication routing tables.

Table 3 gives run times for the first 20 iterations of each problem. Due to the changing topology of the scenario tree between problems our IPM takes dif-

Table 2. Problem Dimensions

Prob	f.s.d.	constraints	variables	nz(A)	nz(L)	peak Mem
A16	2	2.806.987	7.819.462	15.638.922	118.774.704	260MB
A32	4	5.613.959	15.638.884	31.277.766	237.549.408	260MB
A64	8	11.227.903	31.277.728	62.555.454	475.098.816	264MB
A128	16	22.455.791	62.555.416	125.110.830	950.197.632	264MB
A256	32	44.911.567	125.110.792	250.221.582	1.900.395.264	268MB
A512	64	89.823.133	250.221.582	500.443.086	3.800.790.528	276MB
A1024	128	179.646.223	500.443.048	1.000.886.094	7.601.581.056	292MB
B1280	128	352.875.799	1.010.507.968	2.021.015.944	18.869.419.008	661MB

Table 3. Solution Statistics and breakdown by parts of algorithm

Prob	nodes	time(20iters)	peak mem/node	generation	communication	rest
A16	16	1815	260MB	6	26	1783
A32	32	1845	260MB	12	51	1782
A64	64	1911	264MB	23	102	1786
A128	128	2050	264MB	45	206	1799
A256	256	2289	268MB	89	416	1784
A512	512	2797	276MB	178	825	1794
A1024	1024	3818	294MB	361	1666	1791
B1280	1280	1139 (HPCx)	661MB	-	-	-

Table 4. Second series of results

nodes	peak Mem	time	Comm	Cholesky	Solves	MatVectProd
16	426MB	2587 (1.00)	24	1484 (1.00)	956 (1.00)	28.8 (1.00)
32	232MB	1303 (0.99)	13	743 (1.00)	485 (0.98)	18.0 (0.80)
64	132MB	688 (0.94)	6	377 (0.98)	270 (0.88)	13.0 (0.55)
128	84MB	348 (0.93)	3	187 (0.99)	139 (0.86)	9.0 (0.40)
256	56MB	179 (0.90)	3	93 (0.99)	73 (0.82)	5.8 (0.31)
512	46MB	94 (0.86)	2	47 (0.98)	39 (0.76)	3.9 (0.23)

ferent numbers of iterations to reach optimality. We have therefore truncated our benchmark runs after 20 iterations. We have also ensured that the same numbers of centrality correctors were used in each runs, so that results are comparable. The largest problems A1024 and B1280 are solved to optimality in 45 and 53 iterations respectively. To interpret the results note that in the setup of this series the calculations done on each processor for (7, 8) are the same for all the problems. Computation time should therefore not increase with problem size. Indeed this is demonstrated by the results where the only time increase is due to (not entirely parallelisable) problem generation and parallel communications.

Our second series of experiments comprises a 64x24x16x10 decision tree problem, resulting in a total of 271.936 scenarios and a system matrix of size $3.807.119 \times 10.605.544$. This problem is solvable within the available memory on 16 nodes, while the tree architecture should enable efficient parallelisation to up to 512 nodes. As can be seen in Table 4 the algorithm parallelises well, reaching a parallel efficiency of 0.86 on 512 compared with 16 processors. If the time spent by the algorithm is broken down, we see that communications and problem generation (not reported because it was below 1s) are less of an issue than for the first series due to smaller problem size. The factorization parallelises virtually perfectly (the only non-parallel bit, the factorization of S being negligible). The backsolves (8) parallelise fairly well, while the worst efficiency is obtained from matrix vector products (mainly needed to obtain primal-dual residuals), but these do not contribute much to the overall performance.

7 Conclusions

We have demonstrated in this paper that block-structure of matrices can be exploited by an interior point algorithm. IPMs work with indefinite systems which can be transformed to quasi-definite ones by adding regularization terms. After this transformation an arbitrary symmetric reordering of the indefinite matrix can be used and a symmetric decomposition can be computed which does not need 2×2 pivots be used. Hence full advantage of the block-structure in the matrix can be taken and by exploitation of block-operations a high degree of parallelism can be achieved. Indeed, we have demonstrated that a modern implementation of interior point method run on massively parallel computer displays good parallel efficiency. Eventually, this allowed us to solve optimization problems of dimensions reaching one billion variables.

Acknowledgements

We are grateful to the EPCC for allowing us to use the BlueGene service and to Dr Joachim Hein in particular for his help in running OOPS on this machine.

References

1. Karmarkar, N.K.: A new polynomial–time algorithm for linear programming. Combinatorica **4** (1984) 373–395
2. Wright, S.J.: Primal-Dual Interior-Point Methods. SIAM, Philadelphia (1997)
3. Andersen, E.D., Gondzio, J., Mészáros, C., Xu, X.: Implementation of interior point methods for large scale linear programming. In Terlaky, T., ed.: Interior Point Methods in Mathematical Programming. Kluwer Acad Pub (1996) 189–252
4. Altman, A., Gondzio, J.: Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization. Optimization Methods and Software **11-12** (1999) 275–302
5. Gondzio, J., Sarkissian, R.: Parallel interior point solver for structured linear programs. Mathematical Programming **96**(3) (2003) 561–584
6. Gondzio, J., Grothey, A.: Parallel interior point solver for structured quadratic programs: Application to financial planning problems. Technical Report MS-03-001, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2003) Accepted for publication in *Annals of Operations Research*.
7. Gondzio, J., Grothey, A.: Exploiting structure in parallel implementation of interior point methods for optimization. Technical Report MS-04-004, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2004)
8. Gondzio, J., Grothey, A.: Solving nonlinear portfolio optimization problems with the primal-dual interior point method. Technical Report MS-04-001, School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK (2004) Accepted for publication in *European Journal of Operational Research*.
9. Ziemba, W.T., Mulvey, J.M.: Worldwide Asset and Liability Modeling. Publications of the Newton Institute. Cambridge University Press, Cambridge (1998)
10. Arioli, M., Duff, I.S., de Rijk, P.P.M.: On the augmented system approach to sparse least-squares problems. Numerische Mathematik **55** (1989) 667–684

11. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct methods for sparse matrices. Oxford University Press, New York (1987)
12. Vanderbei, R.J.: Symmetric quasidefinite matrices. SIAM Journal on Optimization **5** (1995) 100–113
13. Konno, H., Shirakawa, H., Yamazaki, H.: A mean-absolute deviation-skewness portfolio optimization model. Annals of Operational Research **45** (1993) 205–220
14. Markowitz, H.M.: Portfolio selection. Journal of Finance (1952) 77–91
15. Steinbach, M.: Markowitz revisited: Mean variance models in financial portfolio analysis. SIAM Review **43**(1) (2001) 31–85

FPGA Implementation of the Conjugate Gradient Method

Oleg Maslennikow[1], Volodymyr Lepekha[2], and Anatoli Sergyienko[2]

[1] Technical University of Koszalin, ul. Sniadeckich 2,
75-453 Koszalin, Poland
oleg@ie.tu.koszalin.pl
[2] National Technical University of Ukraine, pr.Peremogy 37,
03056 Kiev, Ukraine

Abstract. The rational fraction number system is proposed to solve the algebraic problems in FPGA devices. The fraction number consists of the n-bit integer numerator and the n -bit integer denominator, and can represent numbers with $2n$ bit mantissa. Experimental linear equation system solver was developed in FPGA device, which implements the recursive conjugate gradient method. Its hardware arithmetic unit can calculate addition, multiplication, and division of fraction numbers with n=35 in a pipelined mode. The proposed unit operates with the band matrices with the dimensions up to 3500.

1 Introduction

Field programmable gate array (FPGA) is considered to be an excellent computational raw for hardwired applications in digital signal processing (DSP), communications, control, multimedia data computing, etc. Modern FPGA devices provide millions configurable gates, and millions bits of built in memories, which can operate at the frequencies up to hundreds of MHz. FPGA platforms, which intended for DSP applications, provide tenths and hundreds of technological areas (islands), each of them has hardware multiplier and long product accumulator. As a result, such FPGA calculator has the peak throughput more than 100 billion operations per second.

Linear algebra problem solving becomes the important part of the modern DSP applications, such as adaptive filtering, curve interpolation, system parameter estimation, signal back propagation problem solving, rigid body dynamic modeling, image improvement and others. The solving of such problems demands high precision calculations. Therefore it is usually implemented using single and double precision floating point numbers. That is why linear algebra problems are usually solved in PC and floating point DSP microprocessors. Such calculations are realized in DSP microprocessors with fixed point arithmetic units (AU) very rare, mainly for solve very small problems (matrix dimensions are usually not higher than 10).

Modern high volume FPGAs give the opportunity to build the highly pipelined floating point AUs with double precision, such as described in [1], [2].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 526–533, 2006.

However, the disadvantages of such AUs are comparatively high hardware volume and pipelining delays. Therefore such FPGA processors hardly compete with the widely used floating point microprocessors.

FPGAs can provide the very precise fixed point number representation up to hundreds of bits. But till now the efforts to solve the linear algebra problems in FPGA are very rare. Really, long word adders can add and subtract large integers very quickly in them. Their high speed can be supported by pipelining. But hardware multipliers for such long words occupy much of chip area. For example, 64 to 64 bit multiplier is built from sixteen 16 to 16 bit hardware multiply units.

In the usual DSP algorithms the division operation is very rare. On the contrary, in the linear algebra algorithms the division is frequently used operation. Moreover, this operation is the source of large calculation errors. The hardware dividers are more complex than multipliers in modern FPGA, because such an n-bit divider consists of n adder-subtractor stages. Therefore, some approaches were used, which are based on the division free algorithms. As the example, in [3], the parallel device for QR-decomposition algorithm, which is based on Givens rotations, was implemented in FPGA.

Among many linear algebra methods and algorithms the conjugation gradient method is famous due its features, for example, minimum computational complexity (operation amount) for the sparse and band matrix solving and full convergence to the exact solution for less than n iterations (where n is the matrix dimension). Besides, this method uses mostly the convolution operation, vector multiply and add, like to most DSP algorithms. It needs less than $2n$ division operations and none operation like square rooting. But the most disadvantage of this method consists in that, that its termination is guaranteed only if all calculations are implemented without errors. Therefore this method is used rarely, because the floating point operations do not provide the needed precision [4]. If the precision problem will be solved, than this method would be very useful for many DSP applications.

In this paper, the rational fraction number system is proposed to implement the conjugate gradient method in FPGAs. Such system was already used in the configurable DSP processor, represented in [5], where the example of the Toeplitz matrix problem solving was shown. Then the FPGA-based processor is described which solves linear equation systems with sparse band matrices. The behavioral model description of this processor is represented, which has provided the dependency search between the problem dimension and needed data bit widths. The processor was configured in FPGA, and showed its high effectiveness.

2 Fraction Number Calculations

Fraction number is the numerical object, which consists of integer numerator and integer denominator. Its name proves that such fraction represents any rational number. Rational numbers are the real numbers, which are derived, for example, as a results of linear equation solutions, or integer polynomial divisions.

Rational fraction a/b has the feature, that it can approximate the given irrational or transcendental number x. If the fraction a/b is less than x, and the fraction c/d is higher than x, then the fraction $(a+c)/(b+d)$, named medianta, is nearest to x than these fractions. Therefore, if a set of mediantes is built, then we can to approximate the number x with any precision.

If the noninteger number x is represented by $2n$ digits with the error eps_1, then it can be represented by the fraction a/b with the error $eps_2 = eps_1$, and the numbers a and b have no more than n digits in their representation [6]. The fraction number representation has a set of advantages. Firstly, any binary fraction is depended on the binary data representation, and not exactly represents the real number. The floating point number in binary representation is equal to the fraction, which denominator is the power of two, and it is not equal to the respective decimal fraction because it has the denominator which is equal to power of ten. For example, the number $1/9 = 1/1001_2$ is the exact fraction in any numeric system, and can be represented with a error as the decimal fraction 0.1111_{10} or binary fraction 0.11100011100011_2. Secondly, the rational fractions help to find the irrational or transcendental number approximation with the given precision. Many elementary functions are effectively calculated by proper rational approximation formulas. Many constants and constant tables are effectively stored as rational numbers. And thirdly, rational fractions provide comparatively simple set of arithmetical operations. The multiplication a/b to c/d and division of them are equal to $ac/(bd)$, and $bd/(ac)$, respectively. Note, that the division of the numerator to the denominator is not calculated. Addition of them is equal to $(ad+bc)/(bd)$. For comparison of two numbers it is enough to calculate $ad-bc$. Moreover, in the rational fraction number system, it is necessary to take into account, that the numerator and denominator bit number is more than two times less than the bit number of integers, which provide the equal precision. Therefore, the hardware complexity of the fraction adder is near the complexity of the integer multiplier with the same precision, and the fraction multiplier complexity is two times less than the integer multiplier complexity.

In seventies, the main hardware implemented operation in mini - and microcomputers was addition. The floating point operations were implemented as subprograms which performed a lot of clock cycles. To speed up the calculations, in that time, the rational fractions were proposed to substitute the floating point numbers. The main disadvantage of rational fractions is that the bit number increases dramatically when operations are implemented precisely. Therefore, to eliminate of this disadvantage, the division of numerator and denominator to their greatest common divisor was made, as in the rational fraction processor, which was proposed in [7]. But when the floating point coprocessors became widely used, the fractional number processors became out of sight.

Then the rational fractions were built in many mathematical CAD tools like Maple, which are implemented in PC. Such fractions are widely used for calculations with unlimited precision, for solving modern cryptographic problems and others. Therefore, such languages as PERL and Java are supported by packages providing unlimited precision calculations. For this purpose in [8] a new standard

of data representation is proposed, named composite dates. These dates among integers and floating point numbers include rational fractions as well.

To solve many mathematical problems the high precision AU are needed. For example, the linear equation systems with sparse and band matrices is effectively solved by the conjugate gradient method. But the only disadvantage of this method consists in that that its convergence is assured, when all the calculations are made precisely [4]. Therefore this method could not be implemented when the single precision floating point is used. Note, that all floating point DSP microprocessors have the built in single precision floating point AU, and could not calculate the double precision floating point numbers effectively. In this situation the rational fraction calculations can have the high effectiveness. Below the rational fraction effectiveness for the conjugate gradient method implementation is shown.

3 Conjugate Gradient Method Modeling

To prove the rational fraction number effectiveness, the linear equation solving by the conjugate gradient method was modeled using VHDL simulator. Firstly, the package FractLib.VHD was designed, in which the type of fraction FractV was declared. The object of this type consists of two binary vectors of the length m. In this package, the functions of addition, subtraction, multiplication and division of fractions are described, which overload the respective operations of the VHDL language. The type ArrayFR1 represents the vector of fractions, the constant NIL represents the zeroed fraction, the function FractReal translates the fraction into the real number.

Then the VHDL program was designed, which loads the initial dates and solves the linear equation system. The diagonal matrix $\mathbf{A}$ of the system is symmetric, positively defined one, and is represented by the arrays $a0$, $a1$, $a2$ of its diagonals. The left column of the linear system, and unknowns are represented by the vectors b, and x. The multiplication of the matrix $\mathbf{A}$ to the column p is implemented in the procedure MATRxVECT. The conjugate gradient method is implemented in the following process.

```
process
     variable k:natural:=0;
     variable x,r,p,w:ArrayFR1(1 to n);
     variable pap,eps1,alpha,beta:FractV;
begin
     wait for 1 ns;
     xf:=(others=>NIL); r:=b; epsi:=NIL;
     for i in r' range loop eps1:=eps1+r(i)*r(i); end loop;
     loop
        k:=k+1;
        if k=1 then p:=r;
          else
```

```
            beta:=eps1/eps2;
            for i in p' range loop p(i):=r(i)+beta*p(i); end loop;
        end if;
        MATRxVECT(a0,a1,a2,p,w);
        pap:=NIL;
        for i in p' range loop pap:=pap+p(i)*w(i); end loop;
        alpha:=eps1/pap; eps2<=eps1; eps1:=NIL;
        for i in x' range loop
           x(i):=x(i)+alpha*p(i);  r(i):=r(i)-alpha*w(i);
           eps1:=eps1+r(i)*r(i) ; x(i)<=FractReal(x(i));
        end loop;
        sqe<=(SQRT(FractReal(eps1)/real(n)));
        wait on clk;
        exit when sqe<1.0e-4;
     end loop;
     report ''End of calculation'' severity failure;
end process;
```

This process is similar to the algorithm which is represented in [4]. All the dates are represented by m bit fractions, and the resulting vector x is the vector of real numbers. The array $\mathbf{A}$ is constant one, and the array b is randomized one. When the process is running, for the first nanosecond the input dates are initialized, then on the each clock edge - the one iteration of the algorithm is performed. The calculations are stopped, when the quadratic mean error *sqe* is less than the given threshold. To control the results the similar process is running, but with the double precision real dates.

The result of the modeling is the dependence between the fraction bit number m and the maximum array length n when the convergention process is stable. The derived dependence is shown in the fig.1. Analysis of this dependence shows the following rule of thumb: each data width increase to 2 digits provides the twofold increase of the maximum problem dimension. Extrapolating of the dependence line shows that the data width 52 and 64 can provide the solving the problem of dimension $n = 1$ mln., and 60 mln. respectively. The given threshold

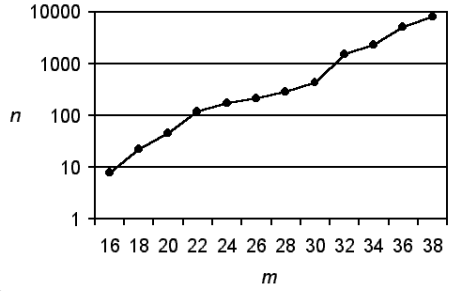


Fig. 1. Dependence between the fraction data width (m) and the maximum array length (n)

of the number *sqe* provides 4-5 true decimal digits of the result, which is enough for most of DSP applications.

4 FPGA-Based Processor for Realization of the Conjugate Gradient Method

To prove the effectiveness of the conjugate gradient method calculations using rational fractions the FPGA - based processor was designed. The processor consists of pipelined AU, memory blocks and control unit, which generates proper address sequences. The algorithm calculations are based on vector multiplication and addition of weighted vectors. Therefore, the basic operation is multiplication and addition of data streams: $\mathbf{P} = \mathbf{AX} + \mathbf{Y}$. The designed AU structure is shown on the fig.2. Here the indexes n and d sign the numerator and denominator of the fraction. The processor is implemented in the Xilinx Virtex2-Pro XC2VP4 device, which has built-in 18 bit width multiply units and 18 kilobit dual port RAMs. The fraction bit width was selected, which is equal to 35. This width provides the maximum problem size 3500. But the memory size of the selected FPGA device provides maximum vector length 1024. The samples in the matrix **A** are represented by 18 bit integers. When this matrix is multiplied, then these integers are expanded to full 35 bits of numerators and 35 bits of denominators. In such a manner the needed memory volume is minimized.

Each multiplier unit (MPU) consists of four multiplication units and three adder stages. The normalizer shifts left both numerator and denominator of operation result to the equal bit number to prevent of significant bit disappear

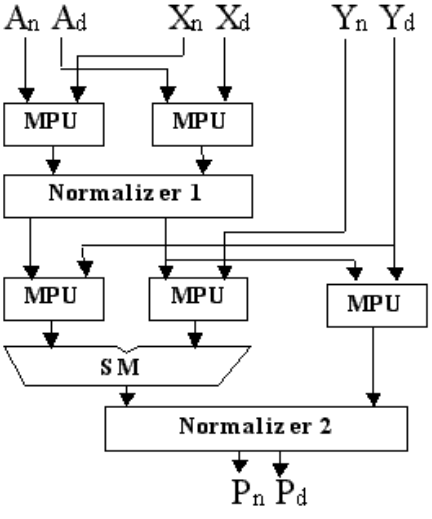


Fig. 2. Internal structure of the proposed AU

Table 1. Comparison of the designed AU with double precision floating point AUs

AU parameter	Proposed AU	AU in [1]	AU in [2]*
Hardware volume: slices	1005	4625	2825
multiplier units	20	9	9
Pipeline stages	9	34	13
Maximum clock frequency	138	120	140

* division is not implemented

after multiplication. First and second normalizer shift the dates up to 7 and 15 bits respectively. To calculate the division, the operands **A** and **X** are substituted to each other and operand **Y** is equal to zero.

In the table 1 the performance of designed AU is represented and is compared with the double precision floating point AUs, which are implemented in similar FPGA devices.

The project comparing shows that the proposed AU has high throughput and minimized configurable hardware volume which is in 2.8 - 4.6 times less than in AUs for similar purpose.

To implement the algorithm, usually less than n iterations are needed, each of them consists of $l * n$ cycles of matrix multiplication, $5n$ cycles of the vector multiplication and addition, and 2 divisions (not to take into account the pipeline loading and flushing). The solving of the equation system with the matrix **A** of dimensions n=1024 and the band width l=5 lasts about 77 milliseconds. The approximate throughput of this processor is equal to 270 Mflops.

5 Future Work

When configuring the designed processor in new Virtex4 FPGA devices, its speed will be increased approximately in two times, and the hardware volume will be decreased dramatically because such device supports the 35 bit multiplication and product accumulation on the structure level.

The above throughput can be also increased in the parallel system consisting of q designed processors (processor nodes). Each processor node in the such system calculates the $1/q$ - th part of the algorithm using the band mining technique. Each of them can store the whole matrix **A** to minimize the interprocessor communications. One of the node gathers the intermediate results, and sends the variables $eps1$, $alpha$ and $beta$ to each processor node. They are the only interprocessor communications, and they could not spend much time. Therefore the speedup of the parallel processor is approximated by q.

One modern FPGA chip, such as Xilinx XC4VSX55 can contain up to $q = 25$ processor nodes. And such system can provide up to 15000 Mflops. Note, that this device can contain only 8 nodes, described in [2] due to their higher hardware complexity.

6 Conclusions

The rational fraction number system has the advantages that it provides higher precision than integers do, and is simpler in its FPGA implementation than the floating number system.

The most advantages the rational fractions get in the modern FPGA implementation due to small hardware volume, high throughput, possibility to regulate the precision by selecting the data width. The VHDL modeling showed the possibility of use such data representation in solving linear equations by the conjugate gradient method, and showed the dependency between the data width and the maximum problem dimensions.

The experimental project of the linear equation solver showed its high throughput and small hardware volume in comparison with the processor based on the floating point AU. The system of q such processor in a single FPGA device can increase the throughput in q times, where q can be equal to 25 for modern FPGA devices. Besides, the rational fraction calculations can get profit, when the algorithms are implemented in fixed point DSP microprocessors, because they are much simpler than floating point calculations and provide the needed precision for many DSP applications.

Acknowledgement

This work was supported in part by the Polish Ministry of Science and Information Society Technologies under grant 3T11B 05926.

References

1. Underwood, K.D., Hemmert, K.S.: Closing the Gap: CPU and FPGA Trends in sustained Floating Point BLAS Performance. Proc. IEEE Symp. Field Programmable Custom Computing Machines, FCCM-2004, (2004).
2. Dou, Y., Vassiliadis, S., Kuzmanov, G.K, Gaydadjiev, G.N.:64-bit Floating point FPGA Matrix Multiplication. ACM/SIGDA 13-th Int. Symp. on Field Programmable Gate Arrays, Feb., 2005, FPGA-2005, (2005), 86–95.
3. Sergyienko, A., Maslennikow, O.: Implementation of Givens QR Decomposition in FPGA. Springer, LNCS, Vol.2328, (2002), 453–459.
4. Golub, G.G., Van Loan, C.F.: Matrix Computations. J.Hopkins Univ. Press. 2-d Ed. (1989), 642p.
5. Maslennikow,O., Shevtshenko,Ju., Sergyienko, A.: Configurable Microprocessor Array for DSP applications. Springer, LNCS, Vol. 3019, (2004), 36–41.
6. Hintchin A.Y. Chained Fractions.: Moscow, Nauka, 3-d Ed., (1978), 112p, (in Russian).
7. Irvin M.J., Smith D.R. A rational arithmetic processor.: Proc. 5-th Symp. Comput. Arithmetic, (1981), 241–244.
8. Holmes W.N. Composite Arithmetic: Proposal for a new Standard. Computer: March, N3, (1997), 65–72.

A Parallel Preconditioning for the Nonlinear Stokes Problem

Paweł J. Matuszyk and Krzysztof Boryczko

AGH – University of Science and Technology, Kraków, Poland
{pjm, boryczko}@agh.edu.pl

Abstract. We discuss a parallel preconditioner for the algebraic systems of equations arising from Newton-Raphson linearization and finite element (FE) discretization of stabilized formulation for the highly nonlinear Stokes equations. We compare SSOR/Jacobi and FSAI(1)/Jacobi preconditioners for the parallel MINRES accelerator implemented using C++/OpenMP. Results are presented for SGI Altix 3700 and IBM Power4 machines. As an example a simulation of the compression test of the rigid-viscoplastic material is shown.

1 Introduction

Steady-state metal forming process, as considered in the work compression test, can be effectively modeled as a steady flow of viscoplastic material. Assuming the incompressibility of metallic material, the Galerkin method results in mixed FE formulation where the velocity $\mathbf{u}$ and the pressure p fields are approximated separately. Such a method requires fulfilling of the Brezzi-Babuška condition for the convergence and unique solvability. This condition imposes severe restrictions on the choice of the combinations of velocity and pressure spaces, which, in consequence, prohibits the use of convenient, equal-order finite elements. Therefore, the stabilized FE method is applied to avoid limitations arising from the use of classical Galerkin method.

The main goal of the paper is developping an efficient method for the sequential and parallel solution of the linear equations system arising from linearization and discretization of stabilized formulation of the nonlinear Stokes problem.

2 Physical and Computational Model

2.1 Strong Form and Constitutive Model

Let the deformed material is represented by the two or three-dimensional domain Ω with a boundary $\Gamma = \partial\Omega$, where $\Gamma = \Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}}$. Strong form describing equilibrium conditions for the incompressible material, in domain Ω, with proper boundary conditions on Γ, has the form:

$$\begin{cases} \nabla \cdot \boldsymbol{\sigma} = \nabla \cdot \mathbf{s} - \nabla p = 0 \quad \text{on } \Omega & \mathbf{n} \cdot \boldsymbol{\sigma} = \mathbf{t} \quad \text{on } \Gamma_{\mathrm{N}} \\ \nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega & \mathbf{u} = \mathbf{u}_{\mathrm{t}} \quad \text{on } \Gamma_{\mathrm{D}}, \end{cases} \tag{1}$$

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 534–541, 2006.

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor, $\mathbf{s}$ the deviatoric stress of $\boldsymbol{\sigma}$, $\mathbf{u}$ is the velocity of the material, $p = -\frac{1}{3}\sigma_{kk}$ is hydrostatic pressure in the material, $\mathbf{u}_\mathrm{t}$ the velocity specified on Γ_D and $\mathbf{t}$ the traction specified on Γ_N.

Due to the large deformations during a compression test, elastic strains are neglected and the rigid-viscoplastic constitutive model is applied: $\mathbf{s} = 2\eta\dot{\boldsymbol{\varepsilon}}$ where $\dot{\boldsymbol{\varepsilon}} = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right)$, $\dot{\boldsymbol{\varepsilon}}$ is strain rate tensor and an effective viscosity η is described by the Norton-Hoff law [8]:

$$\eta(\dot{\varepsilon}_\mathrm{i}) = K\left(\sqrt{3}\dot{\varepsilon}_\mathrm{i}\right)^{m-1} \qquad \dot{\varepsilon}_\mathrm{i} = \sqrt{\tfrac{2}{3}\,\dot{\boldsymbol{\varepsilon}} : \dot{\boldsymbol{\varepsilon}}} \qquad \sigma_\mathrm{i} = K\sqrt{3} \qquad m \in [0,1], \tag{2}$$

where $\dot{\varepsilon}_\mathrm{i}$ is an intensity of strain rate and σ_i is stress intensity. Hence one has obtained the Stokes problem with nonlinear velocity-dependent viscosity.

2.2 Friction Model

Friction on the surface between the material and a tool is described by velocity-dependent model of Chen and Kobayashi [1]. It defines a friction stress as:

$$\mathbf{t} = -mK\left[\tfrac{2}{\pi}\arctan\left(\tfrac{\mathbf{u}_\mathrm{s}}{\alpha}\right)\right] \qquad \text{where constant } \alpha \approx \frac{\|\mathbf{u}_\mathrm{s}\|}{10^4}, \tag{3}$$

where $\mathbf{u}_\mathrm{s}$ is relative velocity between the material and the tool and m is a friction factor. Such defined friction is a nonlinear function with respect to the velocity and has significant influence on the convergence of the procedure of linearization.

2.3 Stabilized Weak Form

The Petrov-Galerkin method for viscoplastic flow, that leads to the SUPG formulation [3], was presented for the first time in [4]. Applying the SUPG method for the strong form (1) results in the following stabilized weak form:
Find $(\mathbf{u}, p) \in \mathbf{V} \times Q$ *such that*:

$$\begin{cases} \displaystyle\int_\Omega \mathbf{s} : \dot{\boldsymbol{\varepsilon}}\,\mathrm{d}\Omega - \int_\Omega p\,\nabla\cdot\mathbf{v}\,\mathrm{d}\Omega = \int_{\Gamma_\mathrm{N}} \mathbf{t}\cdot\mathbf{v}\,\mathrm{d}\Gamma_\mathrm{N} & \forall\mathbf{v} \in \mathbf{V}_0 \\ \displaystyle\int_\Omega q\,\nabla\cdot\mathbf{u}\,\mathrm{d}\Omega + \sum_{K\in\mathcal{T}_h}\int_K \tau\nabla q\cdot\nabla p\,\mathrm{d}K = \sum_{K\in\mathcal{T}_h}\int_K \tau\nabla q\cdot(\nabla\cdot\mathbf{s})\,\mathrm{d}K & \forall q \in Q \end{cases} \tag{4}$$

where function spaces are defined as: $\mathbf{V} = H^1(\Omega)^d$ ($d = 2$ or 3), $Q = H^1(\Omega)$, $\mathbf{V}_0 = \{\mathbf{v} \in \mathbf{V} : \mathbf{v}|_{\Gamma_\mathrm{D}} = \mathbf{0}\}$ and $\Omega = \bigcup_{\mathcal{T}_h} K$ is a triangulation of Ω. According to [4], the stabilization coefficient τ is taken as $\tau = \frac{10^{-1}h_\mathrm{K}^2}{2\eta}$ where $h_\mathrm{K} = \mathrm{diam}\,(K)$.

Following [4], a local reconstruction of deviatoric stress tensor in element nodes is applied to calculate the right-hand side of the 2nd equation in (4) for higher order elements. The deviatoric stress tensor components are reconstructed to be a continuous variables, in a local sense, using an L^2 projection and then can be expressed in terms of nodal quantities and FE shape functions.

2.4 Linearization and Discretization

The system (4) is nonlinear with respect to the velocity $\mathbf{u}$. For simplicity, the last right-hand side term of the 2nd equation is not linearized and it is treated as an additional force term acting on the system. Such simplifications allows to avoid computing of complicated derivatives. Similarly, in contrast to work [4] the stabilization coefficient τ is considered as a constant which results in symmetric bilinear form and implies symmetric matrix of the linear system. Thus, one can use efficient iterative methods based on short three-term recurrences.

The Newton-Raphson method is used for linearization of the system (4). After linearization, discretization and symmetrization, the linear system is obtained:

$$\mathcal{A} \cdot \begin{bmatrix} \delta\mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{S} \end{bmatrix} \cdot \begin{bmatrix} \delta\mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}. \tag{5}$$

Submatrices and vectors appearing in the system have the following forms:

$$\mathbf{A} = \frac{\partial}{\partial \mathbf{u}} \left(\int_\Omega \mathbf{s} : \dot{\boldsymbol{\varepsilon}} \, \mathrm{d}\Omega - \int_{\Gamma_\mathrm{N}} \mathbf{t} \cdot \mathbf{v} \, \mathrm{d}\Gamma_\mathrm{N} \right) \qquad \mathbf{B} = -\frac{\partial}{\partial \mathbf{u}} \int_\Omega q \nabla \cdot \mathbf{u} \, \mathrm{d}\Omega$$

$$\mathbf{S} = -\sum_{K \in \mathcal{T}_h} \int_K \tau \nabla q \cdot \nabla p \, \mathrm{d}K \qquad \mathbf{f} = \int_{\Gamma_\mathrm{N}} \mathbf{t} \cdot \mathbf{v} \, \mathrm{d}\Gamma_\mathrm{N} - \int_\Omega \mathbf{s} : \dot{\boldsymbol{\varepsilon}} \, \mathrm{d}\Omega$$

$$\mathbf{g} = \int_\Omega q \nabla \cdot \mathbf{u} \, \mathrm{d}\Omega + \sum_{K \in \mathcal{T}_h} \int_K \tau \nabla q \cdot (\nabla \cdot \mathbf{s}) \, \mathrm{d}K.$$

2.5 Improving the Convergence of the Linearization

The Newton-Raphson method is quadratically convergent to the solution if initial solution is sufficiently close to the actual one. Three different methods are used together to overcome instabilities and to accelerate the convergence:

- generation of an initial velocity field based on the specimen geometry and imposed boundary conditions [8];
- subincrementation method [8] — a new velocity field is calculated as $\mathbf{u}^{n+1} = \mathbf{u}^n + \gamma \cdot \delta\mathbf{u}^{n+1}$ where coefficient $\gamma \in [0, 1.1]$ is selected to minimize the residual norm $\|\mathbf{R}(\mathbf{x}^n + \gamma \cdot \delta\mathbf{x}^{n+1})\|$ of the solved equations system;
- progressive linearization [4] — starting from linear constitutive model, in successive linearization steps, the constitutive model is progressively made more nonlinear.

2.6 Time Step

Having calculated a new solution $(\mathbf{u}^n, p^n)$ of (4) one can perform one time step of the compression test. New coordinates $\mathbf{x}^n$ of the nodes (mesh distortion) are obtained by time integration due to one-step θ-approximation scheme:

$$\mathbf{x}^n = \mathbf{x}^o + \delta t \left[\theta \mathbf{u}^n + (1 - \theta) \mathbf{u}^o \right] \qquad \theta \in [0, 1], \tag{6}$$

where δt is a time step and index o indicates solutions from the previous step.

3 Numerical Properties of Tangent Matrix

Considered linear system (5) expresses the discrete linearized Stokes problem with the block symmetric matrix $\mathcal{A}$, where submatrix $\mathbf{A}$ is the tangent matrix for linear (with respect to $\mathbf{v}$) form $a(\mathbf{u};\mathbf{v}) = \int_\Omega \mathbf{s}(\mathbf{u}) : \dot{\boldsymbol{\varepsilon}}(\mathbf{v})\,\mathrm{d}\Omega - \int\limits_{\Gamma_\mathrm{N}} \mathbf{t}(\mathbf{u})\cdot\mathbf{v}\,\mathrm{d}\Gamma_\mathrm{N}$, $\mathbf{B}$ is a discrete divergence operator, $\mathbf{B}^T$ — a divergence's adjoint — discrete negative gradient operator and $\mathbf{S}$ is a sum of discrete negative Laplacian operators on elements level.

Matrix $\mathbf{B}$ is full-rank matrix except that the vector $p = [1,\ldots,1]^T$, representing constant hydrostatic pressure, belongs to $\ker(B)$. Author proves in [7] that the bilinear form $\frac{\partial a(\mathbf{u};\mathbf{v})}{\partial \mathbf{u}}$ is coercive on $\mathbf{V}_0$ which implies that $\mathbf{A}$ is a symmetric positive definite (SPD) matrix. Matrix $\mathbf{S}$ is a symmetric negative semidefinite ($-\mathbf{S}$ is SPD matrix).

4 Solution and Preconditioning Methods

Following the discussion in [2], we propose to apply a MINRES method with block-diagonal preconditioner of the form:

$$\mathbf{M} = \begin{bmatrix} \mathbf{P} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q} \end{bmatrix}, \tag{7}$$

where submatrices $\mathbf{P}$ and $\mathbf{Q}$ are both SPD and are the preconditioners respectively for submatrices $\mathbf{A}$ and $\mathbf{S}$. As it was proved in [2], the convergence of the preconditioned MINRES method is essentially determined by the quality of the preconditioner $\mathbf{P}$.

We decided to use the following preconditioners:

- $\mathbf{Q} = -\mu\mathbf{M}_\mathrm{J}(\mathbf{S})$, where $\mathbf{M}_\mathrm{J}$ is the Jacobi preconditioner for submatrix $\mathbf{S}$ and μ is positive scaling constant;
- $\mathbf{P} = \mathbf{M}_\mathrm{SSOR}(\mathbf{A})$, where $\mathbf{M}_\mathrm{SSOR} = (\mathbf{D}_\mathrm{A} + \omega\mathbf{L}_\mathrm{A})\mathbf{D}_\mathrm{A}^{-1}(\mathbf{D}_\mathrm{A} + \omega\mathbf{L}_\mathrm{A}^T)$ is the SSOR preconditioner for submatrix $\mathbf{A}$; submatrices $\mathbf{D}_\mathrm{A}$ and $\mathbf{L}_\mathrm{A}$ are respectively: diagonal and strictly lower part of $\mathbf{A}$;
- $\mathbf{P} = \mathbf{M}_\mathrm{FSAI}(\mathbf{A})$, where $\mathbf{M}_\mathrm{FSAI}(\mathbf{A})$ is Fast Sparse Approximate Inverse preconditioner (FSAI) for matrix $\mathbf{A}$.

Tests performed for several choices for $\mathbf{Q}$ showed that the best results were obtained for simple Jacobi preconditioner with the coefficient $\mu = 0.05$.

The first choice for the preconditioner $\mathbf{P}$ is the SSOR preconditioner. The main advantage of this preconditioner is such that it does not require any additional memory — all preconditioning operations can be performed using the stored values of matrix $\mathbf{A}$. The main disadvantage of the SSOR preconditioner is a need of solving two triangular systems for each preconditioning operation, which makes these operations very difficult to parallelize. However, in sequential computations, the SSOR preconditioner turned out to be the most effective with respect to the calculation time (for parameters $\omega \in [1.5 - 1.7]$).

The second choice for the preconditioner $\mathbf{P}$ is the FSAI preconditioner [5]. For a given SPD matrix $\mathbf{A}$, this metod gives an approximate inverse $\mathbf{H} = \mathbf{G}^T\mathbf{G}$ of matrix $\mathbf{A}$. $\mathbf{G}$ is a sparse lower triangular that approximates $\mathbf{L}^{-1}$ (where $\mathbf{A} = \mathbf{L}\mathbf{L}^T$). To construct $\mathbf{G}$ one has to prescribe a selected sparsity pattern $S_{\mathrm{L}} \subseteq \{(i,j) : 1 \leq (i < j) \leq n\}$ and then a lower triangular matrix $\mathbf{G}$ can be computed by calculating $\mathbf{G} = \mathbf{D}\hat{\mathbf{G}}$ where $\hat{\mathbf{G}}$ is the solution of:

$$(\hat{\mathbf{G}}\mathbf{A})_{ij} = \delta_{ij} \quad (i,j) \notin S_{\mathrm{L}} \qquad \text{and} \quad \mathbf{D} = \mathrm{diag}(\hat{\mathbf{G}})^{-1/2}. \tag{8}$$

The main advantages of FSAI metod in comparison to the SSOR are such that:

- preconditioning operation can be performed fully parallel (two sparse matrix-vector multiplications),
- each row of matrix $\mathbf{G}$ can be computed independently: calculation of components of $\mathbf{G}$ can be performed completely parallel (each row requires the solution of a relatively small dense SPD linear system of size equal to the number of nonzeros in that row).

On the other hand, there are two drawbacks of the preconditioner of that type:

- one has to prescribe the sparsity pattern a priori,
- there is a necessity of use of additional memory for storing matrix $\mathbf{G}$ (in the case of parallel computations — also for $\mathbf{G}^T$).

We decided to use the preconditioner FSAI(1) for which a sparsity pattern of matrix $\mathbf{G}$ corresponds to the sparsity pattern of the lower triangular part of matrix $\mathbf{A}$. Additionally, after the computation of $\mathbf{G}$, a sparsification postprocessing is performed by using dropping strategy to reduce the number of nonzeros of the preconditioner factors. It results in decreasing the computational complexity of preconditioning operation.

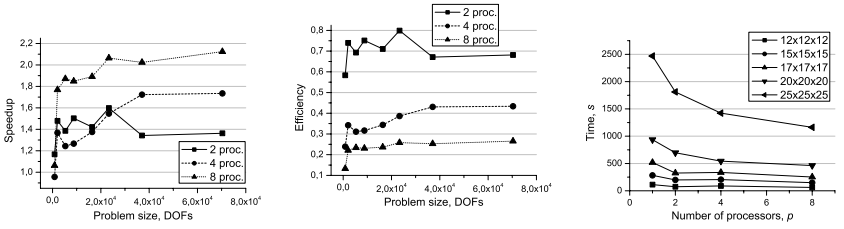
5 Implementation

2D/3D FE solver for simulation of compression tests is implemented in an object-oriented manner using C++ language [6, 7]. Continuous velocity/continuous pressure finite elements are used: P1/P1, Q1/Q1 and Q2/Q1 in 2D, and Q1/Q1 in 3D. Parallelization of the code is performed using OpenMP environment which minimizes the necessary implementation changes of the sequential code.

6 Numerical Examples

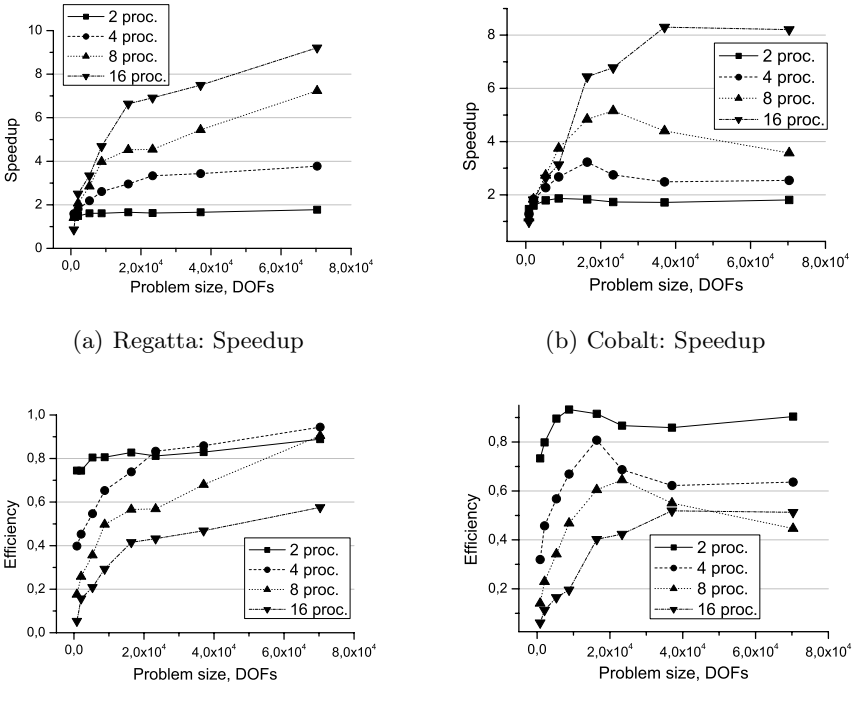
Experimental tests are performed on two supercomputers:

1. IBM Power4 p655 (Regatta) — global shared memory cluster, each node consists 4 chips (2× IBM Power4 1.5GHz + 1.44MB L2) and 32MB L3, 13 nodes connected using IBM High Performance switch, 16GB RAM;



(a) Speedup (b) Efficiency (c) Time for different sizes of meshes

Fig. 1. Results for the SSOR version of the preconditioner on the Regatta computer



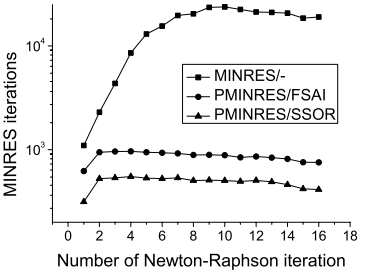
(a) Regatta: Speedup (b) Cobalt: Speedup

(c) Regatta: Efficiency (d) Cobalt: Efficiency

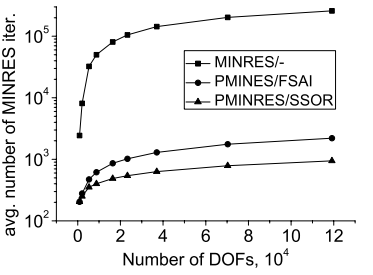
Fig. 2. Results for the FSAI(1) version of the preconditioner

2. SGI Altix 3700 (Cobalt) — ccNUMA architecture (SGI NumaFlex/SGI NumaLink, dual plane fat-tree topology), 48 Itanium 2 1.3GHz and 96GB RAM.

All the tests concern the execution of the initial two time steps of compression test of metallic cube of dimension $15 \times 15 \times 15$mm. Highly nonlinear rigid-plastic constitutive model (sensitivity coefficient in Norton-Hoff equation $m = 0$) is



(a) Number of MINRES iterations for each Newton-Raphson iteration



(b) Average number of MINRES iterations for Newton-Raphson procedure

Fig. 3. Comparison of iteration numbers for different preconditioners (Regatta, the first time step of compression)

used. Flow stress curve $\sigma_{\mathrm{i}} = 20 + 370\,\varepsilon_{\mathrm{i}}^{0.38}$MPa and the friction factor $m = 0.5$ is applied. The constant tool velocity equal 1mm/s is assumed. Structural homogenous hexahedral meshes with isotropic division (5, 7, 10, 12, 15, 17, 20 and 25 elements in each direction) are used.

Results obtained for the solver using the SSOR preconditioner with overrelaxation parameter $\omega = 1.7$ are presented in Fig. 1. Results obtained for the version with FSAI(1) preconditioner are presented in Fig. 2.

One time step of the compression needs carrying out the Newton-Raphson linearization which implies solution of several linear systems. Fig. 3(a) shows number of MINRES iterations needed for solution of these system for the first time step (mesh $15 \times 15 \times 15$, friction $m = 0.5$). Fig. 3(b) shows average number of iterations needed for solution one linear system for different sizes of problem.

7 Conclusions

- Results presented in Fig. 1 confirm that the SSOR preconditioner is very inefficient in the case of parallel calculations. Maximal speedup (equal approximately 2.1 for 8 processors) can be observed for the largest problem size. Such unacceptable low efficiency is a consequence of relatively high serial fraction coefficient which is approximately equal $0.4 - 0.45$ [7].
- Results obtained for the solver with implemented FSAI(1) preconditioner are much better (Fig. 2). The general observation is that the larger size of the problem, the higher speedup. It is a consequence of more balanced partition of work between processors. For all the test one achieves efficiency above 50%. In the case of the Regatta machine — even much higher.
- The differences observed for both computers can be explained by different architecture of the network layer connecting processors and memories and also by different strategies of the assignment of resources for particular computational tasks.

- For tests with smaller number of used processor one can observe the saturation of speedup. Such trends for test with greater number of used processors show a possibility of obtainment better efficiency and speedup for problems of larger size. This is a consequence of good scalability of the solver.
- Comparison of computing time necessary for solving the problem by solvers with different preconditioners shows the predominance of the SSOR method, that runs nearly two times faster. It is a consequence of the absence of a very time consuming stage as the building sparsity pattern and calculating coefficients of the FSAI preconditioner is. However, the superiority of the FSAI preconditioner in the case of parallel computation is clearly evident.
- In the future work we are going to modify sparsity pattern by increasing the number of nonzeros of the FSAI preconditioners to obtain a better inverse approximation of submatrix $\mathbf{A}$. We hope, it results in decreasing of the number of needed to convergence iterations of Krylov method what in consequence gives faster solver.

Acknowledgments

The work financed by Polish Committee for Scientific Research. Project number 4 T08B 009 24.

References

1. Chen, C.C., Kobayashi, S.: Rigid-Plastic-Finite-Element Analysis of Ring Compression. Appl. of Numerical Methods to Forming Processes **28** (1978) 163–174
2. Elman, H.C., Silvester, D.J., Wathen, A.J.: Iterative Methods for Problems in CFD. Technical report CS-TR-3675, Oxford Unversity Computing Laboratory (1996)
3. Hughes, T. J. R., Franca, L. P., Balestra, M.: A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuska-Brezzi condition: A stable Petrov-Galerkin formulation of the Stokes problem accomodating equal-order intrpolations. Comp. Meth. Appl. Mech. Eng. **59** (1986) 85–99
4. Maniatty, A. M., Liu, L., Klaas, O., Shephard, M. S.: Stabilized finite element method for viscoplastic flow: formulation and a simple progressive solution strategy. Comp. Meth. Appl. Mech. Eng. **190** (2001) 4609–4625
5. Kolotilina, L.Yu., Yeremin, L.Yu.: Factorized Sparse Approximate Inverse Preconditionnings I. Theory. SIAM J. Matrix Anal. Appl. **14** (1993) 45–58
6. Matuszyk, P.J.: Object-oriented realization of the FE solver for compression test. Informatyka w Technologii Materiałów **4** (2004) 127–143
7. Matuszyk, P.J.: Object-oriented realization of parallel algorithms for simulation of plastometric tests on shared-memory cluster. PhD Thesis, AGH, Kraków (2005)
8. Wagoner, R. H., Chenot, J. -L.: Metal Forming Analysis. Cambridge University Press (2001)

Optimization of Parallel FDTD Computations Based on Structural Redeployment of Macro Data Flow Nodes

Adam Smyk and Marek Tudruj

[1] Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., 02-008 Warsaw, Poland
[2] Institute of Computer Science, Polish Academy of Sciences,
21 Ordona Str., 01-237 Warsaw, Poland
{asmyk, tudruj}@pjwstk.edu.pl

Abstract. This paper shows methodology, which enables profiling macro data flow graphs (*MDFG*) that represent computation and communication patterns for the Finite Difference Time Domain (*FDTD*) problem in irregular computational areas. *MDFG* optimization is performed in three phases: simulation area partitioning with generation of initial *MDFG*, macro data nodes merging with static load balancing to obtain given number of macro nodes and communication optimization to minimize (balance) inter-node data transmissions, computational cells redeployment to take into account computational system restrictions. Efficiency of computations for several communication systems (*MPI*, *RDMA RB*, *SHMEM*) is discussed. Experimental results obtained by simulation are presented.

1 Introduction

There are many numerical problems that can be characterized by unstructured [9], coarse grain and non-deterministic communication and data patterns. In order to obtain a satisfactory execution time in case of such applications, data decomposition scheme should be specially designed and adopted to the architectural requirements. As an good example of unstructured problem, considered in this paper, we present a simulation of electromagnetic wave propagation in irregular computational area (*CA*) - Finite Difference Time Domain (*FDTD*). Such problems can be also found in many aspects of numerical algebra or VLSI layout design. In most on these problems, computational dependencies can be given by data flow graphs. In order to execute them in parallel they must be partitioned into sub-graphs assigned to processors. There are many methods that enable graphs partitioning (NP-complete problem [4]) but generally two kinds of such methods are distinguished: direct techniques [6] (mainly based on min-cut optimization) and iterative improvement techniques [6][7][8][5][2] (extension of the algorithms proposed by Kernighan-Lin and improved by Fidducia-Mattheyses (*FM*)[3]). Generally, the problem discussed in the paper concerns an extension of the *FM* algorithm that can be applied to partitioning of the macro data flow

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 542–550, 2006.

graphs representing the computations of the *FDTD* problem. In [12] we presented an optimization method that produces statically balanced macro data flow graph representing *FDTD* computations in irregular simulation area assuming negligible inter-node communication cost due to specific *RDMA* communication. In this paper, we extend the algorithm by optimizing a given *MDFG* for different characteristics of communication system with non-negligible communication cost. The algorithm optimizes total program execution time by redeployment of elementary computations among macro data flow nodes, which provides balanced communication and computation costs in the entire *FDTD* program. The paper is composed of four parts. In the first part, the formulation of the *FDTD* problem and its solution according to the macro data flow paradigm is described. The second part presents a brief description of two main operations: macro node partitioning and merging, which are used in the first part of the optimization algorithm. The next part shows an *MDFG* optimization method that transforms given macro data flow graph taking into consideration executive system restrictions. Experimental results of this optimizations are discussed in the last part of the paper.

2 FDTD Implementation Based on the Macro Data Flow Paradigm

The Finite Difference Time Domain method (FDTD) enables simulation of high frequency electromagnetic wave propagation by solving Maxwell equations (1).

$$\nabla \times H = \gamma E + \varepsilon \frac{\partial E}{\partial t},\ \nabla \times E = -\mu \frac{\partial H}{\partial t} \tag{1}$$

$$\begin{cases} \overline{H}_y^n(i,j) = \overline{H}_y^{n-1}(i,j) + RC \cdot [\overline{E}_z^{n-0.5}(i,j-1) - \overline{E}_z^{n-0.5}(i,j+1)] \\ \overline{H}_x^n(i,j) = \overline{H}_x^{n-1}(i,j) + RC \cdot [\overline{E}_z^{n-0.5}(i-1,j) - \overline{E}_z^{n-0.5}(i+1,j)] \\ \overline{E}_z^n(i,j) = CA_z(i,j) \cdot \overline{E}_z^{n-1}(i,j) + CA_z(i,j) \cdot [\overline{H}_y^{n-0.5}(i+1,j) + \\ \qquad + \overline{H}_y^{n-0.5}(i-1,j) + \overline{H}_x^{n-0.5}(i,j-1) + \overline{H}_x^{n-0.5}(i,j+1)] \end{cases} \tag{2}$$

Simulated area is a two dimensional, irregular shape, see Fig. 1. Before simulation, a whole *CA* is transformed into a physical mesh. In a two dimensional *FDTD* problem, the mesh consists of a specified number of points which contain alternately electric component *Ez* of electromagnetic field and one from two magnetic components *Hx* or *Hy* (depending on coordinates). After transformations of two Maxwell equations (1) into a differential form (2), we can formulate dependencies between *Ez*, *Hx* and *Hy* components. The value of *Ez* is dependent on values of four nearest magnetic field components (*Hx* and *Hy*). To calculate the value of magnetic component, we require two nearest electric field components. Computational process is divided into a given number of steps. Each step is divided into two substeps: first - the value of all *Ez* is computed and later - the values of *Hx* and *Hy* are computed. If the shape of computational area is regular (e.g. rectangular), the whole computational process can be easily parallelized (e.g. by stripe or block partitioning of the computational area). In the

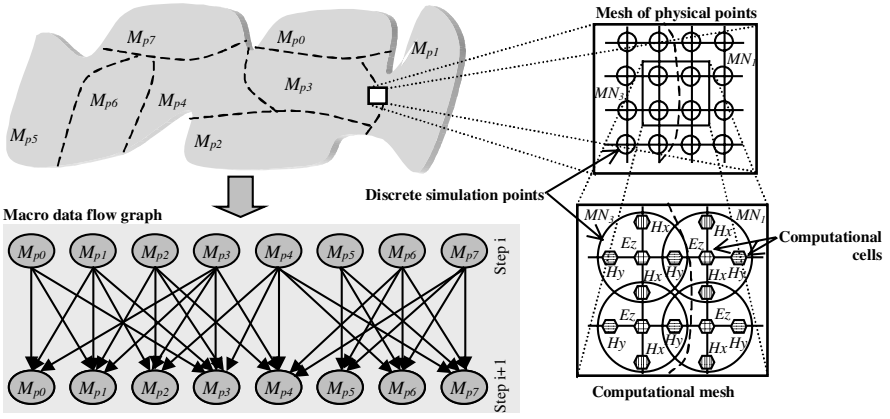


Fig. 1. Irregular computational area with its MDFG for two computational steps

case of an irregular shape of computational area such decomposition is more complicated. It is because we have to take into consideration both a proper load of all processing nodes and the minimal number of data transmissions. To solve this problem, we have implemented the *FDTD* method program according to a macro data flow paradigm. At first, we created a data flow graph of computations to show basic data dependencies. Next, we use a merging algorithm to define macro nodes (*Mpn*, where p is a processor number, a n is a simulation sub-area number). It is described in details in [12]. The computation macro data flow graph (for two computational steps) for the hypothetical simulation area is shown in Fig. 1. Each computations step in each sub-area is represented by one macro node. A macro node can be executed only if all external input data have arrived to the physical processor on which this macro node has been mapped. Data dependencies are given by edges that connect macro nodes. All edges are weighted by the amount of data, which will be sent from one macro node to another. It directly depends on the length of the boundary line between two adjacent sub-areas, which correspond to *Mpx*, and *Mpy*. Each *Mp* produces results, which will be sent to another *Mps* when its computations are finished.

3 Macro Data Flow Graph Optimization

Our algorithm enables designing optimized program macro data flow graphs for the *FDTD* method, which provide execution time reduction. It consists of three steps: simulation area partitioning, macro nodes merging and computational cells redeployment. We do not apply here any geometrical analysis of computational area - it is completely based on the data dependencies existing in the computational *FDTD* mesh. The first step enables creating set of "germs" of macro nodes (*GMN*). *GMNs* are created in the following way: we choose a given number of leader points from the mesh of physical points and next we "stick"

all the nearest physical points to the leader point. One leader point represents one macro node. We tested several methods choosing of leaders as described in [12]. It allows to obtain an initial macro data flow graph that can be used for further optimizations. The number of *GMN* is significantly bigger then available number of computational nodes (processors). For that reason, in the second step we merge chosen *GMN* to reduce the number of total *GMNs*. We also tested several merging criteria that enable both decreasing total communication time and balancing computation node. Both of these steps are described in details in [12]. After execution of these two steps we obtain statically optimized macro data flow graph that represents *FDTD* computation in given simulation area. "Static" means that graph optimization is independent and does not consider the architecture of the given executive system.

4 Load Balancing by Cell Redeployment Between Macro Nodes

In the next step of our optimization algorithm, we transform the program macro flow graph described in previous parts [Fig. 1], so as to reduce program execution time by load balancing in executive processors for given system configuration. First, a given *MDFG* should be transformed into a *MNCG* (Macro Node Communication Graph). The *MNCG* describes data dependencies between all macro nodes, which occurred in the *MDFG* graph during execution of all steps of the *FDTD* algorithm. The structure of a *MNCG* [Fig. 2a] is simpler than that of a MDFG [Fig. 1], because it represents its "concentrated" image of data dependencies. We identify in the *MNCG* the set of all cliques. A clique is a set of all macro nodes that are directly connected by edges with one, "central" macro node, see Fig. 2b. The general idea of the optimizing algorithm is based on equalizing execution time among all cliques, so as to minimize a difference between the average execution time of each step in the *MDFG* and the average execution time *AvgExTime(i)* of all macro nodes in each clique *i*. The *AvgExTime(i)* is computed as the average of sums of computation and communication costs (time) of all macro nodes belonging to clique *i*. In order to balance *AvgExTime(i)* for all cliques, we will redeploy (move) subsets of computational cells between cliques. The maximal amount of computations to be deployed in a sub-set is determined by the difference between *AvgExTime(i)* and average execution time of one step in the *MDFG*, Fig. 2b. The whole redeployment phase is governed by execution time analysis. Our optimizing algorithm takes into consideration relations between computer system architecture and the macro data flow graph that represents a computation scheme for a given wave propagation area. There are three input parameters to the algorithm: a macro data flow graph, the speed of processor node (we assumed a homogenous system) and the speed of available communication system. At first, the set of all cliques for *MDFG* is generated. For all generated cliques, a maximal clique execution time is computed and all cliques are marked as unlocked for redeployment. Next, we determine two cliques, one with the biggest and one with the smallest value of (*Bclique-CMp0*) and we

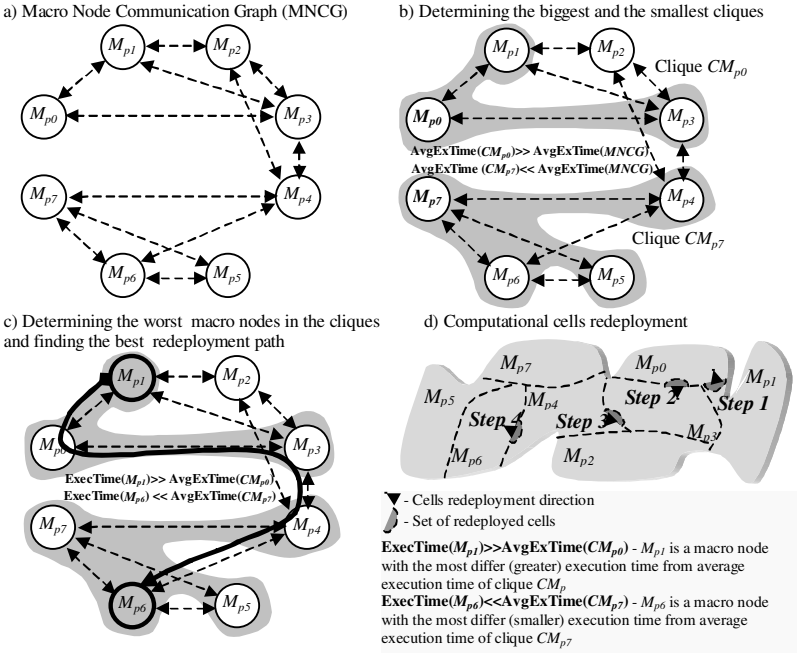


Fig. 2. Redeployment operation scheme

determine (*Sclique-CMp7*), the average execution time. For the two chosen cliques, a redeployment algorithm is performed. It consists of several steps executed in sequence. In the first step, we compute differences between the average execution time of a single step in a given *MDFG* and the execution time (the sum of computation and communication times) of each node from *Bclique* and *Sclique*. Based on these differences, we compute a total cost difference that can be moved between *Bclique* and *Sclique*. In the next step, we find a pair of macro nodes, which will be used in redeployment operation - *Mp1* and *Mp6*. Next, we extract the sets of computational cells, which will be moved between macro nodes *Mp1* and *Mp6*. The number of cells strictly depends on the maximal acceptable part of the execution cost, which can be changed to balance the execution time of *Bclique* and *Sclique*.

In the next part of the algorithm, we find the best path between chosen macro nodes. Our algorithm is a modification of the Dijkstra's Shortest Path algorithm (*DSPA*) [10]. In order to assure that the communication overhead will not be increased during the redeployment operation, we have modified a relaxation operation that is the most important element of the *DSPA*. In the standard *DSPA* algorithm, all nodes in a given input graph are weighted by the distance of one distinguished node - so-called source node (*Mp1* node in our case). The weight of a node depends on the weights of all input edges of the node and all predecessor nodes. It is updated during the relaxation operation successively executed for all nodes on the path from node *Mp1* to *Mp6*, see

Fig 2c. In our algorithm, the relaxation scheme is identical as in the standard one, but the weight of node k is determined by the following equation:

$$weight(node_k) = \underset{for\,all\,predecessors\,of\,node_i}{Min}\,(weight(node_i) + redep(node_i, node_k)) \tag{3}$$

where: $node_i$ is a predecessor of $node_k$; $weight(node_i)$ is equal 0; $redep(X, Y)$ - the number of *Ez* cells moved from node Y to node X that caused an increase of communication cost between these nodes.

The weight of the node k on the redeployment path determines the number of cells that have induced growth in communication cost up to reaching the node k in redeployment. If we have found a path between nodes *Mp1* and *Mp6* with *weight(Mp6)=0*, it means that if we use this path during redeployment operation, the communication volume will not increase. If the *0-weight* path has not been found, we know that it is impossible to redeploy a required number of cells without a growth in communication volume and we must reduce the number of moved cells. After that, the move cells operation is performed for each consecutive pair of macro nodes on the path connecting *Mp1* and *Mp6*. In this operation, all computational cells are successively moved [Fig. 2d] through all consecutive macro nodes belonging to the redeployment path. If after the move of cells *weight(Mp6)¿0*, it means that the number of cells has been reduced and redeployment between *Bclique* and *Sclique* must be repeated. After redeployment operation, we again compute a new value of the execution speedup of the *MDFG*, which corresponds to the re-balanced *MNCG*. If the speedup has increased, we can validate the last, just tested redeployment operation. Otherwise, last redeployment is cancelled and two given cliques are locked. If redeployment for two chosen cliques was not successful (the total graph execution speedup did not increase), the cliques are marked as locked and they will not be used in next optimization steps. The optimization algorithm will complete in the following cases:

1. All cliques are marked as locked - it is not possible to perform any new redeployment operation.
2. The *CliqueB* parameter equals one, where *CliqueB* is a value that determines clique balance in the *MNCG* graph, computed from the following equation:

$$CliqueB = \frac{absolute(L - S)}{total\,number\,of\,cliques} \tag{4}$$

where L (S) is the number of cliques whose maximal execution time is larger (smaller) than the average execution time for the *MNCG*.

If none of these conditions is met, next two cliques are chosen and redeployment step is repeated.

5 Experiment Results

Algorithms described above have been implemented and tested for different shapes of simulation area and for nine system configurations. We introduce a

simple architectural model [1] for a homogenous multiprocessor system, which is described by two values: *CompSpeed* (processor computational speed) and *CommSpeed* (communication performance). In our experiments we have defined 9 types of systems based on various combinations of values of *CompSpeed* and *CommSpeed*, determined by the type of the communication facility applied in a system:

Symbol	*CompSpeed* of a single node	*CommSpeed* between two nodes
FF	Fast - 1 GFlops	Fast - Shared memory
FM	Fast - 1 GFlops	Medium - RDMA RB
FS	Fast - 1 GFlops	Slow - MPI
MF	Medium - 0.3 GFlops	Fast - Shared memory
MM	Medium - 0.3 GFlops	Medium - RDMA RB
MS	Medium - 0.3 GFlops	Slow - MPI
SF	Slow - 0.02 GFlops	Fast - Shared memory
SM	Slow - 0.02 GFlops	Medium - RDMA RB
SS	Slow - 0.02 GFlops	Slow - MPI

First, we have analyzed values of *CliqueB* for different system configurations, Fig. 3a. The *CliqueB* value is measured when a redeployment operation was successful. For small number of processors (64) *CliqueB* achieved 1 almost for all configurations. It means that for all cliques, the maximal execution time is either smaller or larger than the average execution time of the *MDFG*. Further optimization in this case is not possible. If value of *CliqueB* is close to 0, it means that the number of cliques, which can be used in the redeployment algorithm, is large enough to expect a successful optimization. It means that *CliqueB* estimates possible adaptation of a *MNCG* for execution on a given computer system. In Fig. 3b, the speedup of macro data flow graph execution is presented. We can observe that independently on the number of processors and on the efficiency of a single computation node, with or without redeployment of computational nodes, the best speedup was obtained for the computational systems with shared memory (*FF*, *MF*, *SF*) and in two cases with *RDMA RB*[11] communication (*MM*, *SM*). For small number of processors (32) almost the linear speedup is obtained.

It is because, in all of these cases inter-node communication cost was almost negligible. Nevertheless, for configurations with 64 and 256 processing nodes speedup decrease can be observed. It can be explained by larger number of computational nodes that entails increase of the number of communications that must be serialized in each computational step. Slightly worse speedup was obtained for a system with *RDMA RB* communication and with fast computation nodes (*FM*). In this case, the relation between computational and communication parameters of the system does not match program requirements. It is especially visible in the case of systems with slow communication where sometimes the speedup decreased dramatically. It happened only for configurations FS, MS, whereas for *SS* the speedup is similar to the configuration with shared memory.

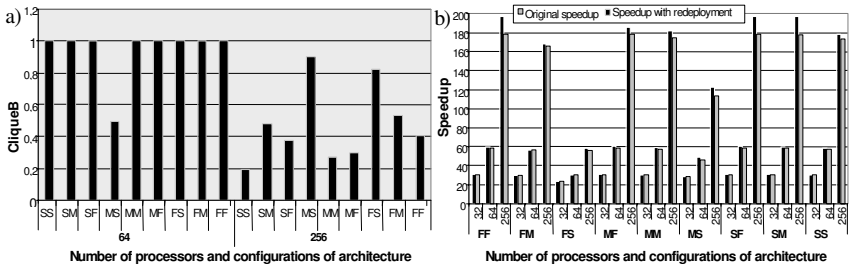


Fig. 3. a) *CliqueB* for various computational system configurations, b) MDFG execution speedup versus single node execution for various computational system configurations

6 Conclusions

A methodology presented in the paper enables designing and profiling macro data flow graphs that represent computation and communication patterns for the *FDTD* problem solved for irregular wave propagation areas. The macro data flow graph is created in two phases, computational area partitioning and merging. After that, it is transformed into a macro node communication graph and load balancing optimization is performed. We defined nine system configurations with different combinations of computational node speed and communication system efficiency (*MPI*, *RDMA RB*, *SHMEM*). It was shown that our *MNCG* transformation strategy could optimize execution speed of *FDTD* applications with success dependent on system features. This work was sponsored by internal grants of the PJIIT and ICSPAS.

References

1. Bharadwaj, D. Ghose, V. Mani, T.G. Robertazi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, California, 1996.
2. S. Dutt, W. Deng, *VLSI Circuit Partitioning by Cluster-Removal using Iterative Improve-ment Techniques*, Proc. IEEE International Conference on Computer-Aided Design, 1997, pp.350-355
3. C.M. Fiduccia and R. M. Mattheyses. *A Linear Time Heuristic for Improving Network Partitions.* Proc. Nineteenth Design Automation Conference, 1982, pp. 175-181.
4. M.Garey, D.Johnson, L.Stockmeyer, *Some simplified NP-complete graph problems*, Theoretical Computer Science, 1 (1976), pp. 237-267
5. G. Karypis, V. Kumar, *Unstructured Graph Partitioning and Sparse Matrix Ordering*, Technical Report, Department of Computer Science, University of Minesota, 1995 (http://www.cs.umn.edu/ kumar)
6. M.S. Khan, K.F. Li, *Fast Graph Partitioning Algorithms*, Proceedings of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, Victoria, B.C., Canada, May 1995, pp. 337-342

7. B.W. Kerighan, S.Lin, *An efficient heuristic procedure for partitioning graphs*, AT&T Bell Labs. Tech. J., 49:291-307, February 1970
8. S.Kirkpatrick, C.D.Gelatt, M.P.Vecchi, *Optimization by simulated annealing*, Science, 220(4598):671-680, May 1983
9. H.X.Lin, A.J.C. van Gemund and J.Meijdam, *Scalability analysis and parallel execution of unstructured problems*, Eurosim'96 Conference
10. R. Sedgewick, *Algorithms in C, Part 5: Graph Algorithms (3rd Edition)*, Addison-Wesley Professional, 3 edition (August 16, 2001), 368 pages, ISBN: 0201316633
11. A.Smyk, M.Tudruj, *RDMA Control Support for Fine-Grain Parallel Computations*, PDP 2004, La Coruna, Spain
12. A.Smyk, M.Tudruj, *Parallel Implementation of FDTD Computations Based on Macro Data Flow Paradigm*, PARELEC 2004, September 7-10, Dresden, Germany

A Note on the Numerical Inversion of the Laplace Transform*

Przemysław Stpiczyński

Department of Computer Science, Maria Curie–Skłodowska University,
Pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland
przem@hektor.umcs.lublin.pl

Abstract. The aim of this paper is to show that the recently developed high performance *divide and conquer* algorithm for finding trigonometric sums can be applied to improve the performance of the Talbot's method for the numerical inversion of the Laplace Transform on modern computer architectures including shared memory parallel computers. We also show how to vectorize the first stage of the Talbot's method, namely computing all coefficients of the trigonometric sums used by the method. Numerical tests show that the improved method gives the same accuracy as the standard algorithm and it allows to utilize parallel processors.

1 Introduction

The problem of the numerical inversion of the Laplace Transform arises in several areas of scientific computing [1]. One of the most popular algorithm for solving this problem is the Talbot's method [9], which was efficiently implemented as simple Fortran routines [5] for variety of computers. Unfortunately, sometimes the algorithm requires a big number of function evaluations. Thus, it is clear that the high performance algorithms for solving our problem should be designed. In [3] one can find a parallel version of Talbot's method which is based on the Reinsch algorithm for computing trigonometric sums defined in terms of linear recurrences, so such a code cannot be simply vectorized. In this paper we consider a new parallel version of the Talbot's method based on the recently developed method for finding trigonometric sums which can be not only parallelized but also vectorized what is essential for achieving reasonable performance of modern processors [4].

2 Talbot's Method

The idea of the Talbot's method [9] for the numerical inverse of the Laplace Transform

$$F(s) = \int_0^\infty e^{-st} f(t)dt, \ \text{Re}\ s > \sigma_0 \tag{1}$$

* The work has been sponsored by the KBN grant 6T11 2003C/06098. The use of Cray X1 from the Interdisciplinary Center for Mathematical and Computational Modeling (ICM) of the Warsaw University is kindly acknowledged.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 551–558, 2006.

is to replace the Riemann inversion formula

$$f(t) = \frac{1}{2\pi i} \int_B e^{st} F(s) ds, \quad B \text{ - Bromwhich contour},$$

with the following integral formula

$$f(t) = \frac{\lambda e^{\sigma t}}{2\pi i} \int_{-\pi}^{\pi} e^{\lambda t s_\nu(\theta)} F(\sigma + \lambda s_\nu(\theta)) s'_\nu(\theta) d\theta, \tag{2}$$

where λ, σ, ν are called geometrical parameters (chosen automatically by the method) and $s_\nu(\theta) = \theta \cot\theta + i\nu\theta$. Using the trapezoidal rule to find the approximate value of (2) we conclude that the Talbot's approximation to $f(t)$ is given by

$$\tilde{f}(t) = \frac{\lambda e^{\sigma t}}{n} \left(\frac{\nu}{2} e^{\lambda t} F(\sigma + \lambda) + S_n(t) \right), \tag{3}$$

where

$$S_n(t) = \sum_{j=1}^{n-1} c_j \cos\phi_j - \sum_{j=1}^{n-1} s_j \sin\phi_j \tag{4}$$

and

$$c_j = e^{\rho_j}(\alpha_j\gamma_j - \beta_j\delta_j), \quad s_j = e^{\rho_j}(\beta_j\gamma_j + \alpha_j\delta_j) \tag{5}$$

and $\theta_j = j\frac{\pi}{n}$, $\rho_j = \lambda t j \frac{\pi}{n} \cot(j\frac{\pi}{n})$, $\phi_j = \lambda t \nu j \frac{\pi}{n}$, $F(\sigma + \lambda s_\nu(\theta_j)) = \alpha_j + i\beta_j$, $\frac{1}{i} s'_\nu(\theta_j) = \gamma_j + i\delta_j$.

It is clear that the formula (4) is the central part of the method and usually (see [5, 3]) the Reinsch algorithm for computing the following sums

$$C(x) = \sum_{k=0}^{n} b_k \cos kx \quad \text{and} \quad S(x) = \sum_{k=1}^{n} b_k \sin kx, \tag{6}$$

is applied to find $S_n(t)$. It can be summarized as follows [6]. We set $S_{n+2} = D_{n+1} = 0$ and if $\cos x > 0$ then we solve

$$\begin{cases} S_{k+1} = D_{k+1} + S_{k+2} \\ D_k = b_k + uS_{k+1} + D_{k+1} \end{cases} \tag{7}$$

for $k = n, n-1, \ldots, 0$, where $u = -4\sin^2\frac{x}{2}$. If $\cos x \le 0$ then we solve

$$\begin{cases} S_{k+1} = D_{k+1} - S_{k+2} \\ D_k = b_k + uS_{k+1} - D_{k+1} \end{cases} \tag{8}$$

where $u = 4\cos^2\frac{x}{2}$. Finally, we compute

$$S(x) = S_1 \sin x \text{ and } C(x) = D_0 - \frac{u}{2} S_1. \tag{9}$$

Note that in case of the Talbot's method, namely for computing $S_n(t)$, it is necessary to use the Reinsch method twice. Firstly to compute $\sum_{j=1}^{n-1} c_j \cos\phi_j$ and then to find $\sum_{j=1}^{n-1} s_j \sin\phi_j$.

The "standard" Reinsch algorithm is equivalent to the problem of solving a second order linear recurrence system, which is rather difficult to parallelize and vectorize. The parallel version of the Reinsch algorithm has been presented in [3], but it cannot be vectorized, because each processor executes a scalar (non-vectorized) code based on (7) or (8).

3 New Improved Method

In [7] we have introduced a new algorithm for computing trigonometric sums (6) (based on the high performance algorithm for solving linear recurrence systems [8]) and discussed its implementation using the operations `AXPY` and `COPY` from the Level 1 BLAS [4]. Unfortunately, it is efficient only for huge values of n, so it cannot be applied to improve the overall performance of the Talbot's method. In this paper we will show how to improve its performance, thus let us briefly present the method. For the simplicity, let us assume that the number n can be factorized as $n = pq$ and set

$$x_k = \begin{cases} S_{n-\lfloor k/2 \rfloor} & \text{for } k = 1, 3, \ldots, 2n-1 \\ D_{n-k/2} & \text{for } k = 2, 4, \ldots, 2n \end{cases},$$

$$f_k = \begin{cases} b_n & \text{for } k = 1 \\ b_{n-1} - \delta b_n & \text{for } k = 2 \\ 0 & \text{for } k = 3, 5, \ldots, 2n-1 \\ b_{n-k/2} & \text{for } k = 4, 6, \ldots, 2n \end{cases} \tag{10}$$

and

$$L = \begin{pmatrix} 1 & & & & & & \\ -u & 1 & & & & & \\ \delta & -1 & 1 & & & & \\ & \delta & -u & 1 & & & \\ & & \delta & -1 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \delta & -u & 1 \end{pmatrix} \in \mathbb{R}^{2n \times 2n}$$

with

$$\delta = \begin{cases} -1 & \text{for } \cos x > 0 \\ 1 & \text{for } \cos x \leq 0 \end{cases}$$

The solution of (7–8) is equivalent to the solution of the system $L\mathbf{x} = \mathbf{f}$, where $\mathbf{x}, \mathbf{f} \in \mathbb{R}^{2n}$. First we find the solution of the block system of linear equations

$$L^{(q)}(\mathbf{z}_1, \ldots, \mathbf{z}_p) = (\mathbf{f}_1, \ldots, \mathbf{f}_p) \tag{11}$$

where $L^{(q)} \in \mathbb{R}^{2q \times 2q}$ is a sub-matrix of L built from its first $2q$ rows and columns and $\mathbf{f}_j = (f_{2(j-1)q+1}, \ldots, f_{2jq})^T$. Next we compute S_1 and D_0 applying

$$\begin{pmatrix} S_1 \\ D_0 \end{pmatrix} = \mathbf{z}_p'' - \sum_{j=1}^{p-1} M^{p-j} \mathbf{z}_j'', \tag{12}$$

where $\mathbf{z}_j^{''}$ denotes two last entries of $\mathbf{z}_j$ and the columns of

$$M = \begin{pmatrix} y_{2q-1}^{(1)} & y_{2q-1}^{(2)} \\ y_{2q}^{(1)} & y_{2q}^{(2)} \end{pmatrix} \in \mathbb{R}^{2\times 2}$$

are built from two last entries of the vectors $\mathbf{y}_1 = (y_1^{(1)}, \ldots, y_{2q}^{(1)})^T$ and $\mathbf{y}_2 = (y_1^{(2)}, \ldots, y_{2q}^{(2)})^T$ which satisfy

$$L^{(q)}\mathbf{y}_1 = (\delta, 0, \ldots, 0)^T, \qquad L^{(q)}\mathbf{y}_2 = (-1, \delta, 0, \ldots, 0)^T. \tag{13}$$

Fortunately, there is no need to solve the systems (13) explicitly. The numbers $y_{2q-1}^{(1)}$, $y_{2q}^{(1)}$, $y_{2q-1}^{(2)}$, $y_{2q}^{(2)}$ can be easily computed using the following theorem.

Theorem 1 ([7]). *For $x \neq k\pi$, k integer, two last entries of the vectors $\mathbf{y}_1$ and $\mathbf{y}_2$ satisfy*

$$y_{2q-1}^{(1)} = (\delta \sin qx + \sin(q-1)x) \,/\, \sin x, \;\; y_{2q}^{(1)} = \delta \cos qx + \cos(q-1)x + \frac{u}{2} y_{2q-1}^{(1)}$$

$$y_{2q-1}^{(2)} = -\sin qx / \sin x, \qquad y_{2q}^{(2)} = -\cos qx + \frac{u}{2} y_{2q-1}^{(2)},$$

where $u = -4\sin^2 \frac{x}{2}$ for $\cos x > 0$ and $u = 4\cos^2 \frac{x}{2}$ for $\cos x \leq 0$.

In [7] we have discussed the implementation of the algorithm. The values for p and q should be chosen to minimize the number of floating point operations (see [7]), thus we set $q = \sqrt{3n/2}$ and $p = n/q$. The matrix $Z = (\mathbf{z}_1, \ldots, \mathbf{z}_p)$ can be found row-by-row using a sequence of calls to the operation `AXPY` and this step can be easily parallelized. However, the use of `AXPY` and `COPY` as an auxiliary routine leads to unnecessary data movements, namely after each call to `AXPY` the result is stored in the main memory. Thus the performance of the algorithm can be poor, especially for smaller values of n.

In order to improve the method, let us consider the special sparse structure of the matrix $F = (\mathbf{f}_1, \ldots, \mathbf{f}_p) \in \mathbb{R}^{2q\times p}$, namely

$$F = \begin{pmatrix} \times & 0 & 0 & \cdots & 0 & 0 & 0 \\ \times & \times & \times & \cdots & \times & \times & \times \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \times & \times & \times & \cdots & \times & \times & \times \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \times & \times & \times & \cdots & \times & \times & \times \end{pmatrix}$$

It consists of $2q$ rows. Each odd row (except for the first one) comprises zero entries, while the entries of k-th row (k even) are as follows:

$$f_k, f_{k+2q}, f_{k+4q}, \ldots, f_{k+2(p-2)q}, f_{k+2(p-1)q}.$$

Using (10) we can observe that this row consists of the coefficients

$$b_{n-k/2}, b_{n-k/2-q}, b_{n-k/2-2q}, \ldots, b_{n-k/2-(p-2)q}, b_{n-k/2-(p-1)q}. \tag{14}$$

Note that the last coefficient in the k-th row is $b_{q-k/n}$ and the first element in the second is equal to $b_{n-1} - \delta b_n$. In case of the Talbot's method, the coefficients (14) are calculated using (5). It should be pointed out that the overall performance of the algorithm can be improved when we vectorize (5). The system (11) can be solved as follows. The solution overrides the matrix F. First we modify the element in the first column of the second row using $F_{2,1} \leftarrow F_{2,1} + uF_{1,1}$. Next for every odd row $k \geq 3$, we perform

$$F_{k,*} \leftarrow F_{k-1,*} - \delta F_{k-2,*}. \tag{15}$$

Then we update the next (even) row using two previous rows

$$F_{k+1,*} \leftarrow F_{k+1,*} + uF_{k,*} - \delta F_{k-1,*}. \tag{16}$$

It should be pointed out that depending on the value of δ, we add or subtract the vector $F_{k-2,*}$ in (15) and $F_{k-1,*}$ in (16).

4 Implementation and Results

The algorithm presented above can be easily parallelized. First let us observe that the matrix Z can be found in parallel. Indeed, the system (11) can be rewritten as $L\left(Z_1, \ldots, Z_{N_p}\right) = \left(F_1, \ldots, F_{N_p}\right)$, where N_p denotes the number of available processors and then each processor is responsible for computing one block Z. The sequential part of the algorithm is based on (12).

The original implementation of the Talbot's method comprises two separate subroutines `TAPAR` for finding geometrical parameters of the contour and `TSUM` for computing (4) and then (3). In our new implementation `TSUM` is replaced with the subroutine `PXTSUM`, which consists of the following main steps. First (in parallel, using the OpenMP [2] construct `parallel`) we calculate coefficients c_j, s_j according to (14), store them in one-dimensional arrays (two arrays for c_j

Table 1. Itanium: $F(s) = 1/s^2$, $f(t) = t$, $t = 100.0$

n	t_1	t_2	e_1	e_2
15195	0.3422E-02	0.2136E-02	0.5171457929E-10	0.5261526326E-10
16281	0.3666E-02	0.2222E-02	0.4826844702E-10	0.4880135407E-10
16801	0.3783E-02	0.2316E-02	0.4677545462E-10	0.4797456654E-10
17177	0.3868E-02	0.2380E-02	0.4575412049E-10	0.4776751439E-10
17711	0.3987E-02	0.2479E-02	0.4436671475E-10	0.4577358936E-10
17941	0.4039E-02	0.2480E-02	0.4379657526E-10	0.4484462579E-10
17865	0.4022E-02	0.2449E-02	0.4399197451E-10	0.4252370900E-10

Table 2. Itanium: $F(s) = \arctan(1/s)$, $f(t) = \sin(t)/t$, $t = 1000.0$

n	t_1	t_2	e_1	e_2
1565	0.6540E-03	0.4010E-03	0.2391931485E-06	0.2391750279E-06
1681	0.6990E-03	0.4220E-03	0.2226849779E-06	0.2226882837E-06
1729	0.7210E-03	0.4182E-03	0.2165089258E-06	0.2165033446E-06
1801	0.7510E-03	0.4539E-03	0.2078623510E-06	0.2078732275E-06
1851	0.7720E-03	0.4790E-03	0.2022340543E-06	0.2022043589E-06
1977	0.8230E-03	0.4809E-03	0.1893500394E-06	0.1893479699E-06
2029	0.8459E-03	0.5062E-03	0.1844935713E-06	0.1845017833E-06
2107	0.8800E-03	0.5372E-03	0.1776737975E-06	0.1776853155E-06
2161	0.9010E-03	0.5391E-03	0.1732306555E-06	0.1732318723E-06
2215	0.9232E-03	0.5641E-03	0.1690140524E-06	0.1690132600E-06

Table 3. Itanium: $F(s) = \arctan(1/s)$, $f(t) = \sin(t)/t$, $t = 10000.0$

n	t_1	t_2	e_1	e_2
588305	0.2448E+00	0.1504E+00	0.9322757351E-09	0.1160215760E-08
639409	0.2660E+00	0.1607E+00	0.7404149541E-09	0.9956170695E-09
690241	0.2870E+00	0.1778E+00	0.7783544787E-09	0.7003191449E-09
792589	0.3296E+00	0.2042E+00	0.6455896457E-09	0.8114018422E-09
844291	0.3511E+00	0.2183E+00	0.6207029052E-09	0.7159488277E-09
946409	0.3935E+00	0.2390E+00	0.5036515674E-09	0.7211593871E-09
998211	0.4151E+00	0.2580E+00	0.4364697181E-09	0.8877389771E-09
1048435	0.4359E+00	0.2714E+00	0.3644243104E-09	0.3733948982E-09
1100497	0.4577E+00	0.2761E+00	0.4464777616E-09	0.2833087599E-09

Table 4. Pentium 4: $F(s) = \arctan(1/s)$, $f(t) = \sin(t)/t$, $t = 1000.0$

n	t_1	t_2	e_1	e_2
1565	0.6928E-03	0.5510E-03	0.2391869846E-06	0.2391608295E-06
1681	0.7119E-03	0.5870E-03	0.2226976753E-06	0.2226957623E-06
1729	0.7231E-03	0.6299E-03	0.2078558583E-06	0.2078766382E-06
1851	0.7741E-03	0.7241E-03	0.2022435329E-06	0.2022139956E-06
1977	0.8249E-03	0.6840E-03	0.1893527022E-06	0.1893563987E-06
2029	0.8581E-03	0.7038E-03	0.1844931378E-06	0.1845026485E-06
2107	0.9139E-03	0.7319E-03	0.1776824021E-06	0.1776997465E-06
2161	0.8988E-03	0.7489E-03	0.1732324141E-06	0.1732241612E-06
2215	0.9251E-03	0.7679E-03	0.1690133932E-06	0.1690198610E-06

to store odd and even rows respectively and similarly two arrays for computing s_j) and find solutions of two systems (11), separately for c_j and s_j. In simple loops we repeat operations (15) and (16) using the arrays defined above. Also let us observe that to calculate coefficients c_j, s_j according to (14) using (5) in a *vectorized way*, we need to use auxiliary arrays to store α_j, β_j, γ_j, δ_j, γ_j and ρ_j. Next we calculate the sums (4) using (12). Finally we calculate the approximate value of $f(t)$ using (3).

Table 5. Pentium 4: $F(s) = \arctan(1/s)$, $f(t) = \sin(t)/t$, $t = 10000.0$

n	t_1	t_2	e_1	e_2
588305	0.2524E+00	0.2166E+00	0.9793066881E-09	0.9892222718E-09
639409	0.2733E+00	0.2392E+00	0.8179742423E-09	0.8752334127E-09
690241	0.2948E+00	0.2566E+00	0.9389228568E-09	0.8547164468E-09
742021	0.3171E+00	0.2764E+00	0.6600419638E-09	0.5786208750E-09
792589	0.3383E+00	0.2943E+00	0.7116522225E-09	0.8339686643E-09
844291	0.3614E+00	0.3155E+00	0.6238256932E-09	0.1262315896E-08
895987	0.3836E+00	0.3357E+00	0.6558746249E-09	0.3426053477E-08
946409	0.4048E+00	0.3540E+00	0.2550043358E-09	0.5339774668E-09
998211	0.4302E+00	0.3739E+00	0.5216922980E-09	0.5289859726E-09
1048435	0.4493E+00	0.3926E+00	0.5044154133E-09	0.4610048013E-09
1100497	0.4708E+00	0.4075E+00	0.4693288363E-09	0.4975408007E-09

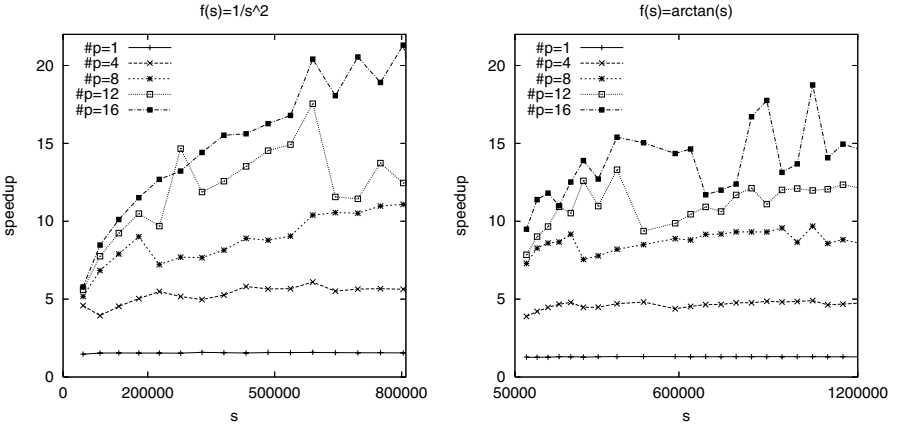


Fig. 1. The speedup on Cray X1

The solution to the systems (11) can be done using a sequence of Level 1 BLAS routines [7]. We can repeat one call to the routine `COPY` and three calls to `AXPY`. The total number of vector loads/stores is equal to 11. When we use (15) and (16) then the total number of vector loads/stores is equal to 7. Also numerical tests show that such simple loops are up to 15% faster on Intel based computers.

The method has been tested for the set of test functions proposed in [5] for various values of t and different desired accuracy on a dual-processor Itanium 2 and Pentium 4 based computers running under Linux with Fortran 95 compiler provided by Intel, and on Cray X1. The method has been compared with the original Talbot's algorithm. The following tables show exemplary results for $F(s) = 1/s^2$ and $F(s) = \arctan(1/s)$, where n denotes number of terms in (4), t_1, e_1 the execution time and the accuracy of the original Talbot's' method and t_2, e_2 execution time (in seconds) and the accuracy of the method described

in this paper. We can observe that on Intel based computers, the speedup of the new method is about 1.6, even when the execution time (and the value of n) is very small. The new method is up to 20% faster on Pentium 4. In Figure 1 we present the speedup on Cray X1. The new method is faster even on one SSP processor and it scales well for greater number of processors.

We haven't observed any significant loss of accuracy of our method in comparison with the original Talbot's method, thus we suppose that the method is numerically stable but it will be a subject of further research.

References

1. Abate, J., Valko, P.: Multi-precision Laplace transform inversion. Int. J. Numer. Mech. Eng. **60** (2004) 797–993
2. Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., Menon, R.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers, San Francisco (2001)
3. de Rosa, M.A., Giunta, G., Rizzardi, M.: Parallel Talbot's algorithm for distributed memory machines. Parallel Computing **21** (1995) 783–801
4. Dongarra, J., Duff, I., Sorensen, D., Van der Vorst, H.: Solving Linear Systems on Vector and Shared Memory Computers. SIAM, Philadelphia (1991)
5. Murli, A., Rizzardi, M.: Algorithm 682: Talbot's method for the Laplace inversion problem. ACM Trans. Math. Soft. **16** (1990) 158–168
6. Stoer, J., Bulirsh, R.: Introduction to Numerical Analysis. 2nd edn. Springer, New York (1993)
7. Stpiczyński, P.: Fast parallel algorithms for computing trigonometric sums. In Tudruj, M., Jordan, A., eds.: Proceedings of PARELEC 2002, International Conference on Parallel Computing in Electrical Engineering, IEEE Computer Society Press (2002) 299–304
8. Stpiczyński, P.: A new message passing algorithm for solving linear recurrence systems. Lecture Notes in Computer Science **2328** (2002) 466–473
9. Talbot, A.: The accurate numerical inversion of Laplace transforms. J. Inst. Maths. Applics. **23** (1979) 97–120

Mesh Adaptation Based on Discrete Data

Barbara Głut and Tomasz Jurczyk

AGH University of Science and Technology, Kraków, Poland
{glut, jurczyk}@uci.agh.edu.pl

Abstract. This paper examines the procedure of mesh adaptation on plane using the concept of an anisotropic metric. The metric is coupled with the curvature of the solution surface and it governs the process of mesh generation. The metric values are determined from the discrete data from the current simulation step and are stored in the background mesh with the appropriate interpolation procedure. If the solution is given in the form of a vector field, each component is treated separately and can define different metric. In order to combine these metrics, an intersection procedure is used. Several examples of numerical mesh adaptation are provided to illustrate the potential of the described method.

1 Introduction

In the numerical simulation of processes, using finite element method, the adaptation of the element mesh is an important factor influencing the quality of the numerical solution. The aim of the mesh adaptation is to adjust size and shape of elements in order to obtain good approximation of the exact solution with possibly lowest number of elements. The popular methods of adaptation [1, 2, 3] perform a sequence of mesh modification, in order to obtain the desired precision of the simulation results.

In the works describing methods of mesh generation for adaptation, many authors use the concept of metric[2, 3, 4, 5, 6, 7]. For two-dimensional cases the metric can be coupled with the curvature of the solution surface. This approach is often used for generation of meshes on surfaces as well[7, 8].

In this article we consider a scalar function $u(x, y)$ being the solution of an arbitrary physical problem defined in the bounded domain $\Omega \subset R^2$. In case of solving this problem with an approximated method (e.g. FEM), the solution is given in discrete points (nodes of the mesh). These data are the input for the procedure of mesh adaptation. Using this information an approximated curvature of the solution surface can be evaluated which is then used for definition of the metric. In order to determine the curvature from the discrete information, the surface can be locally approximated with a polynomial and then the curvature can be calculated analytically[5, 9]. In the approach proposed by the Authors the curvature is approximated directly from the discrete data. Differently from the method proposed in [3] the curvature is not only combined with the Hessian of solution, but it is determined from the second fundamental form of the surface, which allows to further improve the process of mesh adaptation.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 559–566, 2006.

In this article there is also described a procedure of mesh adaptation for solution given in the form of the vector field. Each of the component of such vector field can define different surface of solution. The proposed technique allows to create a uniform metric resulting from the separate surfaces.

2 Metric Definition

The metric, specifying size and shape of elements, is usually defined in the form of a *metric tensor* $\mathcal{M} = \mathcal{R}\Lambda\mathcal{R}^{-1}$, where $\mathcal{R}$ is the eigenvector matrix (responsible for defining main directions) and $\Lambda = \mathrm{diag}(\lambda_i)$ is the eigenvalue matrix, defining the required length of element edges along the main directions.

Using this metric formulation, the distance between two points P_1 and P_2 in metric space can be calculated as follows:

$$l_{\mathcal{M}}(\overrightarrow{P_1P_2}) = \sqrt{\overrightarrow{P_1P_2}^T \mathcal{M} \overrightarrow{P_1P_2}} \tag{1}$$

where $\mathcal{M}$ stands for the metric assumed as locally constant for $\overrightarrow{P_1P_2}$. In a similar manner, other geometrical formulae used during mesh generation can be adjusted.

Another approach, which was used in this work, is to introduce the anisotropic metric through the appropriate transformation of the coordinates of the discretization points, using the *transformation matrix*:

$$P_i' = \mathbf{M_t}(P_i)P_i \tag{2}$$

The relation between the *metric tensor* and the *transformation matrix* is given by formula:

$$\mathbf{M_t}(P)\mathbf{M_t}^T(P) = \mathcal{M}(P) \tag{3}$$

Using the *transformation matrix* approach, the distance between two points P_1 and P_2 can be calculated as follows:

$$l_{\mathcal{M}}(\overrightarrow{P_1P_2}) = l_E(\overrightarrow{P_1'P_2'}) \tag{4}$$

where $l_E(\overrightarrow{P_1'P_2'})$ is the Euclidean distance between the two points P_1' and P_2' with transformed coordinates.

During the mesh generation all geometric properties are calculated from the coordinates transformed with the appropriate transformation matrix. For example, for a given mesh to comply with the defined metric field, the length of each edge in the metric space should be equal to 1.

2.1 Combining Metric with Curvature

At any point on a smooth surface patch (regular, C^2) there can be defined the *principal directions* in which the normal curvature has minimum and maximum

values (i.e. *principal curvatures* κ_1 and κ_2). Assuming size of elements proportional to the appropriate radii of curvature ($r \approx \frac{1}{\kappa}$), the components of the *direct form* of the metric $m_d = (l_x, l_y, \alpha)$ can be calculated as follows:

$$l_{x_i} = \max\left(\min\left(\frac{c}{|\kappa_i|}, l_{\max}\right), l_{\min}\right) \tag{5}$$

where the minimum and maximum element size are introduced in order to avoid unpracticable metrics. The α is the angle between the principal direction for κ_1 and the base vector $\dot{\mathbf{x}}$.

From the *direct form* the *transformation matrix* can be determined:

$$\mathbf{M_t} = \begin{bmatrix} m_{11}\ m_{12} \\ m_{12}\ m_{22} \end{bmatrix} = \begin{bmatrix} \frac{1}{l_x}\cos^2\alpha + \frac{1}{l_y}\sin^2\alpha & (\frac{1}{l_x} - \frac{1}{l_y})\sin\alpha\cos\alpha \\ (\frac{1}{l_x} - \frac{1}{l_y})\sin\alpha\cos\alpha & \frac{1}{l_x}\sin^2\alpha + \frac{1}{l_y}\cos^2\alpha \end{bmatrix} \tag{6}$$

If the reverse operation is required, the *direct form* can be calculated using the following algorithm:

$$\begin{aligned} \Delta &= \sqrt{(m_{11} - m_{22})^2 + 4m_{12}^2} \\ (l_x, l_y) &= \begin{cases} (\frac{2}{m_{11}+m_{22}+\Delta}, \frac{2}{m_{11}+m_{22}-\Delta}) & \text{if } |m_{11}| < |m_{22}| \\ (\frac{2}{m_{11}+m_{22}-\Delta}, \frac{2}{m_{11}+m_{22}+\Delta}) & \text{else} \end{cases} \\ \alpha &= \begin{cases} \arctan\frac{\frac{1}{l_x}-m_{11}}{m_{12}} & \text{if } |m_{12}| > \epsilon \\ 0 & \text{else} \end{cases} \end{aligned} \tag{7}$$

3 Approximation of Curvature from the Solution Surface

For the purpose of an adaptation, the metric for mesh generation was combined with the curvature of the solution surface. The required partial derivatives (both first and second order) are approximated in each mesh node by difference quotients, using linear combinations of values of selected adaptation parameter in the neighboring nodes (Fig. 1). Details of the method of discrete approximation of derivatives by difference quotients for arbitrary node configuration in a mesh are described in [3].

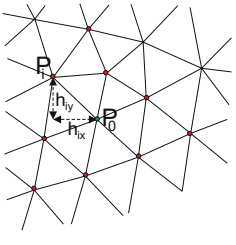


Fig. 1. Selection of nodes

For the second order derivatives and the 2^{nd} order accuracy 9 nodes are required. The partial derivatives are estimated from the linear combination:

$$\begin{array}{l} \frac{\partial^2 u}{\partial x^2} \approx \sum_{i=1}^{9} \alpha_i^{xx}(u_i - u_0) \\ \frac{\partial^2 u}{\partial x \partial y} \approx \sum_{i=1}^{9} \alpha_i^{xy}(u_i - u_0) \\ \frac{\partial^2 u}{\partial y^2} \approx \sum_{i=1}^{9} \alpha_i^{yy}(u_i - u_0) \end{array} \tag{8}$$

The unknown coefficients α_i depend from the node configuration and are calculated from the following sets of equations:

$$\begin{bmatrix} h_{1x} & h_{2x} & \cdots & h_{9x} \\ h_{1y} & h_{2y} & \cdots & h_{9y} \\ h_{1x}^2 & h_{2x}^2 & \cdots & h_{9x}^2 \\ h_{1x}h_{1y} & h_{2x}h_{2y} & \cdots & h_{9x}h_{9y} \\ h_{1y}^2 & h_{2y}^2 & \cdots & h_{9y}^2 \\ h_{1x}^3 & h_{2x}^3 & \cdots & h_{9x}^3 \\ h_{1x}^2 h_{1y} & h_{2x}^2 h_{2y} & \cdots & h_{9x}^2 h_{9y} \\ h_{1x}h_{1y}^2 & h_{2x}h_{2y}^2 & \cdots & h_{9x}h_{9y}^2 \\ h_{1y}^3 & h_{2y}^3 & \cdots & h_{9y}^3 \end{bmatrix} \begin{bmatrix} \alpha_1^{xx} & \alpha_1^{xy} & \alpha_1^{yy} \\ \alpha_2^{xx} & \alpha_2^{xy} & \alpha_2^{yy} \\ \alpha_3^{xx} & \alpha_3^{xy} & \alpha_3^{yy} \\ \alpha_4^{xx} & \alpha_4^{xy} & \alpha_4^{yy} \\ \alpha_5^{xx} & \alpha_5^{xy} & \alpha_5^{yy} \\ \alpha_6^{xx} & \alpha_6^{xy} & \alpha_6^{yy} \\ \alpha_7^{xx} & \alpha_7^{xy} & \alpha_7^{yy} \\ \alpha_8^{xx} & \alpha_8^{xy} & \alpha_8^{yy} \\ \alpha_9^{xx} & \alpha_9^{xy} & \alpha_9^{yy} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{9}$$

For gradient approximation with the same accuracy (2^{nd} order), only five nodes are required. The linear combinations for partial derivatives:

$$\begin{array}{l} \frac{\partial u}{\partial x} \approx \sum_{i=1}^{5} \alpha_i^{x}(u_i - u_0) \\ \frac{\partial u}{\partial y} \approx \sum_{i=1}^{5} \alpha_i^{y}(u_i - u_0) \end{array} \tag{10}$$

and the sets of equations for the α_i coefficients:

$$\begin{bmatrix} h_{1x} & h_{2x} & \cdots & h_{5x} \\ h_{1y} & h_{2y} & \cdots & h_{5y} \\ h_{1x}^2 & h_{2x}^2 & \cdots & h_{5x}^2 \\ h_{1x}h_{1y} & h_{2x}h_{2y} & \cdots & h_{5x}h_{5y} \\ h_{1y}^2 & h_{2y}^2 & \cdots & h_{5y}^2 \end{bmatrix} \begin{bmatrix} \alpha_1^{x} & \alpha_1^{y} \\ \alpha_2^{x} & \alpha_2^{y} \\ \alpha_3^{x} & \alpha_3^{y} \\ \alpha_4^{x} & \alpha_4^{y} \\ \alpha_5^{x} & \alpha_5^{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{11}$$

Having these operators, the matrices of the first and second fundamental form can be calculated (12). The coefficient n_z is the component of the normal vector, which can be calculated as orthogonal to the gradient directions or approximated as an average of the normal vectors of the mesh elements adjacent to the selected node.

$$\mathbf{G} = \begin{bmatrix} 1 + \left(\frac{\partial u}{\partial x}\right)^2 & \frac{\partial u}{\partial x}\frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial x}\frac{\partial u}{\partial y} & 1 + \left(\frac{\partial u}{\partial y}\right)^2 \end{bmatrix} \quad \mathbf{B} = n_z \begin{bmatrix} \frac{\partial^2 u}{\partial x^2} & \frac{\partial^2 u}{\partial x \partial y} \\ \frac{\partial^2 u}{\partial x \partial y} & \frac{\partial^2 u}{\partial y^2} \end{bmatrix} \tag{12}$$

Solving (13) the principal curvatures κ_1 and κ_2 can be established:

$$\det(\mathbf{B} - \kappa \mathbf{G}) = 0 \tag{13}$$

The eigenvectors $\dot{\mathbf{v}}_\mathbf{i}$ defining the principal directions of curvature satisfy:

$$(\mathbf{B} - \kappa_i \mathbf{G})\dot{\mathbf{v}}_\mathbf{i} = \mathbf{0} \tag{14}$$

4 Operation on Metric

During the process of mesh generation some guidance has to be provided to the mesh generator to specify the size and shape of mesh elements. In a typical approach, a *control space* is defined as a covering of the domain responsible for storing sizing information, e.g. in form of a metric. The control space can be defined analytically, but in most cases (and especially in the area of automated and adapted mesh generation) some kind of discrete structure is used (typically background mesh or quad-tree). In our approach the mesh from the previous step of the adaptation procedure is taken as a background mesh.

4.1 Metric Interpolation

Since the metric data is stored in the discrete points only, an additional method of metric interpolation is required in order to provide sizing information in the whole triangulated domain. Whenever the mesh generator needs the sizing information at some point P within the triangulated domain, the control space is responsible for establishing the appropriate metric. This procedure starts with finding the triangle of the background mesh which contains the given point P. Than the set V_* of nodes (*natural neighborhood*) is selected, which consists of all vertices of the containing triangle and all vertices of other triangles which have point P in theirs circumcircles (detailed procedure of using the background mesh can be found in [10]). From this set of nodes the resulting metric is calculated as follows:

$$\mathbf{M_t}(P) = \frac{1}{\sum_{P_i \in V_*} \omega_i} \sum_{P_i \in V_*} \mathbf{M_t}(P_i)\omega_i \tag{15}$$

where $\omega_i = d(P, P_i)^{-2}$.

4.2 Metric Intersection

If the solution is given in the form of a vector field, each component is treated separately and can define different surface. From each surface appropriate metric is approximated giving a vector of metrics for each discrete node of the background mesh. In order to combine these metrics into one metric, an intersection procedure is used (Fig. 2).

First the metric $\mathbf{M_t^*}$ with the highest stretch ratio is selected and its base vectors $(\mathbf{v}_x, \mathbf{v}_y)$ are calculated (using conversion of *transformation matrix* into

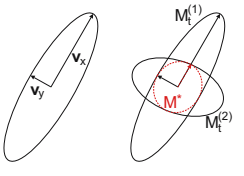


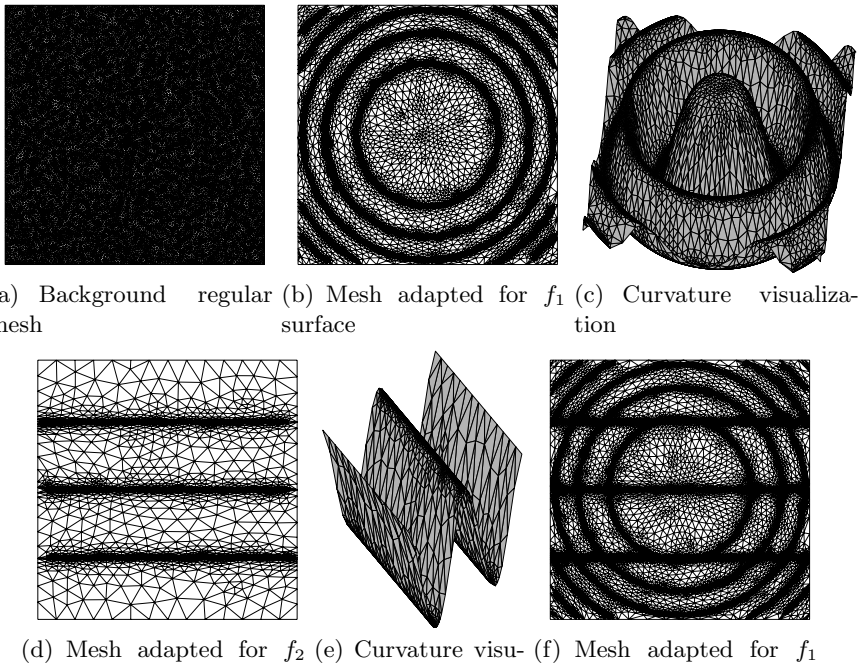
Fig. 2. Calculating intersection of metrics

direct form). Next, the length of these vectors is calculated in all other metrics and appropriately shortened if greater than 1. Finally these vectors are converted back to the *transformation metric* form $\mathbf{M}_{\mathbf{t}}^{*}$.

5 Application Examples

In this section we present examples of unstructured anisotropic mesh adaptation intended to illustrate the possibilities of the described method. In each case an uniform triangular mesh was created for a rectangular domain $[-1,1] \times [-1,1]$, as the initial mesh for the adaptation procedure. The values at the nodes of this mesh were set according to the selected function $f(x,y)$ and it was the only step where analytical form was used. The whole algorithm of automated metric recognition uses exclusively discrete approximations.

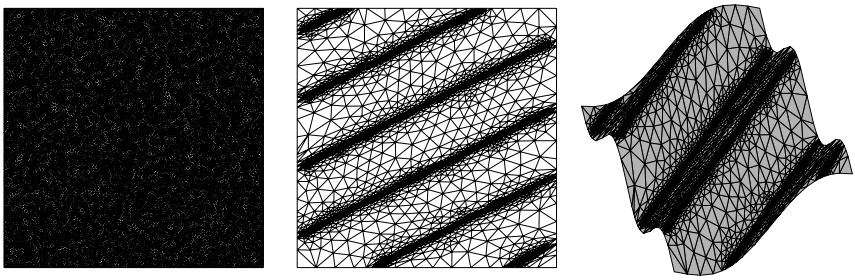
The first example (Fig. 3) presents results of mesh adaptation (the first step) for analytical surface of solution $f_1(x,y) = \cos(10(x^2+y^2))e^{-(x^2+y^2)}$ (Fig. 3(b), 22470 triangles) and $f_2(x,y) = 2\cos(6y)$ (Fig. 3(d), 4606 triangles) independently. Figure 3(a) shows the initial uniform mesh with 31762 triangles, which becomes the background mesh for the adaptation. Figures 3(c) and 3(e) present the matching of the mesh structure to the given surface. In Fig. 3(f) there is



(a) Background regular mesh (b) Mesh adapted for f_1 surface (c) Curvature visualization

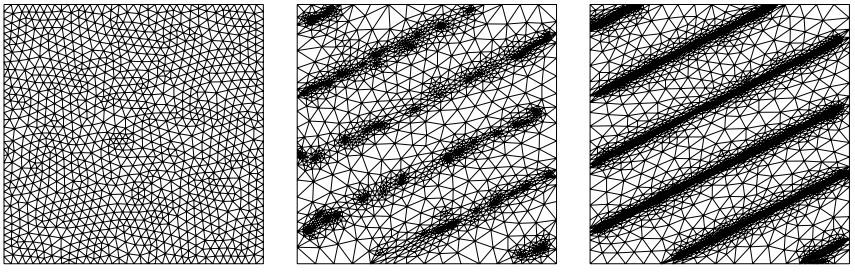
(d) Mesh adapted for f_2 surface (e) Curvature visualization (f) Mesh adapted for f_1 and f_2 surfaces

Fig. 3. Results of mesh adaptation for solution surfaces prescribed by analytical functions $f_1(x,y) = \cos(10(x^2+y^2))e^{-(x^2+y^2)}$ and $f_2(x,y) = 2\cos(6y)$



(a) Background mesh (b) Mesh adapted for f_3 surface (c) Curvature visualization

Fig. 4. First step of dense mesh adaptation for solution surface prescribed by an analytical function $f_3(x, y) = 2\sin(3(x - 2y))$



(a) Background mesh (b) Mesh adapted for f_3 surface (1^{st} step) (c) Mesh adapted for f_3 surface (4^{th} step)

Fig. 5. Subsequent steps of sparse mesh adaptation for solution surface prescribed by an analytical function $f_3(x, y) = 2\sin(3(x - 2y))$

presented the result of mesh adaptation for f_1 and f_2 simultaneously (simulating the performance of the method for solutions with a vector field).

The second example (Fig. 4 and 5) present results of mesh adaptation for different density of the initial mesh. For dense initial mesh (Fig. 4(a), 31762 triangles) the curvature of solution is properly recognized after the first step (Fig. 4(b), 7484 triangles). If the adaptation starts with more sparse mesh (Fig. 5(a), 2028 triangles), several steps of adaptation may be necessary in order to obtain good results (each time the adapted mesh becomes the background mesh for the next adaptation step). Figure 5(b) presents the mesh after the first step of adaptation (with 3874 triangles) and Fig. 5(c) shows the mesh (9272 triangles) after four adaptation steps, when the adaptation procedure converges.

6 Conclusion

The presented method allows to properly adapt the mesh to the curvature of the solution surface. The initial mesh for adaptation needn't be very dense, although

for sparse background meshes several adaptation steps may be required. Since the used method of curvature recognition by differential quotients in each mesh node relies on proper selection of set of neighboring nodes for approximation, the best results are achieved in the interior of the domain. Further work will concentrate on enhancing the quality of curvature recognition at the boundaries of the domain as well.

Acknowledgments. The partial support of the Polish Committee for Scientific Research (KBN) Grant No. 3T11F00828 is gratefully acknowledged.

References

1. Buscaglia, G., Dari, E.: Anisotropic mesh optimization and its application in adaptivity. Int. J. Numer. Meth. Engng. **40** (1997) 4119–4136
2. Dolejsi, V.: Anisotropic mesh adaptation for finite volume and element methods on triangular meshes. Comput. Visual Sci. **1** (1998) 165–178
3. Głut, B., Jurczyk, T., Matuszyk, P., Pietrzyk, M.: Anisotropic 2D meshes in finite element modelling. In: Proc. 2^{nd} European Conference on Computational Mechanics, Kraków, Poland (2001) 1865–1872
4. Labbé, P., Dompierre, J., Vallet, M.G., Guibault, F., Trépanier, J.Y.: A measure of the conformity of a mesh to and anisotropic metric. In: Proc. 10^{th} Int. Meshing Roundtable, Newport Beach, California, USA (2001) 319–326
5. Alauzet, F., George, P., Mohammadi, B., Frey, P., Borouchaki, H.: Transient fixed point-based unstructured mesh adaptation. Int. J. Numer. Meth. Fluids **43** (2003) 729–745
6. Frey, P.: Anisotropic metrics for mesh adaptation. In: Proc. ECCOMAS 2004, Jyvaskyla, Finland (24–28 July 2004)
7. Bottasso, C.: Anisotropic mesh adaptation by metric-driven optimization. Int. J. Numer. Meth. Engng. **60** (2004) 597–639
8. Frey, P., Borouchaki, H.: Surface meshing using a geometric error estimate. Int. J. Numer. Meth. Engng. **58** (2003) 227–245
9. M^{c}Ivor, A., Valkenburg, R.: A comparison of local surface geometry estimation methods. Machine Vision and Applications **10** (1997) 17–26
10. Głut, B., Jurczyk, T.: Definition and interpolation of discrete metric for mesh generation on 3D surfaces. Computer Science, Annual of AGH University of Science and Technology (2005) in press

Applying Cooperating Distributed Graph Grammars in Computer Aided Design

Ewa Grabska and Barbara Strug

Institute of Computer Science, Jagiellonian University,
Nawojki 11, Cracow, Poland
{uigrabsk, uistrug}@cyf-kr.edu.pl

Abstract. This paper deals with a graph grammar model of cooperation and distribution for generating designs in computer aided design domain. It is based on earlier research in formal language theory based models for design, especially graph grammars, and also on the developments in the theory of grammar systems. The motivation for the idea presented is given and some possible modes of application are briefly described.

1 Introduction

Graphs are very useful as a mean to represent complex objects in different domains of computer science [17]. Their ability to represent the structure of an object as well as the relations of different types between its components makes them particularly useful in computer aided design. They can represent an artifact being designed taking into account the inter-related structure of many design objects i.e. the fact that parts of an object can be related to other parts in different ways. Designing new artifacts requires a method of generating representing them graphs as well as an easy method to visualize objects represented by such structures.

This paper proposes a new approach to graph generation in computer aided design domain. In our approach cooperation and distribution give a base for generating designs with the use of graph grammars. It is based on earlier research in the domain of application of the theory of formal languages to the computer aided design [7, 8, 10]. In particular the graph based representation jointly with graph grammars [1, 7, 8, 9, 11, 14], and grammar systems [2, 5, 6] were used as the inspiration for this research.

The main problem of graph generation with graph grammars lies in the complexity and size (understood as the number of rules) of grammars needed in real world problems (that can be encountered in the domains of both engineering and design). Such grammars are either very difficult to define or, even if they are relatively easy to define, they are very difficult to implement or results in very long execution times.

Thus the use of a number of simpler grammars cooperating together is proposed in this paper. Two main types of such systems are considered in the theory

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 567–574, 2006.

of formal languages for string grammars. The one that seems to be the most useful in the design domain is described. Some problems that arise with the use of such formalism are also sketched.

2 Graphs and Graph Grammars

The advantages of graph-based representation, different types of graphs used in this domain, for example composition graphs and their hierarchical extension, their formal definitions and application to design can be found in [7, 8, 9]

A graph is defined as a pair (V, E), where V and E are sets of graph nodes and edges, respectively.

For the needs of a design system a node in a graph may represent either an object or a group of objects or, more generally, it may be seen as a "container" for a part of a design that may be designed at later time. For example, in a house design system a node can represent a floor of a house that in turn will be divided into rooms. Each node of a graph is labelled by a symbol from a predefined set (alphabet) that is later used to identify what the given node represents.

Edges represent relations between objects. They are labelled by symbols being names od the relations. Labels are assigned to nodes and edges by means of node and edge labelling functions.

Both objects and relations between them may in turn have some features. To represent them attributing of both nodes and edges is used. Attributes represent properties (for example size, position, colour or material) of an element represented by a given node. Attributes are assigned by node and edge attributing functions. Formally, an *attribute* is a function $a : W \rightarrow D_a$, where W is a domain of the attribute and D_a a set of possible values of the attribute.

Let R_V and R_E be the sets of node and edge labels, respectively. Let A and B be sets of node and edge attributes and D_a and D_b sets of possible values of attributes of nodes and edges, respectively.

Definition 1. *A labelled attributed graph LAG over* $R_V \cup R_E$ *is defined as a 6-tuple* $LAG = (V, E, lab_V, lab_E, att_V, att_E)$*, where*

1. (V, E) *is a graph,*
2. $lab_V : V \rightarrow R_V$ *is a node labelling function,*
3. $lab_E : E \rightarrow R_E$ *is an edge labelling function,*
4. $att_V : V \rightarrow P(A)$ *is a function assigning a set of attributes to each node,*
5. $att_E : E \rightarrow P(B)$ *is an edge attributing function, i.e. it assigns a set of attributes to each edge.*

In fig. 1 two graphs representing parts of an estate are shown. Nodes represent parts of house (rooms, halls) and its exterior (for example a garden) and edges represent communication possibility between them. This graph may be further developed to add more details, for example each room can be filled with furniture, doors, windows etc.Attributes defining such parameters like size or distance are also shown on the picture.

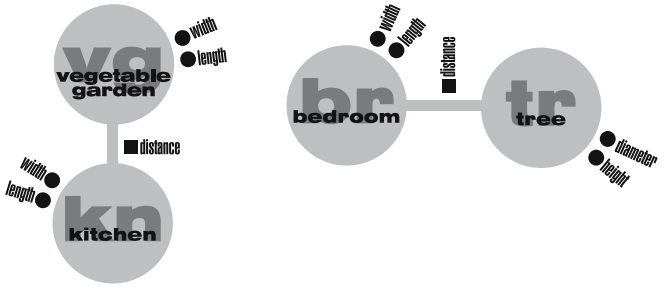


Fig. 1. Example of a graph representing part of a house design

A labelled attributed graph defined above can represent a potentially infinite number of designs. The specification consisting of particular numbers of nodes and edges determines a LAG graph for infinite number of designs having the same structure. To represent an actual design we must define a graph interpretation.

An interpretation of a graph G is defined as a pair of functions (I_V, I_E), where I_V assigns geometrical objects to nodes, while I_E establishes a correspondence between edges and relations between these objects. The objects assigned by I_V are usually called primitives. They may be either simple geometrical objects or more complex objects. The geometry of these objects may be internally represented by means of any known representation that allows for easy application of similarity transformations. Geometrical objects used depend on the domain of application.In a house design system a set of primitives can contain such elements as doors, windows, cabling, different pieces of furniture or elements like bulbs. When designing pieces of furniture like chairs the set of geometrical objects could contain boxes and cylinders, or some predefined objects like legs, seats and other parts of a chair and a set of relations could consist of an adjacency relation.

The interpretation allows us to define attribute functions for a graph and then determine its instances. An instance of a graph is a labelled attributed graph in which to each nodes and edges values of their attributes are assigned. In a house design system for all rooms we have to specify its size attribute for example.

When having a graph interpretation we can create a visualisation of a designed object. By changing the set of available primitives the interpretation (and a resulting design) is changed without any changes made to its graph structure.

The graphs representing designs may be dynamically generated by means of so called graph grammars [1, 8]. A graph grammar can be designed to allow for generating only graphs representing the particular class of designs.

Intuitively, a graph grammar is described as a system of graph transformations consisting of formal rules called productions. Each production is composed of left-hand side and right-hand side graphs and the first graph is replaced by the second one only if it appears in the graph to be transformed. A production is applied to a current graph (starting from the initial one). The sequence of such replacements, which starts from the initial graph, is called a derivation process.

The major problem that arises in practice cases [1, 11, 14] is the size and complexity of a grammar needed to generate real world objects. Very often there is a need to use context grammars, grammars with additional parameters (variables) or other extensions. Such grammars, while relatively easy to define, are more difficult to implement. It thus seems advantageous to use a number of simpler grammars, cooperating in an effort to generate a design, instead of one complex grammar.

3 Designing with Cooperating Graph Grammar System

A grammar system consists of a finite number of grammars (called components of the system) which together influence (change) an environment by applying some operations to it. At a given moment the state of the system is described by the current environment (sub-environments). The system works by changing its state.

Two main types of grammar systems are researched: parallel communicating grammar systems (PC grammar systems) [4, 5, 15] and cooperating distributed grammar systems (CD grammar systems) [2, 3, 12, 13]. The main difference between these types of grammar systems consists in the model of environment they work on.

In PC grammar systems each component operates on its own copy of the form under derivation and they only exchange information on request.

In case of CD grammar systems the grammars operate on one common form, one grammar at a time. As in computer aided design domain a single object is usually developed, the CD grammar systems seem more appropriate here. They allow for a number of grammars to work together on one object. At any given time step there is only one graph being generated. Each component grammar operates on this graph according to a communication protocol called the derivation mode. The protocol may allow a single component to performed a predefined number of steps or to work until no production of this component can be applied. The method of selecting which grammar should be used as the next "active" component is also important. Such cooperating grammars can be seen as independent cooperating agents solving the same problem by modifying a common environment. (often compared with the blackboard model in artificial intelligence) [12].

In this paper we will deal with a modification of CD grammar systems. Yet, in case of graph grammars the requirement that only one grammar can operate on a given temporary form (called sentential form) seems too strong. Moreover in design systems it would be useful to allow more then one grammar to operate on the same, common, sentential form. It can be seen an equivalent of many "design teams" working on different parts of the same design. The only requirement here seems to be ensuring that no two teams work at the same time on the same part of the design, i.e no two grammars (components) operate on the same nodes of the sentential form.

Moreover, some way of activating particular components must be defined. Such method, called a cooperation strategy, may either be based on some predefined order - thus leading the system to operate under the control of an "external supervisor". Or, it can be based on dynamic activation of components related to the current state of sentential form. The way a given grammar works (called derivation mode), i.e. how long it performs its operations must also be defined. In this paper a so called terminal derivation mode is used, i.e. each grammar works as long as it contains a production that can be applied to current graph.

Definition 2. *A graph grammar system is defined as* $(n+3)$*-tuple* $GGS = (N, T, G_1, G_2, \ldots, G_n, g_0)$*, where*

- N *and* T *are the sets of non-terminal and terminal symbols, respectively,*
- $G_i = (N_i, T_i, C_i, P_i), (i = 1 \ldots n)$ *are graph grammars such that*
  - N_i*,* T_i *and* C_i *are the sets of non-terminal, terminal and communication symbols for the* $i-th$ *a component, respectively,*
  - $N_i \cap T_i \cap C_i = \emptyset$*,*
  - $N = \bigcup_{i=1}^{n} N_i, T = \bigcup_{i=1}^{n} T_i$*,*
  - $\forall x \in C_i, (i = 1 \ldots n)\ \exists j \neq i$ *such that* $x \in N_j$*,*
  - $\forall x \in N_i, (i = 1 \ldots n)\ \exists p \in P_i$ *such that its left side graph is labelled by* x*,*
  - P_i *is a set of productions* $G_L \Rightarrow G_R$ *where* G_L *and* G_R*, called left side and right side, respectively, are the labelled attributed graph over* $N_i \cup T_i \cup C_i$ *and* $|V_{G_L}| = 1 \wedge lab_{G_L} \in N_i$*,*
- g_0 *is an initial graph.*

For short nodes labelled with terminal, non-terminal and communication symbols will be called, terminal, non-terminal and communication nodes, respectively.

The language generating by such a grammar system consists of all LAGs over $T = \bigcup_{i=1}^{n} T_i$ that can be generated by applying productions from component grammars starting from the initial graph g_0. Thus this language accepts graphs that contain only nodes labelled by terminal symbols.

Each component grammar contains productions that have only one node on the left side. Moreover this node is labelled by a symbol from the set of non-terminals. Actually the set of non-terminals for each grammar is defined as a set of all symbols used to label nodes of left sides of the productions. The nodes of the graph on the right side of the production can labeled by any symbol, terminal, non-terminal or communication one.

The non-terminal nodes can be intuitively understood as representing a part of a design that is not finished but a given grammar knows how to deal with it further. Taking this intuition further a communication symbol means that a particular grammar knows that some part of design is not finished but it is not "specialist" in this part and thus it "communicates" with other grammar that knows how to deal with this part. Terminal symbols finally represent part of a design that is considered to be finished.

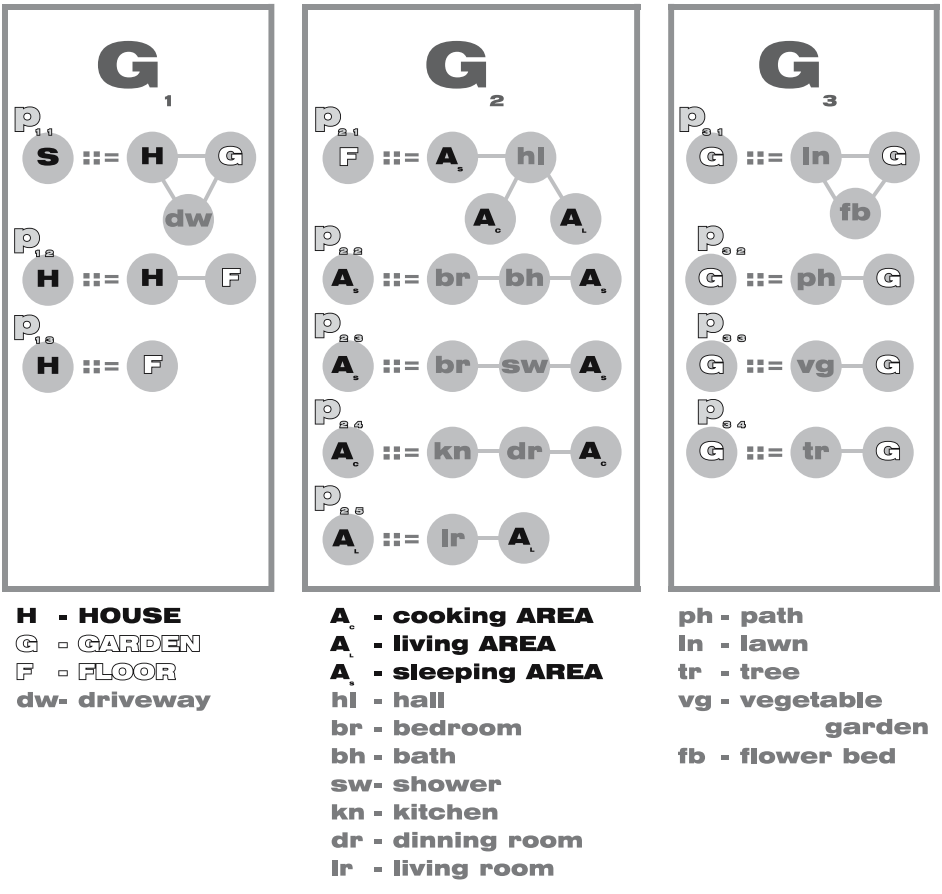


Fig. 2. Selected productions from house design system

The definition of a grammar system guarantees that if any grammar introduces a communication symbol into the sentential form then at least one other grammar exists such that it contains the same symbol in its set of non-terminals. It should be noted here that the set of non-terminal symbols of the grammar system contains not only all non-terminal symbols from all component grammars but also all their communication symbols. It means that the communication symbols are in fact non-terminals symbols, but the grammar that introduces them does not know what they mean and thus another grammar must be found that knows how to deal with these symbols.

In fig. 2 some productions of grammars used to design a house are presented. The grammar G_1 is responsible for the general structure like placing a garden (node labelled G), and house interior (H) and then dividing it into floors (F). Then grammar G_2 deals with floor layout. Nodes labelled by A_S, A_L and A_C

correspond to sleeping, living and cooking/eating areas of a floor, hl - to a hall, dr - dining room, kn - kitchen, br and bh - to bedroom and bathroom. Grammar G_3 is responsible for designing a garden, with fb corresponding to flower beds, ln - lawn, tr - tree and ph to paths of a garden.

In this example the grammars may be compared to different teams (agents) building the house. One agent can build walls and divide the house into floors (grammar G1 in fig. 2). Then another one divides the floors thus creating rooms, but it may not know how to design elements of the room so it "calls" another agent that knows how to deal with this particular task. In a graph representation a floor would be represented by a communication symbol introduced by a grammar responsible for designing the house layout and left for a grammar that is responsible for designing the floors. Moreover these grammars can work in parallel way. When a grammar introduces a communication symbol it "broadcasts" this information and a grammar that understands this symbol starts working (i.e. applying its productions). This grammar may in turn activate other grammars. Each grammar is expected to work as long as it has a production that it can apply to a current graph.

4 Conclusions

In this paper a new idea of intelligent design by cooperating grammar systems was proposed. This method was illustrated by a simple and easy to understand problem of house designing. Currently an implementation of a CD graph grammar system is planned. There are problems that have to be addressed of both theoretical and practical areas.

As nearly all of the theoretical results available now are based on systems with components being string grammars, it seems important to find whether at least some of these results could be transferred into graph based systems. The results concerning the ability of a (small) number of context-free grammars to generate a language identical with the one produced by non context-free grammar seems especially desirable. It has already been proved for a special case of graph grammars [18] but an extension for other types of grammars seems possible.

From the practical point of view the choice of the most appropriate derivation mode enabling all grammars to contribute to the final design and the choice of active grammar are very important. Also the way the components of a system are defined is very important as it is possible to define several different grammar systems generating the same language (and thus the same class of designs) but some systems may be easier to use than others.

In this paper simple graphs and graph grammars were used. Yet in computer aided design many objects have hierarchical structure so we also plan to research a possibility of extending this method to hierarchical graph and hierarchical graph grammars.

References

1. Borkowski A., Grabska E., Nikodem P, and Strug B. Searching for Innovative Structural Layouts by Means of Graph Grammars and Esvolutionary Optimization, Proc. 2nd Int. Structural Eng. And Constr. Conf, Rome (2003).
2. E. Csuhaj-Varj, J. Dassow, J. Kelemen and Gh. Paun. Grammar systems. A grammatical approach to distribution and cooperation. Topics in Computer Mathematics 8. Gordon and Breach Science Publishers, Yverdon, 1994.
3. E. Csuhaj-Varj, J. Dassow, and Gh. Paun. Dynamically controlled cooperating/distributed grammar systems. Information Sciences, 69(1-2), pp. 1-25, 1993.
4. E. Csuhaj-Varj and Gy. Vaszil. On context-free parallel communicating grammar systems: Synchronization, communication, and normal forms. Theoretical Computer Science, 255(1-2), pp. 511-538, 2001.
5. E. Csuhaj-Varj: Grammar systems: A short survey, Proceedings of Grammar Systems Week 2004, 141-157, Budapest, Hungary, July 5-9, 2004.
6. J. Dassow, Gh. Paun, and G. Rozenberg. Grammar systems. In A. Salomaa and G. Rozenberg, editors, Handbook of Formal Languages, volume 2, chapter 4, pp. 155-213, Springer-Verlag, Berlin-Heidelberg, 1997
7. E.Grabska. Theoretical Concepts of Graphical Modelling. Part one: Realization of CP-graphs. Machine GRAPHICS and VISION, 2(1), pp. 3-38, 1993.
8. E.Grabska, Theoretical Concepts of Graphical Modelling. Part two: CP-graph Grammars and Languages. Machine GRAPHICS and VISION, 2(2),pp. 149-178, 1993.
9. E.Grabska, W. Palacz. Hierarchical graphs in creative design. MG&V, 9(1/2), pp. 115-123, 2000.
10. E. Grabska. Graphs and designing. Lecture Notes in Computer Science, 776 (1994).
11. E. Grabska, P. Nikodem, B. Strug. Evolutionary Methods and Graph Grammars in Design and Optimization of Skeletal Structures Weimar, 11th International Workshop on Intelligent Computing in Engineering, Weimar, 2004.
12. J. Kelemen. Syntactical models of cooperating/distributed problem solving. Journal of Experimental and Theoretical AI, 3(1), 1-10, pp. 1991.
13. C. Martn-Vide and V. Mitrana. Cooperation in contextual grammars. In A. Kelemenov, editor, Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems, pp. 289-302. Silesian University, Opava, 1998.
14. P. Nikodem and B. Strug. Graph Transformations in Evolutionary Design, Lecture Notes in Computer Science,vol 3070, pp. 456-461, Springer, 2004.
15. Gh. Paun and A. Salomaa editors, Grammatical models of multi-agent systems. Gordon and Breach, Amsterdam, 1999.
16. Rozenberg, G. Handbook of Graph Grammars and Computing by Graph. Transformations, vol.1 Fundations, World Scientific London (1997).
17. Rozenberg, G. Handbook of Graph Grammars and Computing by Graph. Transformations, vol.2 Applications, Languages and Tools, World Scientific London (1999)
18. M. Simeoni and M. Staniszkis. Cooperating graph grammar systems. In Gh. Paun and A. Salomaa, editors, Grammatical models of multi-agent systems, pp. 193-217. Gordon and Breach, Amsterdam, 1999.

An Adaptive Filter Mechanism of Random Number Generator in a Crypto Module

Jin Keun Hong[1] and Ki Hong Kim[2]

[1] Division of Information and Communication, Cheonan University,
115 Anse-dong, Cheonan-si, Chungnam, 330-740, South Korea
jkhong@cheonan.ac.kr
[2] Graduate School of Information Security, Korea University,
1, 5-Ka, Anam-dong, Sungbuk-ku, Seoul, 136-701, South Korea
hong0612@hanmir.com

Abstract. Yet, when using a real random number generator (RNG) with only a hardware component, as required for statistical randomness, it is quite difficult to create an unbiased and stable random bit stream. Although the hardware generating processor generates an output bit stream quickly, if the software filter algorithm is not efficient, the RNG consumed a relatively large amount of time. This factor becomes the limiting condition when the RNG is applied. Accordingly, this paper proposes an adaptive filter approach to ensure the model of software filtering in the RNG processor of a crypto module. Therefore, an adaptive model is proposed that applies a filter algorithm, thereby consuming less time than the conventional filter algorithm scheme.

1 Introduction

At present, ubiquitous computing is advocating the construction of massively distributed computing environments that sensors, global positioning system receives [1] [2]. As such, this case is significantly different from those contemplated by the canonical doctrine of security in distributed.

A H/W random number generator (RNG) uses a non-deterministic source to pro-duce randomness, and more demanding random number applications such as cryptography, a crypto module engine, and statistical simulation benefit from sequences produced by a RNG, i.e., a cryptographic system based on a hardware component [1]. As such, a number generator is a source of unpredictable, irreproducible, and statistically random stream sequences. A popular method for generating random numbers using a natural phenomenon is the electronic amplification and sampling of a thermal or Gaussian noise signal. However, since all electronic systems are influenced by finite bandwidth, $1/f$ noise, and other non-random influences, perfect randomness cannot be preserved by any practical system. Thus, when generating random numbers using an electronic circuit, a low-power white noise signal is amplified and then sampled at a constant sampling frequency. Yet, when using a RNG with only a hardware component, as required for statistical randomness, it is quite difficult to create an unbiased and stable random bit stream.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 575–582, 2006.

The studies reported in [3] [4] [5] show that the randomness of a random stream can be enhanced when combining a real RNG, a LFSR number generator, and a hash function. In previous studies about RNG schemes in the field of security, Fabrizio Cortigiani, et al. (2000) examined a very high speed true random noise generator and S. Rocchi and V. Vignoli (1999) proposed a high speed chaotic CMOS true random analog/digital white noise generator. Adel et al. (2001) approached design and performance analysis of a high speed AWGN communication channel emulator. A noise based random bit generator IC for applications in cryptography was also studied (Craig S, et al. 1998 [4]). However, the randomness of this combined method is still dependent on the security level of the hash function and the LFSR number generator. Our previous paper proposes a real RNG that combines a RNG and filtering tech-nique that is not dependent on the security level of the period. Therefore, controlling stable input voltage for a RNG is an important aspect of the design of the RNG. In particular, it is important that the hardware RNG offers an output bit stream that is always unbiased. In a previous study [5], Jin Keun Hong et al. proposed a combined method of a hardware component and a software filter algorithm. However, in the case of this algorithm, although hardware generating processor generates the output bit stream rapidly, if the software filter algorithm is not efficient, the RNG consumes relatively much time and this factor becomes the limiting condition when the RNG is applied. Accordingly, this paper proposes an effective approach to ensuring the method of software filtering in the RNG processor of a crypto module. Thus, to consistently guarantee the randomness of an output sequence from a RNG, the origin must be stabilized, regardless of any change of circumstance elements. Therefore, a RNG is proposed that applies a filter algorithm, thereby consuming less time than the conventional filter algorithm scheme. Additionally, computational quantity is analyzed when the filter algorithm is applied in this mechanism.

Hereinafter, section 2 reviews the framework of the RNG in a crypto module. Then, section 3 examines the filter algorithm of conventional model and introduces the proposed adaptive filter algorithm. Experimental results and conclusions are presented in sections 4 and 5, respectively.

2 Framework of RNG in a Crypto Module

A H/W random number generator includes common components for producing random bit-streams, classified as follows (in Fig. 1): characteristics of the noise source, amplification of the noise source, and sampling for gathering the

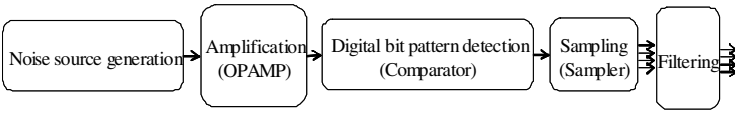


Fig. 1. Process of real random number generator in crypto moduel

comparator output [4] [6]. The applied noise source uses Gaussian noise, which typically results from the flow of electrons through a highly charged field, such as a semiconductor junction [7] [8] [9] [10]. Ultimately, the electron flow is the movement of discrete charges, and the mean flow rate is surrounded by a distribution related to the launch time and momentum of the individual charge carriers entering the charged field.

The Gaussian noise generated in a PN junction has the same mathematical form as that of a temperature-limited vacuum diode. The noise seems to be generated by the noise current generator in parallel with the dynamic resistance of the diode. The probability density $f(x)$ of the Gaussian noise voltage distribution function is defined by Eq. (1).

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{x^2}{2\sigma^2})} \tag{1}$$

where σ is the root mean square value of Gaussian noise voltage. However, for the designed Gaussian noise random number generator, the noise diode is a diode with a white Gaussian distribution. The power density for noise is constant with frequency from 0.1Hz to 10MHz and the amplitude has a Gaussian distribution. Noise comes from agitation of electrons within a resistance, and a lower limit on the noise present in a circuit is set. When the frequency range is given, the voltage of noise is decided by the factor of frequency. The crest factor of a waveform is defined as the ratio of the peak to the rms value. A crest value of approximately 4 is used for noise. However, for the proposed real random number generator, the noise diode has a white Gaussian distribution. The noise must be amplified to a level where it can be accurately thresholded with no bias using a clocked comparator. The noise must be amplified to a level where it can be accurately threshold with no bias by a clocked comparator. Although the rms value for noise is well defined, the instantaneous amplitude of noise has a Gaussian, normal, distribution.

$$V_n(rms) = \sqrt{4kTRB} \tag{2}$$

where k is Boltzmann constant $(1.38 \times 10E-23J/deg.K)$, T is absolute temperature (deg. Kelvin), B is noise bandwidth (Hz), R is resistor (Ohms). In Fig. 2, If $4kT$ is $1.66 \times 10E-20$ and R is $1K$, B is $1Hz$, then $V_n(rms) = \sqrt{4kTB} = 4nV/\sqrt{Hz}$. Noise comes from agitation of electrons within a resistance, and it sets a lower limit on the noise present in a circuit. When the frequency value is 10MHz, a crest value of rms value, 0.2mV is occasionally occurred.

3 Conventional Algorithm and Proposed Adaptive Filter Algorithm

3.1 Conventional Filter Algorithm

The conventional filter algorithm is applied in the next process of the output stream of the sampler to reduce the biased statistical randomness [5]. Establishing the optimum buffer size (32bits) and threshold level (γ) are support unbiased

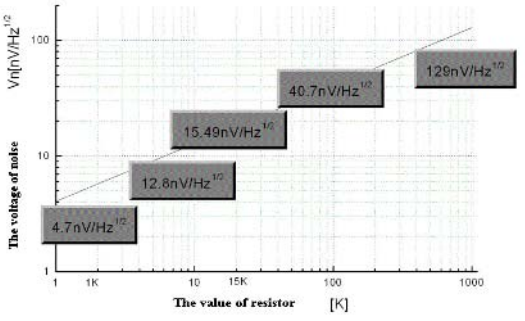


Fig. 2. Plot of noise voltage vs. resistor

and stable randomness. In conventional filter model, a static buffer memory of 32bits is used to buffer the pass data in the decision boundary and the threshold level for the P value is between 0.9995 and 1.0005.

When the static buffer (S) is fixed at 32bits, the half-value of the static length is 16bits. If the value of (Σ (the number of a pattern 1bit) / the half-value of the static length within the total length) is included in the threshold level, the decision will be the state of pass. In step 1, if the condition of pass is decided, this is added as pass data to the buffer memory. In steps 3 - 4, if fail is decided through the process of conventional filtering, this is decided into the decision process. The process is then completed when the size of the desired bit stream is gathered. The failed bits (32bits) are reduced by conventional filter (for example, the duty distribution of the bit stream 0 and 1 is normalized). In conventional model, the output bit stream is expanded by steps of 32bits, evaluated threshold level simultaneously. If the value of the duty cycle of the collected output bit stream, P, satisfies the condition of the threshold level, it is added 32bits (S) stream.

If P does not satisfy the condition of the threshold level, it is discarded 32bits stream. Through this filter process, unbiased characteristics of the output bit stream are guaranteed.

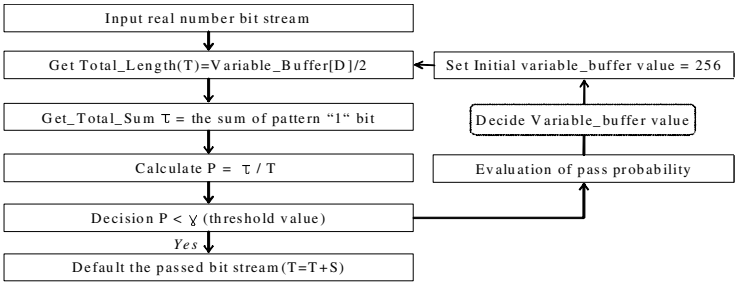


Fig. 3. Process of proposed filtering algorithm

3.2 Adaptive Proposed Filter Algorithm

The adaptive filter model is applied in the next process of the output stream of the sampler to reduce characteristics of biased bit stream. It is an efficient method to reduce the consumed time.

In Fig. 3, the static buffer (S) size is fixed at 32bits, but the variable buffer (D) size is variable. When it decide the size of variable buffer, it is used by the measure throught the evaluation of passed probability during setup time interval.

4 Experimental Results

A multiple bit stream of consecutive bits as the output from the RNG is subjected to a frequency test, et al. [12]. If any of the tests fail, the module then enters an error state. The statistical RNG test methods are used, on the basis of the statistical RNG randomness. When the RNG has an output bit stream of 256bits, the buffer window size is 32bits, As such, in the case of conventional filter model, 8 rounds need to be processed for the filter evaluation, as if the output bit stream (256bits) is divided into a window size of 32bits, this makes 8 fields. If P is not satisfactory for 6 fields out of the 8 fields, i.e., the level of passed probability is 75%, this can be presented as follow: [window size(discarded blocks of error state)] = [128(0), 128(2)] = [128(0), 64(0), 64(2)] = [128(0), 64(0), 32(1), 32(1)]

When the failed probability presents a probability that does not satisfy the significance level in 2^0 level consisting of 256 bits, the mother node is divided into the left child node and the right child node consisting of 128bits, respectively. In the case of two failed fields, two failed fields occur in the right child node, while zero out of the two failed fields occur in the left child node. If it is assumed that two of the failed fields in level 2^2 (64bit units) diverge into a failed field unit in the left child node and a failed field unit the right child node, in 2^3 level (32bit units), four nodes occur that are not failed fields, while two nodes occur as failed fields. Therefore, the six nodes about the nodes that do not occur as failed fields are not processed.

Accordingly, in the case of the combined method consisting of a hardware method and software filtering to create the output bit stream of the RNG, whereas given in buffer size 256bits, lower bound 32bits in passed probability 50%, conventional model needed 8 rounds to evaluate the 8 fields, the proposed model take 15 rounds of inefficient range in worst case and 7 rounds in best case. When increasing the passed probability, the round number of computation iterations is decreased. Also, given in buffer size 128bits, lower bound 32bits in passed probability 50%, general model needed 8 rounds to evaluate the 8 fields, the proposed model take 14 rounds of inefficient range in worst case and 6 rounds in best case. As decreasing the buffer size in fixed buffer length, the round number of computation iteration is decreased. Therefore, if passed probability is high, it is needed to increase the buffer size.

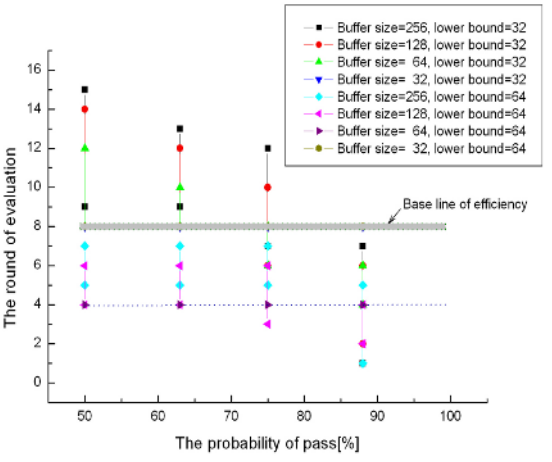


Fig. 4. Comparison of evaluation rounds according to passed probability

Table 1. Comparison of filtering loss rate efficiency between conventional model and proposed model according to passed probability (to obtain 2Mbits)

Loss rate and each mode	Lower bound = 32bits			
	50	63	75	88
Conventional model buffer size = 32	1.560	1.170	0.780	0.390
porposed model buffer size = 256, 128, 64	1.560	1.170	0.780	0.390

Then it considered that if P value of output bit stream of RNG in evaluation interval is satisfy the threshold level, i.e., the passed probability is about 88%, computa-tion quantities for evaluation of threshold level in conventional model are required 100probability during evaluation interval is reduced, its computation quantity is increased. Especially, when the passed probability is lower 75%, if it is decided the buffer size 256, it is not efficient. Therefore, from these results in Fig. 4, with regard to the passed probability and computation quantity, the performance of the proposed model is need to decide adaptive buffer size. If the passed probability is upper to 88%, then it is decided buffer size 256bits. The comparison of loss rate between conventional model and the proposed filter

Table 2. Comparison of filtering loss rate efficiency between conventional and proposed model according to passed probability (to obtain 2Mbits)

Loss rate and each mode	Lower bound = 64bits			
	50	63	75	88
Conventional model buffer size = 32	1.560	1.170	1.560	0.390
proposed model buffer size = 256, 128, 64	3.120	2.340	1.560	0.780

Table 3. Randomness of output bit stream by filter model (to obtain 2Mbits)

Test items	Threshold level value of randomness	Pass rate
Frequency test	< 3.841	100% (20/20)
Serial test	< 5.991	100% (20/20)
T-serial test (3 blk)	< 9.488	100% (20/20)
Poker test (3 blk)	< 14.067	90% (18/20)
Autocorrelation test	< 0.05	100% (20/20)

model is presented in Table 1 and Table 2, where the output bit streams of RNG are gathered about 2Mbits. In condition of passed probability 88% and lower bound 64bits, the loss rate is increased than that of conventional model. But the consumed time of proposed model is less than that of the conventional model. If the lower bound is decided 64bits and the failed field is existed in 64bits level, then the 64bits (32bits left and 32bits right field) are discarded. But if the lower bound is decide 32bits, then the 64bits is evaluated in each 32bits left field and 32bits right field. Therefore, if the passed probability is lower to 75%, then it is needed to decide the lower bound 64bits.

Table 3 represents various randomness tests along with their references, typical pass/fail boundaries, the measured average of iteration tests based on 2MB samples, and whether the sequence passed all the trial tests. The test results are extremely positive: i.e. the proposed system passed all the trial tests.

5 Conclusions

The conventional papers are proposed a real RNGs that combine an RNG and filtering technique that is not dependent on the security level of the period. Therefore, it is important that the RNG hardware offers an output bit stream that is always unbiased. Even though the hardware generating processor generates an output bit stream quickly, if the software filter algorithm is inefficient, the RNG becomes time consuming, thereby restricting the conditions when an RNG can be applied. Accordingly, this paper proposes an adaptive method of software filtering for an RNG processor in a crypto module. Therefore, a RNG is proposed that applies a filter algorithm that is less time-consuming than conventional filter algorithm scheme. In addition, the computational burden is also analyzed when applying the filter algorithm.

References

1. H. Alireza and V. Ingrid. High-Throughput Programmable Crypto-coprocessor. *IEEE Computer Society*, 2004.
2. A. M. Jalal, R. Anand, C. Roy, and M. D. M. Cerberus. A Context - Aware Security Scheme for Smart Spaces. *IEEE PerCom'03*,2003.
3. Robert Davies. True Random Number. http://webnz.com/robert/true rng.html.

4. C. S. Petrie and J. A. Connelly. A Noise-Based Random Bit Generator IC for Applications in Cryptography. *ISCAS'98*, 1998.
5. Jin Keun Hong, et al. Design of Real Random Number Generator. *CSN'03*, 2003.
6. M. Delgado-Restituto, F. Medeiro, and A. Rodriguez-Vasquez. Nonlinear Switched-Current CMOS IC for Random Signal Generation. *IEE Electronic Letters*, vol. 29, 1993.
7. http://www.io.com/ ritter/RES/NOISE.HTM.
8. http://www.clark.net/pub/cme/P1363/ranno.html.
9. http://webnz.com/robert/true rng.html.
10. Boris Ya, Ryabko and Elena Matchikina. Nonlinear Switched-Current CMOS IC for Random Signal Generation. *IEE Electronic Letters*, vol. 29, 1993.
11. M. Delgado-Restituto, F. Medeiro, and A. Rodriguez-Vasquez. Fast and Efficient Construction of an Unbiased Random Sequence. *IEEE Trans. on Information Theory*, vol. 46, no. 3, 2000.
12. Diehard. http://stat.fsu.edu/ geo/diehard.html. 1998.

Computer Analysis of the Sensitivity of the Integrated Assessment Model MERGE-5I*

Vyacheslav Maksimov[1], Leo Schrattenholzer[2], and Yaroslav Minullin[2]

[1] Institute of Mathematics and Mechanics, Ural Branch of RAS,
16 S.Kovalevskoi Str., 620219 Ekaterinburg, Russia
maksimov@imm.uran.ru
http://www.imm.uran.ru
[2] International Institute for Applied Systems Analysis,
A-2361, Laxenburg, Austria
leo@iiasa.ac.at, minullin@iiasa.ac.at
http://www.iiasa.ac.at

Abstract. This paper reports on an application of a large-scale nonlinear optimization model to the analysis of environmental problems. The authors present first results of the initial stage of the study, a sensitivity analysis of the model's input parameters. This analysis is part of the more comprehensive study Analysis of Economic Implications of Russia's Participation in the Kyoto Protocol, undertaken by a collaborative group of researchers from IIASA and Russian research institutes.

1 Introduction

Following intensive and heated public discussions, the President of the Russian Federation signed the Federal Law "on the ratification of the Kyoto Protocol to the United Nations Framework Convention on Climate Change" on 4 November 2004. This law led to the approval and the subsequent ratification of the Protocol by Russia. Russia's ratification was the final step of fulfilling all requirements for the Protocol to enter into force. The 90-day period between the fulfillment and entering into force — as specified in the Protocol — ended on 16 February 2005, which therefore marks the date of entry into force. Still, the debate about future costs and benefits of being a Party to the Kyoto Protocol has all but ended, and to the present day, the advantages and disadvantages of Russia's participation in the Kyoto Protocol are put forward by proponents and opponents respectively.

This situation was the motivation for a collaborative study involving two IIASA Programs (Energy and Dynamic Systems) and several energy research

* The work was supported in part by the International Institute for Applied Systems Analysis, Laxenburg, Austria. For the first author the work was also supported by the Russian Foundation for Basic Research (Project 06-01-00359) and by the Program on Basic Research of the Russian Academy of Sciences in Changes of Natural Terrestrial Objects in Russian Zones of Intense Technogenic Influence (Project 3 10002-251/ П-13/196-018/300503-340).

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 583–590, 2006.

groups of Russia who have set out to examine, in a formal framework, the implications of Russia's participation in the Kyoto Protocol from an economic and an environmental perspective.

The aim of IIASA aspiring to contribute to the debate is to formalize the debate by quantifying the main drivers of economic growth, energy consumption, and environmental impact. As an instrument, IIASA has selected the MERGE model [1, 2, 3], one of the most well known and most widely accepted E3 (energy-economy-environment) models.

In this paper, we focus on the first step of the assessment. This first step is mainly methodological, i.e., a sensitivity analysis, which is intended to characterize the input parameters by their impact on the output of the model.

Section 2 will be devoted to a brief description of the MERGE model and its modifications performed at IIASA. Section 3 will describe the approach to sensitivity analysis, followed by a presentation of the results of the sensitivity analysis.

2 The MERGE-5I Model

2.1 A "Nutshell" Description of MERGE

The global optimization model MERGE [3] describes the interaction between macroeconomic production, the energy system (demand and supply), pollutant emissions, and climate change. The model consists of three logical parts: a macroeconomic module, an energy supply part, and a climate module. It combines a top-down description of the economy and energy demand with a bottom-up description of the energy sector.

The *macroeconomic module* defines an inter-temporal utility function of a single representative producer-consumer in each of the model's world regions, which is then maximized by MERGE subject to given constrains. The main variables of this module are the production factors capital stock (K), available labor (L), and energy inputs (electric, EN and non-electric, NN), which together determine the total output of an economy according to a nested CES (constant elasticity of substitution) production function.

$$\mathrm{YN}(t,\mathrm{RG}) = [a(K(t,\mathrm{RG})^{\mathrm{KPVS}} L(t,\mathrm{RG})^{1-\mathrm{KPVS}})^{\rho} +$$

$$b(\mathrm{EN}(t,\mathrm{RG})^{\mathrm{ELVS}} \mathrm{NN}(t,\mathrm{RG})^{1-\mathrm{ELVS}})^{\rho}]^{\frac{1}{\rho}},$$

$$\rho = 1 - \frac{1}{\mathrm{ESUB}},$$

where t — time; RG — region; a — scale parameter; KPVS — share of capital (capital value share) in capital—labor pair; ELVS — share of electric energy (electric value share) in the electric—non-electric energy pair; ESUB — elasticity of substitution between capital—labor and electric—non-electric energy. The optimal quantities of the production factors are determined by their relative prices.

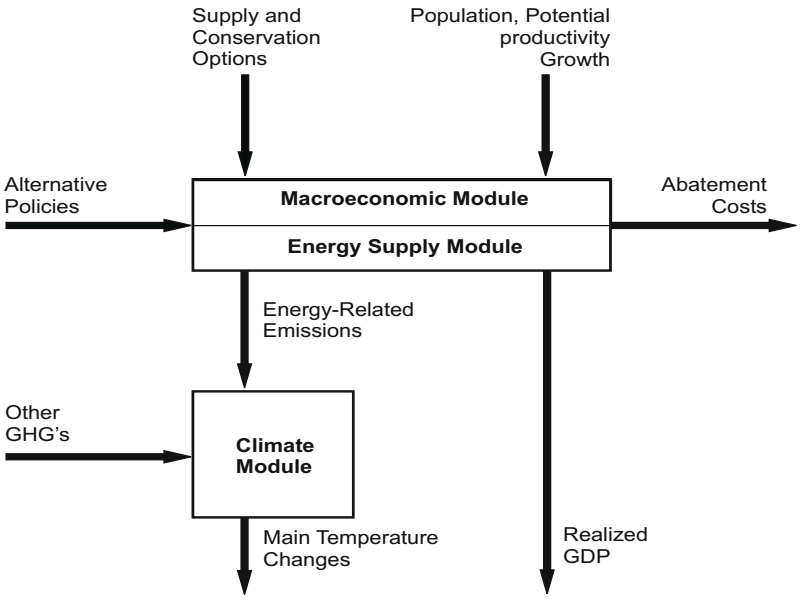


Fig. 1. An overview of MERGE

The core of the *energy module* is a comparatively simple Reference Energy System (RES) describing the technological options available to supply the energy needed as a production factor.

The *climate module* takes greenhouse gas (GHG) emissions, converting them systematically into atmospheric GHG concentrations and temperature change. The equations for calculating the radiative forcing and the temperature change are derived from the IPCC Third Assessment Report [6]. For CO_2, the radiative forcing is proportional to the logarithm of the ratio of the current to the initial level of atmospheric concentrations. The outputs of the climate module include trajectories of GHG emissions, atmospheric concentrations, and temperature change (Fig. 1).

MERGE was designed as an integrated-assessment model (IAM) to study global GHG mitigation scenarios and to conduct cost-benefit analysis. For the given purpose of our analysis, IIASA-ECS amended the original MERGE 5 model. The changes relative to the original MERGE 5 model are described in the following subsection.

2.2 IIASA Extensions of MERGE

In order to be able to model the important players in the Kyoto Protocol, the two MERGE regions CANZ (Canada, Australia, and New Zealand) and EEFSU (Eastern Europe and Former Soviet Union) were split into the four

regions Canada, ANZ (Australia and New Zealand), EEU (Eastern Europe) and FSU (Former Soviet Union). The model now includes 11 world regions[1].

Also, all six "Kyoto gases" are now included in the dynamics determining emission permit trade in the First Commitment period, CH_4 leakages from natural gas pipelines have been included, and the limits on sequestration from forest management as given in the Marrakech Accord were included.

The Clean Development Mechanism (CDM) is one of the Kyoto Protocols "flexible mechanisms", designed to reduce the economic costs (and to thus increase the global efficiency) of greenhouse gas abatement. Equations describing the CDM mechanism and a price-responsive CDM supply were incorporated into the model. As to quantities and prices of CDM projects, we used an interpolation of two supply curves reported by Point Carbon [5].

As to model input data, MERGE-I now includes recent information on expected economic growth and energy consumption of the complying regions. In the same spirit, the power sector options of these regions were restricted for the year 2010 (relative to the REF scenario) to avoid the possibilities of unrealistically high build-up rates of power plants. For Japan, we have additionally assumed that no more LNG terminals will be available by 2010 than in the Reference scenario and that therefore natural-gas imports in 2010 must not exceed those given by the Reference scenario (see also [4]).

3 Sensitivity Analysis

To analyze the consequences of assuming different geopolitical scenarios guiding the implementation of the Kyoto Protocol, we formulated two "limiting cases". One limiting case is a Reference scenario (REF or R0) without GHG emission constraints. The costs of GHG abatement are calculated relative to this scenario. The second limiting case is an extreme compliance scenario (DOM or R1) in which the Parties to the Kyoto Protocol must comply with their "Kyoto limits" by domestic measures only. Relative to this scenario, benefits of GHG emission trading (including CDM) can be calculated. The third limiting case, so called "Full trade" will be implemented as one of the next steps of the collaborative group's work.

In our scenarios, we focus on the Former Soviet Union (FSU), assuming that Russia's indicators describe the evolvement of the economy-energy-environment (E3) system of the whole FSU region with reasonable accuracy. For the future, we plan to define Russia as a separate region.

We selected the arithmetic difference between realized GDP according to the reference case (REF) and that of the domestic measures case (DOM) as the most important result, and we use the term GDPLoss to refer to it.

$$\mathrm{GDPLoss} = \mathrm{RlzGDP}_{\mathrm{REF}} - \mathrm{RlzGDP}_{\mathrm{DOM}}.$$

[1] All regions follow the same convention as EIA-USDOE's International Energy Outlook (2004), with the only exception that we included the new Baltic EU members Estonia, Latvia, and Lithuania in EEU (and not in FSU).

This indicator shows the value of GDP that the country will lose or gain as a consequence of observing the Kyoto limits. Note that even in the "domestic measures only" case, realized GDP of the FSU will depend on the development of the E3 system in other world regions. For example, a possible decrease of Western Europe's (WEUR) GDP due to observing the Kyoto constraints would affect energy imports from the FSU, what in turn will result in a slight decrease of FSU's GDP. In general, fluctuations of GDPLoss are expected to occur within the time horizon of our scenarios. One of the most distinctive results of our model runs is the point in time, when these fluctuations end and a clear increase in the GDPLoss function will occur. This point in time describes a situation in which the FSU will have exhausted its reserves of "hot air", and measures aimed at a restructuring of the industry and energy sector towards a low-emission system will be initiated.

The following model parameters were included in our sensitivity analysis. The parameters ESUB, KPVS, ELVS, KGDP of the macroeconomic production function (described in Section 2); the "autonomous energy efficiency increase" (AEEI); the annual depreciation rate (DEPR); the international oil price (INTPR); the oil-gas price differential (OGPD); the maximal annual decline factor for the capacities of electric and non-electric technologies (DECF); the maximal market share for electric and non-electric technologies (NSHF); the coefficients describing the energy-intensive sectors (REIS); and parameters quantifying restrictions on abatement measures, namely, abatement quantity multipliers (ABMLT), abatement limits at alterative cost levels (ABLIM), and limits on sinks forestration (AppendixZ). All these parameters are defined for each model region. The parameters AEEI and REIS — in addition to their region-dependency — also depend on the time period. The parameter ABMLT depends only on the time period and has four indices: GHG (type of greenhouse gas), ABX (abatement cost index), TP (time period), and RG (region).

We begin our presentation of results with an illustration of the impact of changes in one of the most powerful driving factors, the parameters AEEI (Fig. 2.)

As a rule, each parameter V (where applicable) was varied (for FSU and simultaneously for the pair of scenarios R0 and R1), within the interval $[0.7 * V_0, 1.3 * V_0]$ with a step size of $0.1 * V_0$, which is the initial value of the parameter. An exception was the parameter DECF, which equals 0.98 in the initial model. DECF was varied within the interval $[0.9, 1.0]$ with a step size of 0.01. Since REIS equals 1 for all regions after 2050, this parameter was varied only for the time periods 2005 through 2050. Let us note in passing that changing PNREF (reference price of non-electric energy) requires an appropriate change of PEREF (reference price of electric energy) as well.

For each parameter V, the maximum relative deviation of GDPLoss(FSU) with respect to the initial configuration of the model was computed at each time period:

$$\mathrm{IS}_V(\mathrm{TP}) = \frac{\max_i \left| \mathrm{GDPLoss}_{V_i}(\mathrm{FSU}, \mathrm{TP}) - \mathrm{GDPLoss}_0(\mathrm{FSU}, \mathrm{TP}) \right|}{\mathrm{RlzGDP}_{\mathrm{R0}}(\mathrm{FSU}, \mathrm{TP})}.$$

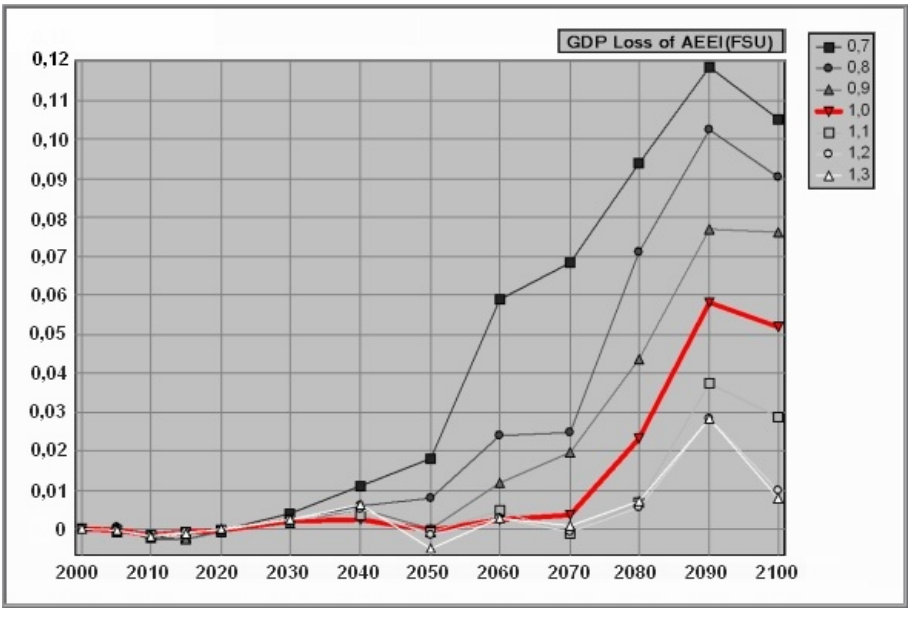


Fig. 2. Dynamics of GDPLoss(FSU) (trln.USD) under variation of AEEI(FSU). The legend shows the corresponding coefficients applied to all FSU components of the parameter

Here i reflects the index of observed parameter and IS stands for "instant sensitivity", which we use to define the so-called C-norm of IS_V in the following way (NS stands for "continuous-norm sensitivity"):

$$\mathrm{NS}_V(t) = \|\mathrm{IS}_V(\cdot)\|_{C[2005,t]} = \max_{\mathrm{TP}\in[2005,t]} \mathrm{IS}_V(\mathrm{TP}).$$

With the help of this formula, we can summarize the results of our sensitivity analysis as shown in Fig. 3.

Fig. 4 displays the patterns of comparative sensitivity of GDPLoss(FSU) to variation of the above listed parameters for different time periods. The percentage is calculated by the formula:

$$\mathrm{CS}_V(t) = \frac{\mathrm{NS}_V(t)}{\sum\limits_V \mathrm{NS}_V(t)} 100\%.$$

Here CS stands for "comparative sensitivity". This diagram may be treated as cross-sections of the graphs shown in Fig. 3 at certain time moments, and it will be noticed that the patterns change in time. This means that at different time periods the model is most sensitive to different parameters.

This sensitivity analysis identifies the degree of influence that the selected parameters have on the most important model result. Among these parameters,

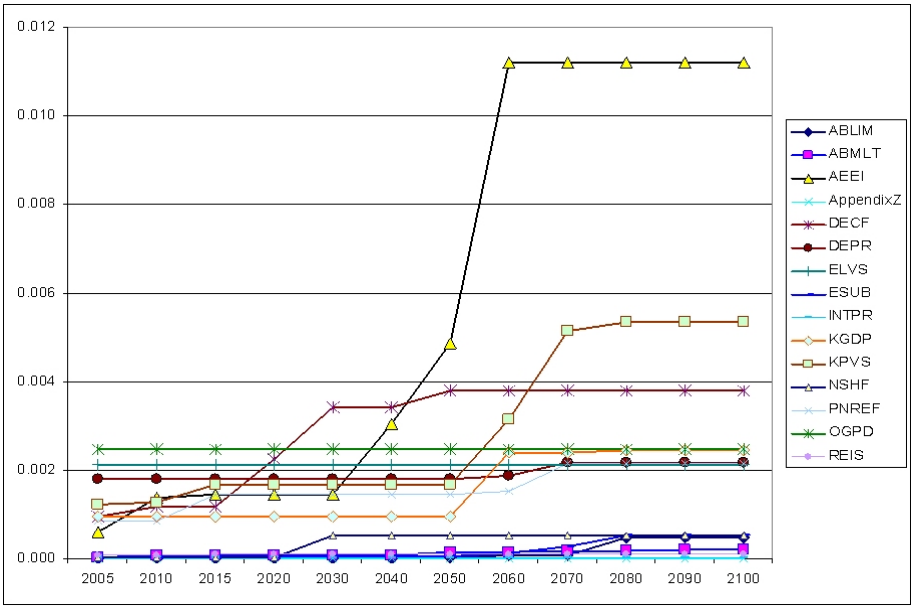


Fig. 3. C-norm of maximum relative deviations of GDPLoss(FSU) under variation of different input parameters

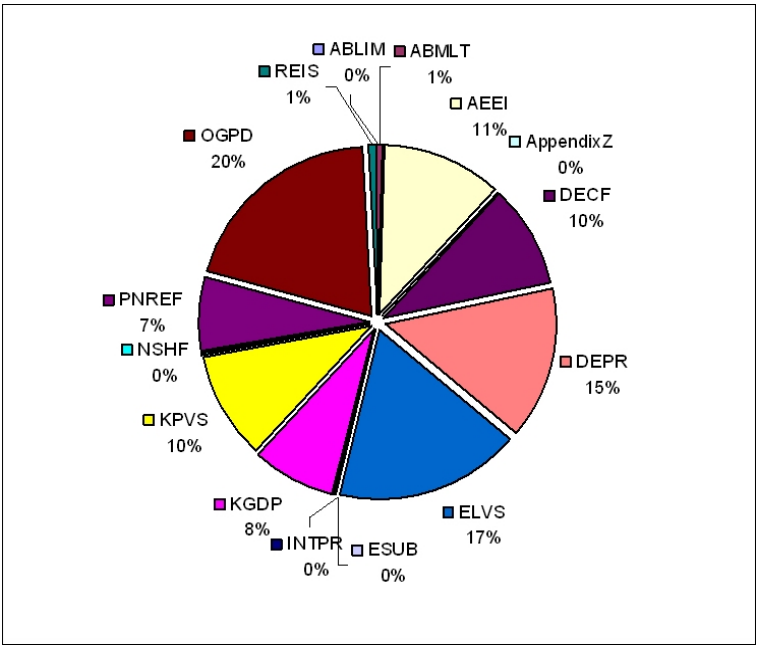


Fig. 4. Pattern of comparative sensitivity of GDPLoss(FSU, 2010) to variation of the parameters

the model is most sensitive to variations of AEEI (autonomous energy efficiency improvement) and KPVS (capital value share). Some of the parameters (such as ABLIM, ABMLT, AppendixZ, INTPR, ESUB, NSHF, REIS) have either negligible effects on FSU or no effect at all. The latter case usually reveals those constraints that are not binding for the region.

In addition to these main results, we have observed some noteworthy behavior of the model results. For example, our analysis shows that setting DECF = 1.0 (no decline of the technologies) generates a kind of unstable behavior of GDPLoss that is distinct from all the other values of DECF less than 1.0. This means that the unit value of DECF is unnatural for the model.

4 Conclusions

The results of our sensitivity analysis carried out in the framework of ongoing research identify the most sensitive input parameters of the model. This information will serve as a basis for future work, especially for setting up of scenarios, a task that involves the determination of the most accurate values of the input parameters. One example for which an accurate value is particularly important is *energy efficiency* - the most sensitive parameter according to presented above results. A separate study of *the Energy Efficiency in Russia* has been initiated with the aim of estimating the dynamics if this indicator considering technological and structural changes together with peculiarities of Russia's E3 (energy-economy-environment) system. Other parameters, such as the capital value share (KPVS) and the electricity value share (ELVS), etc. could be estimated on the basis of statistical data. It is planned to produce additional reporting in more detail on the results of described research. They will be announced at IIASA website.

References

1. Manne, A., Richels, R.: Buying Greenhouse Insurance: The Economic Costs of Carbon Dioxide Emission Limits. MIT Press, Cambridge, MA (1992)
2. Manne, A., Mendelson, R., Richels R.: MERGE-A Model for Evaluating Regional and Global Effects of GHG Reduction Policies. Energy Policy **23(1)** (1995) 17–34
3. Manne, A., Richels, R.: MERGE: A Model for Evaluating the Regional and Global Effects of GHG Reduction Policies. `http://www.stanford.edu/group/MERGE/` (2004)
4. Schrattenholzer, L., Totschnig, G.: Economic Analysis of Imperfect Implementations of the Kyoto Protocol. Final Report on the TEPCO-IIASA Collaborative Study submitted to the Tokyo Electric Power Company, Japan (2004)
5. Tangen, K., Atle C. C., Skogen, A., Roche, I., Imperfect implementation of the Kyoto Protocol. Draft Report to TEPCO-Environment (2004)
6. IPCC: Climate Change 2001: The Scientific Basis; Contribution of Working Group I to the Third Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press. `http://www.grida.no/climate/ipcc_tar/` (2001)

Hierarchical Classifier

Igor T. Podolak*, Sławomir Biel, and Marcin Bobrowski

Institute of Computer Science, Jagiellonian University,
Nawojki 11, Kraków, Poland
uipodola@theta.uoks.uj.edu.pl

Abstract. Artificial Intelligence (AI) methods are used to build classifiers that give different levels of accuracy and solution explication. The intent of this paper is to provide a way of building a hierarchical classifier composed of several artificial neural networks (ANN's) organised in a tree-like fashion. Such method of construction allows for partition of the original problem into several sub-problems which can be solved with simpler ANN's, and be built quicker than a single ANN. As the sub-problems extracted start to be independent of one another, this paves a way to realise the solutions for the individual sub-problems in a parallel fashion. It is observed that incorrect classifications are not random and can be therefore used to find clusters defining sub-problems.

1 Introduction

In data mining the given data sets composed of patterns are described with models. In a *predictive model* the output variable Y is expressed as a function of other *explanatory* variables X. For a categorical Y variable, the task is a *classification task* [2, 11]. The possible approaches, *e.g.* perceptrons, linear discriminant models, decision trees, various clustering methods, Naive Bayes model, neural networks (ANN's) etc., vary greatly in terms of *accuracy* and *explanatory power*. ANN's provide high accuracy, but predictions lack any reason why a particular answer was given, whereas decision trees give answers easy to understand by humans but do not generalise well. On the other hand it is hard to find an optimum ANN architecture for a problem, at least it is a lengthy process whose successful outcome depends more on experience and luck, than on clear rules. Clearly there is a need for a classifier easy and quick to build, accurate, with easy to explain predictions, fit for parallel implementation.

In this paper we aim at proposing a methodology of building a hierarchical classifier (HC) composed of several small ANN's, each constituting an easy to build *weak* classifier. Composition of weak classifiers' predictions provides for a more accurate one [3]. Each such classifier divides the problem into sub-problems whose training is independent, therefore fit for parallel implementation. From some of the classifiers (mainly these near the tree root) rules would be extracted providing for the predictions to be explainable.

* Research was funded by grants KBN Grant 3 T11C 054 26, and Jagiellonian University's "Multiagent systems".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 591–598, 2006.

The paper is organised as follows: first the problem and the model used are described, then the actual algorithm is depicted, followed with analyses of the algorithm. We also apply the proposed HC to the problem of electric power consumption prediction. The paper ends with conclusions.

2 Problem Definition and Model

A *classifier* may be defined as a function Cl

$$Cl : X \ni x \longrightarrow p \in P \tag{1}$$

that assigns each example x, defined with a vector of features, to a class p from a finite number of possible classes P. Cl is a realisation of an unknown function $F(\cdot)$ that is defined with the training data set D that comprises of pairs (x, p) where x is a vector of features, and p is the class that x belongs to.

It is possible to build an ANN that realises $F(\cdot)$ with accuracy less than any $\epsilon > 0$ [4]. On the other hand, ANN's that better assign classes to training examples, need more neurons and have lower generalisation level, which is defined as the ability of correct classification of examples that where not used during training. It is possible to construct *committee machines* in which responses from several predictors are combined. Each predictor, may have a different starting point, different training subset, *etc.*, therefore the overall combined answer may give better generalisation (see [4] for a discussion).

We propose to construct a tree classifier with a classifier Cl_i at each node

$$Cl_i : D_i \ni x \longrightarrow p \in P_i \tag{2}$$

where $P_i = \{p_{i1}, \ldots, p_{ik}\}$ is the set of classes of examples from $D_i \subset D$, and D is the original set of examples defining the whole problem. To have a quick algorithm, a small ANN is used to realise Cl_i, therefore the classifier is a *weak* one, *i.e.* it has not the perfect accuracy, but still a good generalisation rate. Some of the classes are confused with each other more frequently, therefore they are combined into groups, using the confusion matrix analysis and a merging algorithm as described below. If m groups of classes were formed,

$$Q_{ij} = \{p_{il} \in P_i | l = 1, \ldots, n_{Q_{ij}}\} \quad j = 1, \ldots, m \tag{3}$$

where $n_{Q_{ij}}$ is the number of classes in Q_{ij} and which form a set of groups Q_i

$$Q_i = \{Q_{ij} | j = 1, \ldots, m\} \tag{4}$$

then the original classifier Cl_i is replaced with classifier $Clmod$

$$Clmod : D_i \ni x \longrightarrow q \in Q_i \tag{5}$$

In other words, the new classifier $Clmod$ answers *not* with the class from the original problem, but tells which subset Q_{ij} the actual class most probably belongs to. The data set D_i is divided into subsets D_{ij} corresponding to groups Q_{ij}, and new ANN classifiers are built. The leaf node ANN's classify into original problem classes. Tree is built recursively and the algorithm stops when satisfying accuracy is achieved.

3 The Hierarchical Algorithm

At each levelof the tree classifier, the original problem is subdivided into sub-problems. At each node an ANN classifier is trained, but to save time and obtain better generalisation, ANN's are trained only as *weak classifiers*, *i.e.* such that classify examples only a bit better than the average (minimal correct classification rate grows at each level). Thanks to that, the networks in all the nodes can be kept small resulting in short learning time. The algorithm partitions the whole input space into sub-problems through analysis of the confusion matrix.

3.1 Confusion Matrix Based Problem Partition

Since classifiers are weak, some of examples are classified incorrectly. Nevertheless, we postulate that classification, although incorrect, is not random, *i.e.* if an example from class A (a label from the training set) gets classified as, say, class B, this means that classes A and B are *similar*. This is a *clustering* effect. Therefore, we partition the problem at this node (the problem is given by the examples used to train the ANN at this node) into sub-problems by selecting groups of examples which *frequently* get classified similarly. Each such group defines a new sub-problem for which a new ANN would be trained.

This is done by inspecting the *confusion matrix* M generated at the end of ANN training. Therefore, if we have the set of training examples D

$$D = \{x_1, x_2, \ldots, x_N\} \tag{6}$$

and a set of classes K that these examples can be assigned to

$$P = \{p_1, p_2, \ldots, p_M\} \tag{7}$$

and functions

$$\varphi(x_i) = p_k \text{ example } x_i \text{ is from class } p_k \tag{8}$$

$$\alpha(x_i) = p_k \text{ example } x_i \text{ was classified by the ANN as from class } p_k \tag{9}$$

then the confusion matrix M can be defined as

$$M[i][j] = a \tag{10}$$

where a is the number of examples from class p classified as q, *i.e.* the number of elements in the following set

$$\{x_i \in D \mid \varphi(x_k) = p_i \wedge \alpha(x_k) = p_j\} \tag{11}$$

A perfect classifier would have non-zero elements only on the diagonal of matrix M. On the other hand, an ANN which is not perfect, but still better than majority voting type classifier, confuses examples from a group of classes, while some classes get easily separated. This can easily be seen in the confusion matrix where the examples from some groups of classes are frequently inter-classified, and the corresponding elements of M are non zero, while some are never mistaken and the elements of M are zeroed. This conveys the information about clusters of examples easily mistaken by an ANN. With that information the groups can easily be found, see Fig. 1.

3.2 Construction of the Classifier Tree

For each of the desired output class p, basing on confusion matrix M, we find all the classes that are mistaken (corresponding to non-zero elements in each of the matrix M's row, see Fig. 1). Each group consists of classes that examples from desired output class (corresponding to matrix row) are classified as. In this way *groups of classes* are produced. In order not to have too many classes in any of the groups, only k maximum values are taken (k is usually 1/4 of classes).

Definition 1. *The generalised accuracy is a function that gives the probability that examples from a given class p are classified by an ANN into a group that to which p belongs.*

The normally used notion of accuracy, hereafter referred to as *standard accuracy*, is a special case of a generalised accuracy in which all the classes are singletons.

If an ANN has the standard accuracy of 50% for examples from a class p_1, but 95% of examples classified as p_1 actually belong to a group $p_{i_1} \cup p_{i_2} \ldots \cup p_{i_l}$, then by grouping all these examples into one, the generalised accuracy would be 95%. This enables us to construct a strong classifier out of weak ones.

Each of the groups correspond to a new sub-problem which has *less* output classes than the original one. Therefore, we postulate that this problem is easier to solve. A new classifier is constructed for each of the sub-problems. The objective of each of the sub-classifiers would be to distinguish among classes that were similar (therefore they were grouped together) and harder to discern by the original classifier. In this way the classifier tree represents a *hierarchical partition* of the original problem. The partition is continued up to the moment when satisfying accuracy rate is achieved.

During actual classification, an example would be classified recursively by classifiers at each level. Each classifier would decide to which of the groups the example does belongs, therefore selecting a sub-classifier that the example is passed to. Leaf classifiers would return the actual answer.

3.3 Group Merging

With the partition algorithm described, at each tree level a separate class would be constructed for each of the possible classifications. This would result in an enormous number of classifiers. Recursive equation for number of classifiers can be written as:

$$T(x) = x * T(\frac{1}{q} * x) \tag{12}$$

so using "master method" [12] for solving recursive equation we can approximate number of classifiers as

$$M^{\log qM} \tag{13}$$

But each of the trained classifiers finds regularities within the training set, *i.e.* some classes are similar to other. Therefore we propose to reduce the number of classes by using a Sequential Agglomerative Hierarchical Nesting (SAHN) [7] which is a bottom-up clustering algorithm. Each group is represented as a binary

valued vector with bits set in positions that correspond to classes occurring in that group. SAHN finds two closest groups, *i.e.* output class vectors, and combines them together. Similarity is defined with the Hamming distance

$$H(x, y) = \text{number of positions that vectors differ on} \tag{14}$$

Groups are merged with SAHN until the number of classes in any of the groups does not exceed a threshold. For a threshold we use $\lambda * n$ where n is the number of all classes, and $\lambda \in (0, 1)$. We have used $\lambda = 0.5$. For higher λ's we obtain less sub-classifiers, but with more classes in each of them. Bigger groups would resemble more the original problem, and therefore less would be achieved by partitioning. Only for the resulting groups of classes new classifiers would be constructed, *i.e.* we decrease the number of them making the whole tree construction feasible. The resulting hierarchical classifier for the zoo problem [1] is shown in Fig. 1.

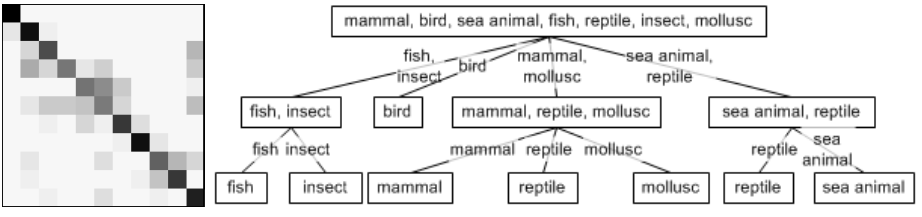


Fig. 1. The confusion matrix with cells shaded accordingly to the number of examples classified, and a hierarchical classifier for the zoo problem [1]

3.4 Rule Generation for Selected Networks

Rule extraction from trained ANN's algorithm is based on FERNN – Fast Extraction of Rules from Neural Networks[9]. The following steps need to be performed to generate a rule classifier (RC):

1. Train a neural network
2. Build a decision tree that classifies the patterns in terms of the network hidden units activation values
3. Generate classification rules

Neural network training. ANN's with a single hidden layer is trained to minimise the augmented cross-entropy error function (15) Each pattern x_p is from one of the C possible classes. t_{ip} is the target value for pattern p ($p = 1, 2, \cdots, P$) at output unit i. S_{ip} is the ANN's output unit value. J is the number of hidden units. v_{ij} is the weight of the connection from hidden unit j to output unit i and w_{jk} is the weight of the connection from input unit k to hidden unit j.

The ANN is trained to minimise the augmented cross-entropy error function:

$$\theta(w, v) = F(w, v) - \sum_{i=1}^{C} \sum_{p=1}^{P} [t_{ip} \log S_{ip} + (1 - t_{ip}) \times \log(1 - S_{ip})] \tag{15}$$

where $F(w, v)$ is a penalty function with positive parameters $\epsilon_1, \epsilon_2, \beta$. It is added to encourage weight decay.

$$F(w,v) = \epsilon_1 \sum_{j=1}^{J} \left(\sum_{i=1}^{C} \frac{\beta v_{ij}^2}{1+\beta v_{ij}^2} + \sum_{k=1}^{K} \frac{\beta w_{jk}^2}{1+\beta w_{jk}^2} \right) + \epsilon_2 \sum_{j=1}^{J} \left(\sum_{i=1}^{C} v_{ij}^2 + \sum_{k=1}^{K} w_{jk}^2 \right) \tag{16}$$

Penalty function causes irrelevant connections to have very small weights. Connections are cut beginning from the smallest one as long as network error has acceptable value. As a result the ANN is produced which can be easily converted to the RC. It is used for some nodes of the HC.

Construction of a decision tree. The decision tree is built using the hidden unit activations of correctly classified ANN patterns along with the patterns' class labels. The C4.5 [8] algorithm is used to build the decision tree. In the construction of the HC, we used WEKA's [5] J48 decision trees implementation.

Rule generation. A sigmoid function is used for hidden units activation. Node splitting conditions in the decision tree can be written as follows:

$$\text{if } \sigma\left(\textstyle\sum_{k=0}^{n_j} w_{jk} x_k\right) \leq S_v \text{ then LeftNode else RightNode}$$

By computing the inverse of the sigmoid function $\sigma^{-1}(S_v)$ for all node splitting conditions in a decision tree, we obtain conditions that are linear combinations of the input attributes of the data.

Below is a set of rules generated from the root classifier of the HF for the zoo problem from [1], where the objective is to classify an animal defined with 18 features into one of seven classes (mammals, birds, sea animals, fish, reptiles, insects, mollusks). One can see that the output of the classifier is frequently not a single class, but a group of classes found to be similar. Subsequent classifiers in the HF tree would find the actual classification.

```
if(+3.03 *"eggs" -2.14 *"milk" -1.0 *"legs" +3.84 *"tail"  <= 1.7) {
  if(-2.23 *"feathers" +3.56 *"eggs" -2.26 *"milk" +2.55 *"aquatic"
     -1.0 *"catsize" <= 2.88) {
       class group = "mammal OR sea animal"
   } else {
     if(-2.23 *"feathers" +3.56 *"eggs" -2.26 *"milk" +2.55
         *"aquatic" -1.0 *"catsize" <= 3.88) {
           class group = "fish OR insect"
       } else {
           class group = "reptile OR mollusk"
       }
   }
} else {
   if(+3.03 *"eggs" -2.14 *"milk" -1.0 *"legs" +3.84 *"tail"  <= 4.88) {
       class group = "bird"
   } else {
     if(+2.02 *"aquatic" -1.0 *"legs"  <= 0.02) {
           class group = "mammal OR sea animal"
```

```
        } else {
            class group = "fish OR insect"
        }
    }
}
```

4 Experiments

We have performed a number of experiments using benchmark datasets from the [1, 6] database and compared them with the results obtained by Rudy Setiono who used an algorithm that constructed an optimum ANN with one hidden layer [10]. The results obtained in [10] are from among the best obtained for these datasets. A comparison with our results is shown in Tab. 1. It can be seen that the proposed classifier built with several small ANN's gave better test results for all training sets.

Table 1. Results of experiments of some files available from the UCI Repository [1, 6] compared with results of Setiono's N2CS2 algorithm [10] (arrhythmia is missing from there). Results with better means are given in boldface.

	level 1		level 2		level 3		Setiono	
	train	test	train	test	train	test	train	test
audiology	88.4	90.4	87.1	90.8	91.6 ± 1.08	$\mathbf{91.2} \pm 3.17$	$\mathbf{95.5} \pm 1.14$	79.5 ± 1.61
arrhythmia	88.9	85.2	88.2	85.3	92.2 ± 1.41	90.3 ± 2.91		
primary tumor	68.0	67.9	71.3	71.7	$\mathbf{78.3} \pm 1.45$	$\mathbf{78.5} \pm 2.35$	62.1 ± 1.41	46.0 ± 1.37
vowel	82.5	81.9	89.7	89.2	$\mathbf{95.9} \pm 1.05$	$\mathbf{95.4} \pm 1.48$	94.3 ± 1.23	88.9 ± 1.46
zoo	91.7	95.5	97.6	97.1	97.6 ± 1.56	$\mathbf{97.1} \pm 2.31$	$\mathbf{100.0} \pm 0.00$	94.3 ± 1.46

We have also tried to use the proposed methodology to predict electric power consumption in a week's time horizon basing on past consumption. Input data consisted of the day, time, weekday, current consumption, consumption a year earlier, which coded together gave 127 input neurons. The output power prediction was divided into 57 classes each representing an interval of 250MW. Thanks to this the approximation problem was changed into a classification one. Only 50 neurons were used in each network.

An HC with 2 levels was trained. There were 6 second layer classifiers, each with 18 to 28 output classes representing intervals of the same width. The classification accuracy of the root classifier was only 47%, corresponding to a mean 443MW error. One should bear in mind that most of the incorrect classifications were into the neighbouring classes represented by a near diagonal confusion matrix. At the same time, the generalised classification accuracy, which gives the accuracy of classification examples into groups of classes, each represented with second layer ANN's was as high as 97%. The overall accuracy was still only 57% with overall approximation mean error of 353MW. This is better than most of the single ANN's we have tried for the same problem which used more input features, most with a much higher number of hidden neurons and layers.

5 Conclusions and Future Work

The proposed methodology seems to be promising since experiments show high accuracy obtained with the benchmark tests. It is possible to build the HC stepwise starting with a small (in term of the hidden neurons number) root ANN, then continue with similar ANN's in subsequent layers. The usual problem of ANN training is the choice of the correct number of hidden layers and neurons, but in our solution the whole HC is built with the same ANN's in all layers, which saves time needed for experimenting with sizes. All ANN's in a layer can be trained independently, therefore the training process is run in parallel.

We want to modify the HC architecture so that classification of a single example would not follow a single group from a classifier, but the example would be passed onto all the classifiers in the next layer that include the class assigned to the example in the previous ANN. In this way a committee would be built to enhance accuracy.

References

1. C. J. Merz C. L. Blake. Uci repository of machine learning databases.
2. P. Smyth D. Hand, H. Mannila. *Principles of Data Mining*. MIT Press, 2001.
3. E. Bax. Validation of voting committees. *Neural Computation*, 10(4):975–986, 1998.
4. S. Haykin. *Neural networks, a comprehensive foundation*. Prentice Hall, 1999.
5. Eibe Frank Ian H. Witten. *Data mining: practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.
6. M. Sokolic M. Zwitter. Primary tumor data set.
7. R. R. Sokal P. H. A. Sneath. *Principles of Numerical Taxonomy*. Freeman, San Francisco, 1973.
8. J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, 1993.
9. W. K. Leow R. Setiono. FERNN: An algorithm for fast extraction of rules from neural networks. *Applied Intelligence*, 12(1-2):15–25, 2000.
10. R. Setiono. Feedforward neural network construction using cross validation. *Neural Computation*, 13:2865–2877, 2001.
11. J. Friedman T. Hastie, R. Tibshirani. *The Elements of Statistical Learning*. Springer Verlag, 2001.
12. Ronald L. Rivest Thomas H. Cormen, Charles E. Leiserson. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

Knowledge-Based Runtime Prediction of Stateful Web Services for Optimal Workflow Construction

Zoltan Balogh[1], Emil Gatial[1], Michal Laclavik[1], Martin Maliska[1], and Ladislav Hluchy[1]

Institute of Informatics, Slovak Academy of Sciences, Dubravska cesta 9, 845 07, Bratislava
balogh@savba.sk
http://www.ui.sav.sk/

Abstract. This article proposes an approach for predicting runtime of web services (WS) with state - also called stateful web services. Estimating WS runtime is particularly critical during construction of composite WS workflows. Each workflow job must be scheduled in a way that the overall workflow run time will satisfy the overall workflow constrains. Such workflows are commonly used in Grids for connecting individual Grid WS to large, complicated and distributed applications. Prediction of WS run times optimizes scheduling and supports efficient use of grid resources. In our approach we propose to estimate expected WS run time based on invocation parameters of WS operations, states of resources maintained by a WS and properties of resources used as processing inputs for a WS. We adopt knowledge based approach where the history of WS operations is examined and a model is created and updated for each class and instance of a WS. Such WS run time prediction models can be then used by workflow schedulers to compute expected run times of a range of WS for the purpose of identifying the most appropriate WS for a given job within given constrains.

1 Introduction and State of the Art

This article deals with run time prediction of stateful web services (WS) for the purposes of optimal WS workflow construction and scheduling. Work presented in this paper is a part of effort to design a Knowledge Assimilation Agent (KAA) responsible for WS run time prediction in scope of the 6th IST FP called K-Wf Grid (Knowledge-based Workflow System for Grid Applications) [3].

At the present it is difficult to predict the behavior of a WS which carries out a job in a shared distributed computing environment such as Grid. The more complex the job is the more difficult is to predict the time needed by a web service to complete the execution. The run time of a WS is influenced by several aspects: internal WS performance, run times of other services utilized by WS during its operation and load of Grid resources which are used by WS during its execution. Additional complexity is introduced when constructing composite

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 599–607, 2006.

WS workflows constructed of several interconnected WS. For the purpose of this article we work with stateful web services, i.e. web services which are stateless in nature, but which manage resources with state. Such stateful WS are responsible for creation, maintenance and destruction of WS resources with state. There have been already attempts to predict application performance in Grids [6], however existing approaches do not deal with WS prediction in context of WS workflows. The main difference of our approach is that we investigate the service performance in dependence of invocation parameters and related resource properties.

The following chapter introduces the context of our work. We describe the way in which the WS are selected, planned and executed in Grid environment. In chapter 3 we describe the approach we use to measure performance of WS. Next chapter describes our approach to prediction of some WS performance measures. We conclude with short description of application in which the approach will be used and with plans for future work.

2 Optimal Construction of Grid Service Workflows

WS workflows are used in the K-Wf Grid project to interconnect separate WS into logically coherent Grid applications. The construction of workflow is supported by knowledge in order to optimize grid resource usage and comply with requirements defined by a user.

The Scheduler is a component of the Grid Application Control layer. It determines which instances of a set of alternative Web Service Operation instances will be selected in order to be executed in the current workflow execution. As input, the Scheduler receives the workflow description and searches for nodes that are mapped to a WS (a list of WS instances). Then, all possible alternative instances of each WS are compared considering different possible comparison criteria (metrics) to choose the best one in the given situation. The most important criterion used to choose the instance is the execution time. The runtime of individual WS operations is generated by the KAA using also method presented in this paper. Some other criteria (e.g., fault tolerance) may also be used. Finally, one of the instances is returned as output to the GWES - Grid Workflow Execution Service. Comparison criteria are inserted and taken from the Grid Organizational Memory (GOM). The criteria are usually experience-based and are derived from earlier executions of the WSs. The information collected from the previous runs contains execution time, reliability, availability rate, and other possible metrics. KAA is the component which extracts experience from historic data which are used to improve scheduling decisions made by the Scheduler.

It is in general very hard to predict the execution time of a Web Service Call. The Client is often in situation to decide which instance of WS to use for a certain computation. In distributed environments it is rare to have full control over all the computing resources and therefore it is desirable to have an infrastructure which would be able to measure performance, keep records and make predictions about WS classes and their individual deployments (instances).

3 Capturing Information About Web Service Performance

The performance of a WS is measured exploiting the WS-Notification of the WS-Resource Framework (WSRF). It is depicted on the figure bellow how we use WS-Notification to measure duration of certain WS operations based on changes of WS states. Apart from KAA there are three other entities on sequence diagram on the figure 1. "Client" is a component which uses WS interface to initialize and launch required operations of a WS. The "Web Service" implements the service itself and uses "WS-Resource" to represent a model of computation carried out. The WS-Resource has parameters which describe the state of computation. In this explanatory case the state of the computation is: initialized, started or finished. Further we assume that the WS is deployed constantly in a container and is capable of accepting and processing requests (state UP according to [2]). In the first step KAA subscribes for WS-Notification which notifies about activation of states "started" and "finished". Client submits a request to initialize a computation which will be formally represented as a WS-Resource. Initialization must include setting of input parameters required for computation, which will be stored and used as a case for future predictions.

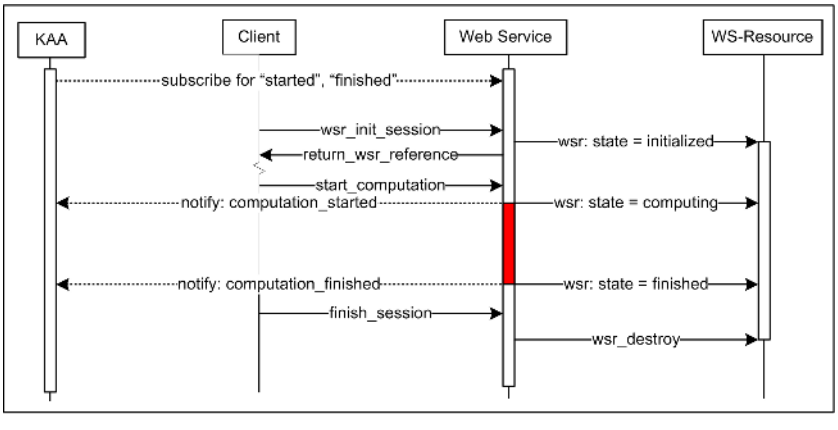


Fig. 1. Sequential graph of WS performance information capture using WS-Notification

4 Prediction of Web Service Runtime

We are proposing an approach of run time prediction for stateful WS based on past cases in context with used resources and invocation parameters. The Capture - Capitalization - Reuse (see Figure 2) cycle used in our work is a traditional knowledge management cycle. The main role of KAA in this cycle is to use historical data about individual WS instances for WS performance prediction. KAA formalizes such findings and store them in a format that could be reused

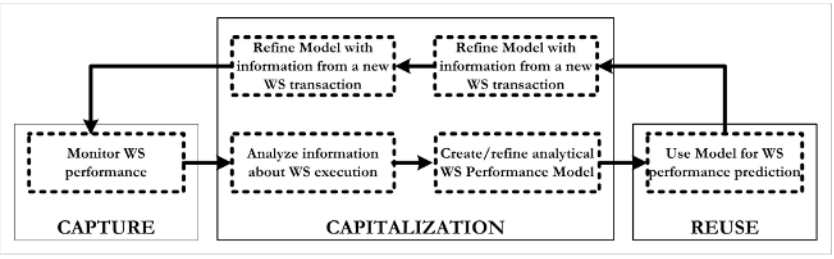


Fig. 2. The Capture - Capitalization- Reuse knowledge management cycle is used for capturing and reusing past cases for future predictions

by other agents (human users or computational services) - preferably in semantically described way. Each discovered dependency between a characteristic and WS Instance (WSI) is used for decision support during a process of selecting a proper WSI for concrete workflow task execution.

Let's suppose we have already captured several cases of WS operations during a certain period of time. Cases are composed of invocation parameters and result runtimes, stored as records in a database. In order to predict the runtime of an operation with known parameters for a certain class of WS we need to perform the following steps:

1. Identify WSI deployments of a chosen WS Class on Grid Resources;
2. Retrieve a history of all relevant executions (cases) based on invocation parameter similarity;
3. Use adjusted or initialized weights for each parameter.
4. Compute similarity between our concerned case and the selected set of all relevant cases;
5. Identify the most similar case/cases.
6. Compute an estimate expected runtime using past runtimes of similar cases.

After WS finishes, we can compare our estimates with the measured runtime values and adjust weights for individual parameters. Representation of whether an WSI is deployed on a Grid Resource (GR) can be represented by a matrix where columns represent GR and rows WSI. Value 1 means that WSIy is deployed and value 0 means that is not deployed on GRx. From such matrix above we can identify for example that WSI2 is deployed on GR1, GR3 and GR4. Therefore for a WSI2 we would retrieve all relevant historical cases (runtime information) only from those GR which have WSI2 deployed. For the purpose of this article let's consider that we examine performance of WSI2 deployment on a single GR while predicting runtime of the future WSI2 invocations based on four invocation parameters.

We need to capture runtime history for our WSI_2 deployment for at least a single operation. We define that runtime duration of an operation is a time elapsed between transformations from WS state Sstarted to WS state Sfinished. For each WS we can measure at least one duration of operation which is the time difference between the WS state "finished" and "started":

$$t_{O_i}^{WSI_2} = t_{S_{finished}}^{WSI_2} - t_{S_{started}}^{WSI_2}. \quad (1)$$

Expression (1) states that runtime of an operation Ox executed by WSI2 can be computed as difference between the states "finished" and "started". Overall WSI runtime is the sum of all operation runtimes:

$$t^{WSI} = \sum_{i=1}^{n} t_{O_i}^{WSI} \quad (2)$$

Cases collected in the capture phase are stored into a database in a form represented in Table 1.

Table 1. Sample case base

Case	p1	p2	p3	p4	R
1	2	2	4	60	175
2	2	2	4	120	328
3	2	2	4	180	472
4	2	2	4	240	638
...					
11	2	2	6	60	132
12	2	2	6	120	234
13	2	2	6	180	338
14	2	2	6	240	443
...					
21	2	2	8	60	126
22	2	2	8	120	219
...					

Let's suppose our concerned WSI2 invocation is going to be executed with the following parameters: p1 = 2, p2=4, p3 = 4 and p4 = 300. Our goal is to predict what will be the runtime of such invocation. Firstly we need to compare and find most similar cases. In our situation the comparison of cases is based on a set of four application relevant parameters P' = p'1, p'2, p'3, p'4 = 2, 4, 4, 300. The parameters and runtimes of past cases are stored in a database and represent the base of cases. Table 1 shows a portion of such case base. The cases in table 2 were generated from an experiment, where each case was performed exactly just once. The similarity of two cases with the initialized case can be measured by a distance in Euclidean space:

$$d(P, P') = |P - P'| = \sqrt{\sum_{i=1}^{n} |p_i - p'_i|} \quad (3)$$

We will have some parameters more considerable, so we use weights to higher or lower importance of a certain parameter:

$$d(P, P') = |P - P'| = \sqrt{\sum_{i=1}^{n} w_i |p_i - p'_i|} \quad (4)$$

Where wi is a weight of i-th parameter, i.e. parameters with a lower priority add bigger distance to the final distance between cases. This type of weights can be called weight of parameter degradation, i.e. essential parameters remain and others are depressed. The use of Euclidean distances is not the only one method how to measure an affinity of two cases. Similarity measuring method usually depends on the parameters' domain. If some parameters cannot be compared due to lack of information, then those parameters are excluded, i.e. are replaced by zero.

Due to problems that accrue from parameters with very different relative values and ranges, we compute a so called z score for each concrete parameter. z score is computed as follows:

$$zscore_{p_i^{C_i}} = \frac{p_i^{C_i} - \bar{p_i}}{\sigma_{p_i}} \tag{5}$$

The equation (5) states that the zscore value of parameter pi for case Cj is computed as difference between the measured (actual) value of pi for case Cj and mean value pi divided by standard deviation of pi. By computing Euclidean distances we get a set of numbers which represent the degree of similarity between the initialized cases compared to all other cases from the available case base. The case with the smallest Euclidean distance is identified as most similar to acase for which we estimate the runtime. We can see the effect of using weights for computation of similarities using the distances in Euclidean space by comparing figures 3 and 4. Firstly we have not used any wages for estimating the similarity. The result is that the two most similar cases are cases 5 and 35 and very similar are 4, 6, 34 and 37. Then we initialized the wages using a correlation coefficient computed for z values of each set of parameters. The result shows that cases 4,

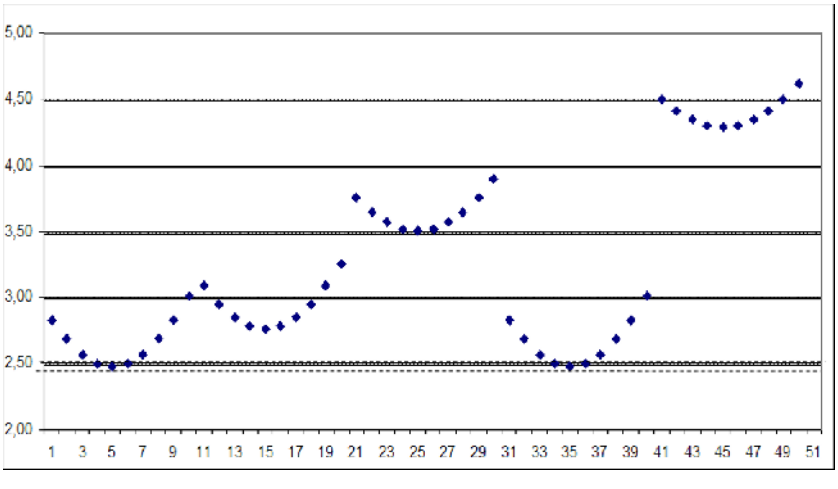


Fig. 3. Graphical representation of the similarity between concerned case and a representative set of past cases.

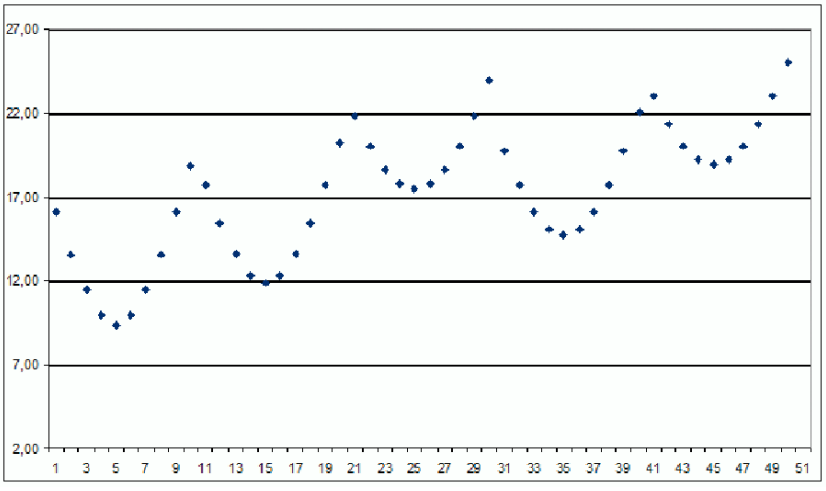


Fig. 4. Graphical representation of similarities represented by Euclidean distances computed using weights

5 and 6 remained to be evaluated as most similar but cases 34, 35 and 36 were evaluated as definitely not similar to our case. From the computed results we have identified, that case 5 is most similar to our case because the Euclidean distance computed for this case is the lowest among the overall case base. For purposes of more accurate prediction we estimate the expected runtime as average value ofrun times of 3 most similar cases. The predicted runtime for our concerned case was therefore computed as follows:

$$t_{C'} = \frac{\sum_{i=4}^{6} t_{C_i}}{3} \tag{6}$$

The result therefore is that a WSI invoked with parameters p1 = 2, p2=4, p3 = 4 and p4=300 will run approximately (638 + 794 + 949)/3 = 793.6 minutes according to formula (6).

We have invoked the service with P? = 2, 4, 4, 300 and the resulted runtime was 792.8 minutes. We can see that our prediction using weighted Euclidean distance provided us with very accurate estimation of the result.

5 Application of the Approach

The K-Wf Grid [3] project deals with knowledge supported Web Service workflow construction. Performance prediction of individual Web Service Instances is crucial for decision making process of both the Workflow Composition Tool (WCT) and Application Builder Agent. A sample scenario is the use of the platform for the flood prediction platform in the K-Wf Grid project. The system is responsible for suggesting best workflow composed of several concrete WS implementations.

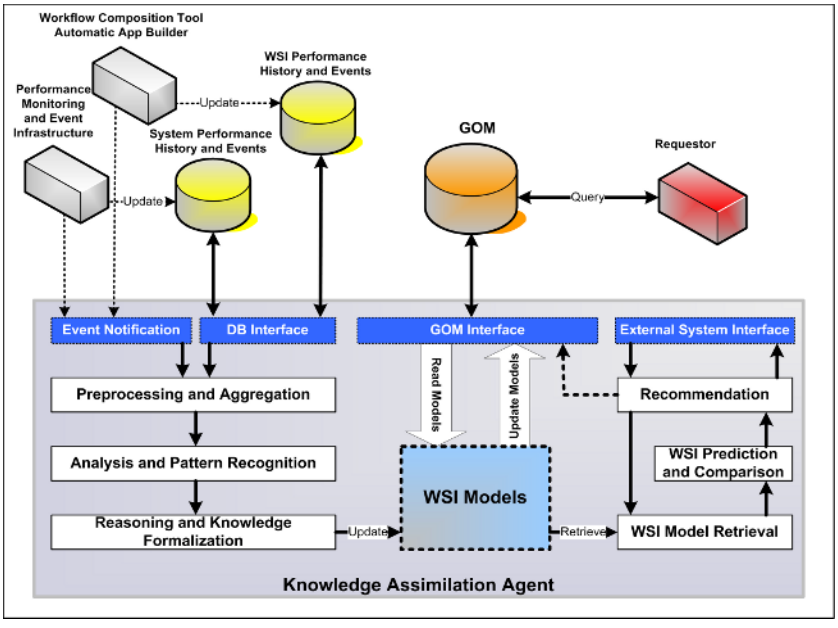


Fig. 5. Architecture of the Knowledge Assimilation Agent (KAA) from the KWf-Grid EU project

Knowledge Assimilation Agent (KAA) in the flood forecasting application can be used to discover for instance dependencies between execution time of a WS and the area for which we make the forecast. The area is stored in a special file (resource) which must be semantically described. In the simplest case the semantic description might contain information about the area and the density of the area grid. When flood prediction will be launched, the KAA will be notified about invocation parameters used by a WS - which can be the URL of a file containing a description of the geographical area. The KAA determines the description of the resource used as input for WS and stores all information into Grid Organizational Memory (GOM). The KAA will be also notified about forecast completion and will store information about several such forecasting computations, thus having a base for successful forecasting. If the KAA discovers any dependencies between an input resource parameter and WS performance it stores the model into GOM. Having relevant models, the KAA is able to predict how a concrete WS will perform, based on the provided input resource description.

6 Conclusion

In this article we have presented our approach for predicting runtime of stateful web services. Our approach is based on computation of similarities using weighted Euclidean distances. The prediction of the runtime for a concrete WS

instance is computed as average value of the most similar past cases. Sample cases used in this paper were collected using experimental invocation of a WS used in an application for flood forecasting [4]. Invocations were carried out on a single dedicated PC cluster infrastructure with predefined steps of invocation parameters. Such conditions of measurements caused relatively smooth and complete results, which could be used for a very accurate prediction of runtime for a selected property set. Future work will include collection of performance data from more complicated environments - loaded grid infrastructures - which will possibly produce very different results for a variety of invocation parameters. Other challenges for the future comprise inclusion of input data (metadata) description properties, prediction of other qualitative performance measures (such as availability or reliability of WS) and adjustment of weights according to comparison between predicted and really measured results.

References

1. Web Services Notification: http://www-106.ibm.com/developerworks/library/specification/ws-notification/
2. Web Service Management: Service Life Cycle: http://www.w3.org/TR/2004/NOTE-wslc-20040211/
3. EU IST *K-Wf Grid* Project IST-2002-511385 http://www.kwfgrid.net/
4. EU IST CrossGrid Project IST-2001-32243 http://www.crossgrid.org/
5. Delpoi Grid(Lab) ? Adaptive Component System: http://www.gridlab.org/WorkPackages/wp-7/
6. Faerman M., Su A., Wolski R., Berman F., Adaptive Performance Prediction for Distributed Data-Intensive Applications, August 9, 1999, Proceedings of the ACM/IEEE SC99 Conference on High Performance Networking and Computing, Portland
7. Balogh Z., Laclavik M., Hluchy L., Nguyen T.G., Gatial E.: "Capture, Discovery and Reuse of Knowledge in REMARK". ICETA'2004, pp. , IEEE Computer Society. September 2004, Kosice, Slovakia.
8. Menasce, D., A., Virgilio, A., F.: Capacity Planning for Web Services - Metrics, Models, and Methods. Prentice Hall, 2002. ISBN 0-13-065903-7

Real-Time Visualization in the Grid Using UNICORE Middleware

Krzysztof Benedyczak[1], Aleksander Nowiński[2], Krzysztof Nowiński[2], and Piotr Bała[1,2]

[1] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Chopina 12/18, 87-100 Toruń, Poland
[2] Interdisciplinary Center for Mathematical and Computational Modelling, Warsaw University, Pawińskiego 5a, 02-106 Warsaw, Poland

Abstract. This paper presents real-time data streaming and client side visualization which was developed using UNICORE middleware. Described here IVis system extends UNICORE capabilities and allows for streaming of the data produced by the running simulations. IVis client provides user with advanced tools for data visualization and analysis. The developed software can be also used for remote steering and control. The important feature of the developed solution is utilization of the UNICORE security infrastructure which allows for data streaming without opening new holes in the firewall. The performance of the IVis system has been measured and is presented. We have found that bandwith is mostly limited by the network speed. Performed tests conformed that developed tools can be efficiently used for data streaming and on-line visualization and streaming on the grid.

1 Introduction

Existing grid middleware has been developed with the main focus on the batch type jobs. Therefore the real-time, on-demand data streaming doesn't suit paradigm of any grid middleware. The reasons of such situation are well known: a large scale computations are performed as batch jobs which produce data postprocessed than on the local workstations. A job is *submitted*, then possibly *queried* for status and output is *fetched* to the local system for postprocessing and visualization.

The main disadvantage of the grid middleware is large communication overhead caused by the infrastructure which provides seamless access to the resources and ensures interoperability between different systems. This overhead results from transfer quite large amount of *metadata* and from need to invoke many *metaoperations*. Both of them are not needed for main computations but are mainly caused by the grid middleware itself.

As it has been observed, as systems get more universal and powerful, the grid middleware is increasing volume of additional communications and operations.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 608–615, 2006.

This is especially true for forthcoming XML based technologies. Grid tools become extended web services and invocation of even simple service job involves serious data preparation and processing.

Streaming of the data generated as a result of the remote computations which run on the grid is not trivial. In particular, there are two main problems to be solved. The first one is access to the job running in the batch system on the target computer which allows to fetch data, and the second one is performance which can be limited by the different components of the grid middleware making real time streaming unrealistic.

The first problem can be solved by the simulation of the interactive access using simple batch jobs. There was number of successful implementations of this approach, for example interactive access tools developed for the UNICORE [1]. Such approach assumes that each interactive command is submitted by a simple batch job, sometimes called service job. In this way user can mimic interactive access but middleware overhead is large. As consequence, the large amount of data cannot be transferred and this solution cannot be used for advanced visualization or real-life steering.

The performance required for the visualization on the grid is usually achieved by creation of the dedicated connection - directly from the system which produces data to the visualization client. This connection is initiated by the grid middleware but is realized using standard network technology bypassing grid security infrastructure. Therefore user cannot use this mechanism with firewalls and local networks.

This paper presents true real-time data access and client side visualization which was developed based on the UNICORE platform [2], however the basic middleaware has not been modified. The UNICORE offers complete and ready to deploy solution. Developed extensions benefit from the UNICORE modular design and flexibility. The detailed description of the UNICORE middleware, its architecture and basic functionality can be found elsewhere [3, 4].

2 Extensibility of the Core Unicore Software

The UNICORE middleware offers well known plugin infrastructure for the UNICORE Client. Basically, the most of the Client functionality is based on the plugins: some of them are distributed together with Client's standard distribution and some are available as third party add-ons developed for the specific applications. The UNICORE plugin can be implemented using two schemes: Task and Extension. The purpose of the *task* is job creation and its submission to the target system. The *task* is incoroprated in the UNICORE workflow and is part of the job created by the user and executed using UNICORE machanisms.

The *extension* gives more freedom for developer. Especially it can use different communication schemas offered by the UNICORE Client and even allows for implementation of whole communication system form the scratch. This functionality has been used previously for example to access databases through their

HTML interfaces. This schema is therefore naturally suitable for the real-time streaming needs.

The plugin concept is not limited to the UNICORE Client. The UNICORE Gateway have similar feature to Client's plugins. The Gateway can be extended with new *connection handlers*. Connection handler is used after initiation of connection between the Client and the Gateway. The UNICORE Gateway checks if Client's certificate was issued by trusted CA — it is the very first step of security scheme. Every instance of the connection handler is associated with one of the Network Jobs Supervisors (NJS) registered in the gateway. When Unicore Protocol Layer (UPL) request comes and is targeted to the NJS[1] it is passed as argument to the handler's *processRequest* method. Normally it transfers request to the NJS, waits for answer, reads it, and forwards to the Client. This schema is simple but there are two important things to note. In order to send NJS's reply to the Client *processRequest* method has access to the gateway$\longleftrightarrow$client streams. We must also remember that during interaction with the NJS, sent reply doesn't mean that the last job has finished execution. This happens only for short service jobs like querying for another regular job status[2].

The UNICORE Gateway extensions aren't well known and, as rarely needed, nearly not used. The one example, besides standard implementation, is alternative file transfer implementation. This technology is also used for SSH tunneling through UNICORE gateway in the "Interactive Access" extensions [1].

We have explored this technology as good candidate to implement data streaming and real-time visualization and steering.

3 IVis Design

The IVis (Interactive Visualization) system [5] consists of three software components: IVisServer, IVisGateway and IVisPlugin (see. Fig. 1). The componants are connected by the IVis protocol used during communication. Elements of the IVis system are either universal or extendable in many ways. Before going into details we would like to give overall description of the actual implementation as a good starting point.

IVisServer is a software component which is deployed together with the standard, already operational, NJS server. It is used to give access, in the real-time fassion, to the user working directory (Uspace) on the target system controlled by the NJS. IVisServer is visible in the UNICORE universe as another Vsite with a special type and protocol[3]. As any other Vsite (so usually NJSes) it is designed to communicate directly with the UNICORE Gateway. The NJS which has corresponding IVisServer available will be called NJS *IVis enabled*.

[1] Conforming to UPL there is one request that is serviced by Gateway itself: ListVsites. Recent Gateways add one more, ListPorts.

[2] Details can be found in [4] in sections regarding *UPL* modes.

[3] Regular UNICORE NJSes have "Unicore AJO and UPL" as its protocol and type "NJS".

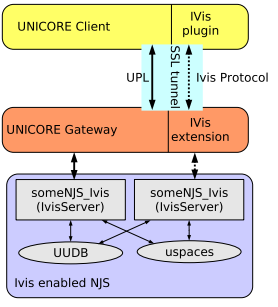


Fig. 1. The overall architecture of IVis system. It consists out of three components: Client's plugin, Gateway's extension and standalone server coupled with NJS.

IVisGateway is a middle point of the system. It's main functionality is very simple: it forwards (without any processing) the whole traffic from the server to the client and another way round. There is also other function of the IVisGateway: it passes client identification data to the server.

The final component of the interactive visualization system is **IVisPlugin** which allows to use the UNICORE Client as end user's platform for real-time streaming applications. It hides existence of special IVis Vsite[4]. With the IVisPlugin user can choose one of the jobs already submitted to IVis enabled NJS. The working directory can be accessed and user can pick up file of interest. Than this file can be processed in the number of ways:

- Mirrored to local file, which grows as remote file grows.
- Monitored with simple text viewer which shows current file contents.
- Parsed and visualized on the fly with the IVisPlugin. In this case the data format must be recognized by the IVisPlugin.

3.1 Gateway Extension

The IVisGateway is a set of classes with two entry points for the UNICORE Gateway. Using them Gateway initializes extension with parameters supplied by the system administrator in the gateway configuration file. In result, an object ready to handle UPL Requests directed towards it is created. The UNICORE Gateway uses initialized object whenever it receives request for extension's Vsite and passes through it request and object streams to and from client who sent request. The IVisGateway blocks connection to its end (i.e. the processRequest method returns when streaming is completed). It can be noted here that purpose of the UPL request sent from the client to the gateway is only to signal that proper extension should be used. IVis extension to

[4] The IVis Vsite *is* visible in the client but user doesn't have to use it in any way. The only valuable information for him is that this Vsite signifies that corresponding regular NJS is IVis enabled.

the gateway performs sanity checks on the received request, e.g. finds if client is aware that it will talk to the IVisServer. If something goes wrong the standard UPL reply is sent and session is unblocked. If no problem is detected, the IVisGateway makes connection to the IVisServer and sends to it the client certificate. IVisGateway immediately, no waiting for server's respons switches to the bridge mode: data is transparently copied between client and server (in both directions).

There is one important technical implication which results form the fact that Gateway passes Java *Object* streams to the IVisGateway. There is no way to come round polymorphic invocation of write/read methods in the Object streams. This complicates the process of a simple data forwarding. Because streaming should be as fast as possible we care not to trigger any (de)serialization what happens if readObject/writeObject methods are used. Therefore usage of read/write methods is acceptable solution[5] but it has one side effect: IVis gateway extension cannot forward serialized objects (you simply can't read byte representation of serialized object with writeObject method using read on the other end). So any client/server cannot use write/readObject methods on IVis streams. With little more work Java serialized objects can be send: object can be serialized to the temporary byte array and then this array is sent over the socket's Object streams.

3.2 IVis Server

The IVisServer fundamental role is to stream files form the Uspace to the UNICORE client. Because of the possible number of use scenarios, we have made server's architecture modular and it has been split into two parts. The base module receives authentication data from the IVisGateway. Using standard UNICORE tools the received certificate is mapped onto UNICORE User (so called Incarnated User) using the same UUDB as accompanying NJS uses. Of course if user's certificate doesn't exist in database the connection is refused. Then, server expects information from the client (recall that IVisGateway just after sending client's certificate connects client's and server's streams) about protocol name it uses. If protocol has matching handler (i.e. class that implements it) it is started. Otherwise connection is refused.

The handler compose second part of the IVis server. This modularity allows for extensibility and fot tunneling of well established protocols. Of course, client must send protocol identification string first. Every handler receives information about (incarnated) user and streams to client. It has also access to the server's configuration data.

Currently the dedicated protocol — IVisProtocol — is implemented but another implementations are being prepared.

[5] This solution still is not ideal. In the Object streams read and write from the basic streams are overridden with versions which packs even primitive Java types like byte arrays into form which can be distinguished from the serialized objects. Fortunately amount of an extra data and therefore processing overhead is in practice unnoticeable.

3.3 IVis Protocol

The IVis protocol was designed as a very simple one to stream both files and constant portions of files. The most significant feature is possibility to fetch file as it is growing. The IVis protocol is binary and is based on a request–answer schema which can be followed by streamed data.

The IVisServer's module waits for file requests. Requests are formed from Unicore job identifier and file name. Handler is responsible for assembling proper real path of the file on the target system. Client does not need to know about any site specific details like Uspace directory.

Before data transmission starts, handler ensures if user has sufficient rights to read file. This operation checks for Unix file read permission.[6]

While session is set up, the file streaming is almost trivial. In the case of a continuous file streaming efficient algorithm is used for file monitoring, and any newly written bytes are almost immediately sent to client.

3.4 IVis Plugin

The last part the developed middleware, the IVisPlugin provides feature-rich user interface. Using IVisPlugin, the end user does not have to possess any information on IVisServer or an details of IVis operation. The user pick ups submitted job, listed in the UNICORE client and to start streaming enters the name of the file. Additionally, the user selects if he wanted to download current file contents or mirror the file as it is growing. These possibilities improves significantly the basic UNICORE Client capabilities which allow for the file transfer after the job was submitted. The transfer of the files created by the running job has been also provided by the FilterPlugin [3], however without any streaming functionality.

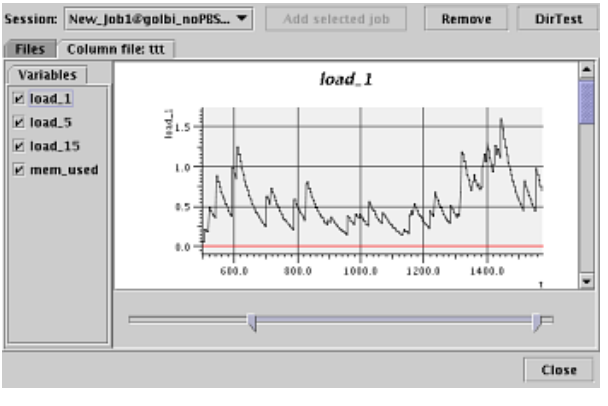


Fig. 2. The output of a simple 2D visualization. With bottom slider user can control display range. Graph is updated automatically as new data is produced on the server.

[6] Since this task is performed by the simple shell script site administrator can change it if needed.

The file picked by the IVisPlugin can be parsed and then presented in the text or graphic form. Currently text preview of the remote file is available. Additionally, the files containing data can be plotted live, as they are transferred to the client workstation (see Fig. 2). Currently column data, for example in the popular gnuplot format, can be plotted with the help of developed by us advanced visualization toolkit. Dedicated filter for visualization of the data generated by the Amber molecular dynamics package [6] has been developed. The additional filters for example for 3D rendering of the molecular data can be easily added to the existing IVisPlugin.

In all cases user can interact with visualization. He can refine intervals of interest which allow to view all data as it is coming or only some parts of it. New visualization capabilities require only implementation of the data parser which is able to interpret data and pass them to the visualization tools.

4 Note on Security

The lack of the security or problems to adjust to the existing firewalls and security policies is the main problem for the data streaming in the grid environment.

The usage of the UNICORE middleware and possibility to establish connections through Gateway resolves low-level security problems like transmission confidentiality. Still there is a need for higher-level, grid related security; namely users authentication and authorization. We put effort to make these tasks as similar as possible to the UNICORE authentication and authorization [7].

The process of authentication is done once per connection by IVisGateway. The first request send by the client must be, signed with users private key. Please note that the the only real constraint here is that the request must be signed: it's existence and kind is determined by the way Gateway's extensions are used. When the job's signature is verified (user's public key is attached to the request) we can be sure that the job was really generated by user owning private key. The only security hole here is that possibly any signed document by this user (which can be publicly available) can be used as "ticket" to the system. To avoid this we put some magic string in signed AJO which is checked in Gateway. As jobs containing this cookie are never stored anywhere and send only within encrypted channel third party can't take it over. To increase safety of the system, a timestamp is put into signed AJO and IVisGateway checks if it is in reasonable period upon job retrieval.

5 Performance Results and Conclusions

Since performance is one of the most important parameters of the data streaming, we have performed number of tests in the different experimental setups.

The throughput measurements are summarized in a table 1. It can be seen that data transfer speed is limited by the network bandwidth and possibly by the disk operations. The final transfer exceeds 60% of the theoretical bandwidth

Table 1. Results of the throughput tests for various configurations

Speed	*Transfer mode*	*Medium type*
7.4 MB/s	RAM to RAM	100 Mbit, LAN no routers between
6.4 MB/s	HDD to HDD	100 MBit, LAN no routers between
625 kB/s	RAM to RAM	11Mbit Wi-Fi + 3 routers (100Mbit LAN)
614 kB/s	HDD to HDD	11Mbit Wi-Fi + 3 routers (100Mbit LAN)

and is in our opinion good result. One should remember that the transmission is encrypted and data goes through intermediatory point – UNICORE Gateway.

The another parameter describing quality of the data streaming is communication latency. In the presented case the latency is not important as far as delay between start and time first results arrive is small. For the developed software the delay is unnoticeable assuming that recent hardware and resonable network connection is used.

In conclusion we can state that the results of the data streaming and interactive visualization using UNICORE infrastructure are very promising. The presented software proofs that real-time visualization using UNICORE middleware is possible and usable.

Acknowledgements. This work is supported by European Commission under IST grant UniGrids (No. 004279). The financial support of the State Comeetee for Scientific Research is also acknowledged.

References

1. Dietmar W. Erwin, David F. Snelling, UNICORE: A Grid Computing Environment In: *Lecture Notes in Computer Science* vol. 215, Ed. R. Sakellariou, J. Keane, J. Gurd, L. Freeman Springer-Verlag 2004, p 825
2. UNICORE. Unicore Forum. http://www.unicore.org.
3. Wypychowski J., Pytliński J., Skorwider Ł., Benedyczak K., Wroński M., Bała Life Sciences Grid in EUROGRID and GRIP projects. *New Generation Computing 22, 2*, pp. 147–156, 2004
4. D. Erwin, ed. UNICORE plus final report – uniform interface to computing resources Forschungszentrum Jülich, 2003, ISBN 3-00-011592-7.
5. IVis. SourceForge. http://unicore.sourceforge.net
6. Kollman, P., *AMBER (Assisted Model Building with Energy Refinement)*. University of California, San Francisco, USA, 2001.
7. T. Goss-Walter, R. Letz, Th. Kentemich, H.-Ch. Hoppe, and Ph. Wieder An Analysis of the UNICORE Security Mode Global Grid Forum, Grid Forum Document – Informational, GFD-I.18, Sept., 2003.

Development of a Grid Service for Scalable Decision Tree Construction from Grid Databases

Peter Brezany[1], Christian Kloner[1], and A. Min Tjoa[2]

[1] Institute of Scientific Computing,
University of Vienna, Nordbergstrasse 15/C315, A-1090 Vienna, AUT
brezany@par.univie.ac.at, kloner@par.univie.ac.at
http://www.par.univie.ac.at/~brezany/
[2] Institute for Software Technology and Multimedia Systems,
Vienna University of Technology, Favoritenstrasse 9-11/188/2, A-1040 Vienna, AUT
tjoa@ifs.tuwien.ac.at
http://www.ifs.tuwien.ac.at/~tjoa/

Abstract. Classification deals with discovery of a predictive learning function that classifies a data object into one of several predefined classes. We present a novel decision-tree-based classification service which can be used either autonomously or as a building block to construct distributed and scalable classifiers that operate on data repositories integrated into the Grid that typically involve large, complex, heterogeneous, and geographically distributed datasets. Although classification is considered as a well-studied problem – a lot of classification methods were proposed for sequential, parallel and distributed environments, so far, to our best knowledge, no effort was devoted to building classifiers based on federation of Grid resources. The Grid service described in this paper was fully implemented and integrated into the GridMiner framework (www.gridminer.org). Scalability and performance of the prototype implementation have been examined and the results are presented.

1 Introduction

Grid Data Mining denotes efforts to utilize data mining and knowledge discovery techniques leveraging the large-scale computational and storage power offered by Grid infrastructures. Data preprocessing and data mining algorithms are known to be both compute and data intensive and therefore appear to be ideal pilot applications to test whether Grid toolkits hold what they promise.

One of the most important data mining tasks is classification - it deals with discovery of a predictive learning function that classifies a data object into one of several predefined classes. The intuition is that by classifying larger datasets, the accuracy of this function can be improved - this hypothesis has been confirmed in several studies.

In this paper, we present a novel decision-tree-based classification service which can be used either autonomously or as a building block to construct distributed and scalable classifiers that operate on data repositories integrated into the Grid.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 616–624, 2006.

Decision Trees are representations of classifiers that can be depicted as graphs [5]. The root node and the inner nodes represent tests and the outgoing edges represent the outcomes of the test. The leaf nodes are class labels indicating the group the example belongs to.

Considerable efforts in parallelizing classification algorithms for tightly coupled systems have been undertaken. SLIQ [4] introduced the idea of separate attribute lists written to files but still requires that a data structure, that grows in direct proportion to the number of examples in the dataset, stays memory-resident all the time. SPRINT [7], a follow-up work, extended the attribute list technique to remove this memory restriction and proposes a parallelization.

So far, to our best knowledge, no effort was devoted to building classifiers based on federation of Grid resources. Our previous work [3] outlines an idea of a distributed classification algorithm, denoted as DIGIDT[1] that leverages concepts introduced by SPRINT but uses a different approach to dataset partitioning and workload and task assignment. DIGIDT not only partitions the workload optionally row-wise (horizontal), as described in [7] but assigns attributes to worker nodes, i.e. it performs a column-wise (vertical) partitioning. Column-wise partitioning allows very interesting applications. Projects such as the traumatic brain injury project [2] impose very strong restrictions on data security and privacy. For example the data collected at different institutions must not be combined into a universal dataset, but a global decision tree classifier should be constructed. DIGIDT supports such scenarios, where attributes or attribute groups are located at different worker nodes, each worker node works on its local data and a master node builds the global classifier. This paper further extends and implements the DIGIDT ideas in a real Grid environment. The details of the SPRINT and DIGIDT algorithms are not discussed.

The rest of the paper is organized as follows. Section 2 introduces the internal architecture of the service and the global communication model of the distributed solution. The data flow concepts applied in the distributed approach are discussed in Section 3. The implementation concepts are briefly presented in Section 4. Section 5 presents the first experimental results. Finally, we briefly conclude in Section 6.

2 Global System Architecture Design

Building distributed systems is an inherently difficult and complex task. To reduce complexity, the first goal was to design a sequential version of a Decision Tree Service [1] (*DT Service*) based on the SPRINT algorithm. As a target platform, the Globus Toolkit 3 container was chosen. This sequential version was completely written in Java and then extended to implement the SPRINT and DIGIDT concepts in a distributed context.

The main objective for the development of the distributed version was to find a Grid-enabled solution. The ability to use the Globus Toolkit 3 for transferring messages via SOAP and files via Reliable File Transfer Protocol matched our

[1] DIGIDT stands for Distributed Grid-enabled Induction of Decision Trees.

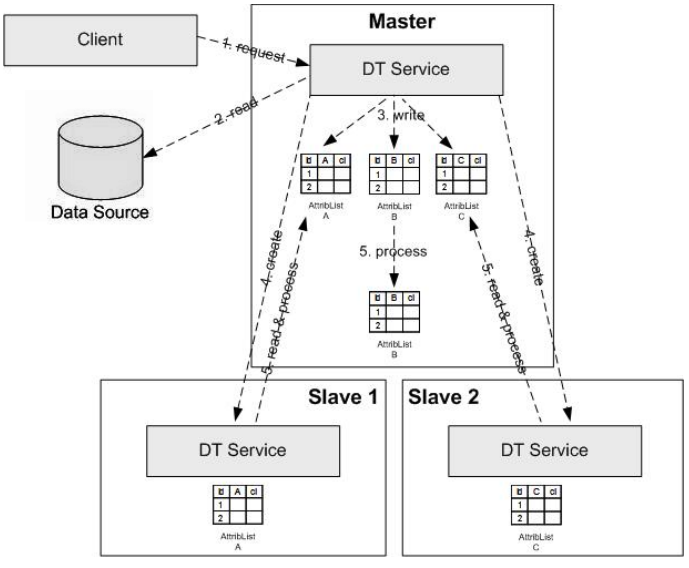


Fig. 1. The global communication model. The first requested Node is the master Grid node, the others are the slave Grid nodes controlled by the master. Each node encapsulates the DT Service communicating over SOAP.

needs perfectly. So the distribution mechanism was directly integrated in the Grid Service extending the *DT Service* interface, named as *DTServicePortType*. The implemented concept is illustrated in Fig. 2. The *DTServicePortType* interface provides all the functionality needed by other remote DT Service instances to transfer informations and resources like *attribute list* files. The distribution specific methods of the *DTServicePortType* interface are invoked by the *AttributeListAgent* class located on another Grid node. Fig. 1 outlines the global communication concept between the individual Globus Toolkit containers, referenced as Grid nodes. The Grid node, getting the request to start the DT Service, is chosen as *master* managing all the connections to other Grid nodes. These nodes are called *slaves* and are used to provide functionality for storing, sorting and splitting *attribute list* files as well as for evaluating class histograms[2], also called *count matrices*.

3 Distributed Data Flow

Using a distributed mechanism for building a decision tree, three approaches for reading an input dataset can be considered: (1) The input dataset is read on the master, transforming the records into attribute lists and writing them into files. If the given attribute lists must be split or sorted, each file is read by another remote DT Service instance and the resulting split or sorted attribute lists are

[2] A data structure counting for each attribute value the occurrence of its classes.

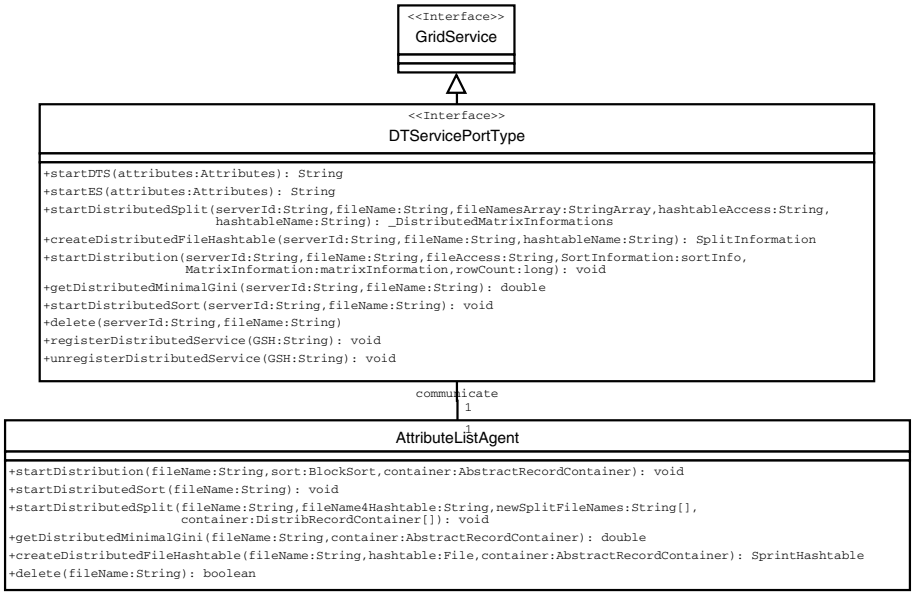


Fig. 2. The interfaces and class, which manage distribution on the Grid. *DTPortType* extends the OGSA Grid Service and *AttributeListAgent* is invoking distributed operations on an implementation of the *DTPortType* interface via SOAP. For each Grid node (see Fig. 1) there is one *AttributeListAgent* instance referencing to the particular implemented *DTPortType* instance on another node.

locally stored on the particular remote nodes; (2) This approach differs slightly from the previous one in writing the attribute lists not into files but writing them directly to other Grid nodes using streams; and (3) The involved Grid nodes - the master and the slaves - read the same input dataset whereas each node writes another attribute list locally into a file. In the first and second approaches, the input dataset is only read from one location and each column value is used to write the attribute lists on the master. Hence, the attribute lists of the master must then be distributed to its slaves to parallelize the most time-consuming parts of the decision tree construction like selecting the right splitting point for each attribute list and splitting the lists. Thus, the advantage in the third approach is, that the attribute lists don't have to be transferred from the master because each slave generates its needed attribute lists locally from the same read input dataset. However, the big disadvantage over the previous approaches is, that the whole input dataset must be read by each slave transferring a lot of unnecessary column data for writing at least one attribute list per slave in a local file. Comparing the three approaches under assumption of the network connection and bandwidth of the master node and the node, from where the input dataset must be read, is equal, the first and the second approach are the best ones. The DT Service implementation was designed to use the first approach.

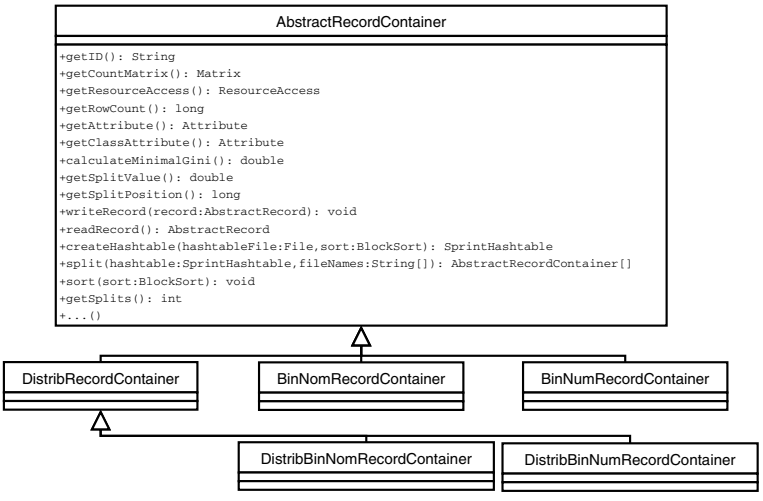


Fig. 3. Overview over the local and distributed containers using SPRINT functionality. The distributed containers have an additional reference to the *AttributeListAgent* class shown in Fig. 2.

Each generated attribute list file is wrapped by a special *Container* class which is either distributed, meaning that the attribute list file is on another Grid node remotely working on it with the help of the *DistribRecordContainer* extending classes, or local, in which way the *BinNumRecordContainer* and *BinNomRecordContainer* classes are used. Its specification can be found in Fig. 3. The prefix *Bin* of the *Container* classes is highlighting, that they are implemented for building binary split decision trees. Because numerical attribute lists must be sorted before they can be used for evaluating the right split value as well as the calculation of the *gini* index[3] of discrete and continuous attribute lists is slightly different, *Container* classes for both were implemented using the inheritance advantage of object oriented programming. These classes provide all the functionality for the SPRINT algorithm like sorting, calculating the mentioned *gini* index and splitting. Hence, the master uses a local container for each attribute list file at the beginning. With the help of *Grid Service Handlers* (GSHs), which are URLs pointing at other DT Services, the distribution of the attribute list files to other Grid nodes is done. For each GSH an *AttributeListAgent* class is generated, which communicates with a remote DT Service. These GSHs can either be specified by the user as parameters or read from a configuration file. Besides, other Grid nodes can register their DT Service as default to others. Depending on the given GSHs, parts of the local containers are converted to distributed containers. Distributed containers are wrappers around local containers on other slaves. Therefore, each distributed container uses an *AttriubteListAgent* class to communicate with other DT Services getting access to their local containers.

[3] A method for finding node split points.

In order of the conversion from a local container to a distributed container, the class histogram of the attribute list, the URL for accessing the attribute list file on the master and the file name, used as identifier of each container, is sent to the appropriate slave. Because the names of the slaves' attribute list files are generated per timestamp by the master and each slave can be used by more than one master, the file names can not be used to identify attribute list files uniquely. Two masters must be able to work remotely on files with the same file name but containing different attribute lists. Thus, the server identifier of the master is also sent to the slave, which is in fact the IP Address. This ID is used to identify attribute list files in combination with their file names uniquely on each slave by creating for each master a directory named after the server ID. In this directory all the attribute list files are stored controlled by the master's distributed containers.

One problem during the development of a distributed service is, how to make sure, that the distributed containers on the master side are definitely working on the referenced local containers on the slave side. The solution for this was in developing a special naming service. If the master sends a request to a slave via *AttributeListAgent* class to distribute a file, a local container is created on the slave using a *ResourceAccess* class to operate on an URL stream instead of a file. If the resource is only needed once on a slave, its records are read directly by an URL input stream. This results in noticeable performance gain because the file doesn't have to be transferred to the local file system to be read afterwards and then deleted. In order to identify the local container uniquely on the slave, it is bounded with the server ID as root context and the file name as context of the root context by the naming service. If the master wants to get the gini index of a distributed attribute list, it sends the appropriate request with the help of the distributed container, using the *AttributeListAgent* class for communication on the Grid, to the DT Service, specifying server ID and file name as parameters. These values are used by the naming service to get the slave's local attribute list container where the appropriate method is invoked returning the gini index to the master node. The master invokes the same method for getting the gini index of each attribute list on either a distributed or a local container. If a distributed container has the minimum gini index, a hash table file, which simply maps the attribute lists's record identifiers to attribute list identifiers, is created by splitting the winning attribute list on the slave and the location and the file name is sent to the master. In order to evaluate, whether an attribute list has only one class value or more, the splitting point and the splitting value as well as the count matrices of the attribute lists, obtained by splitting the winning attribute list, are also transferred. If one attribute list has reached a certain precision of predicting the class value, the splitting for this decision tree branch is stopped and the attribute lists are transformed to a clean leaf node. If there are more splits to execute, the attribute lists are transformed to an inner node containing a test on the split value. Additionally, the master sends the hash table file name, the URL—information how to access the hash table file—its server ID, the file name of the attribute list, which must be split and the file names

for the newly split attribute lists to the appropriate slaves. The slaves transfer the hash table file, if it is not already on their disk, with the help of the given URL parameter and the hash table file name and split the given attribute list file using the local container, looked up in the naming service with the server ID and the file name as identifiers. The new created attribute list files are then wrapped by local containers on the slaves and by distributed containers on the master. In order to user the local containers by the distributed containers, they must be bound by the naming service. Then the gini indices for the partitioned attribute lists must be calculated again to get the next splitting point.

4 Implementation Notes

The implemented DT Service converts its input dataset, which is given as WebRowSet data, to a binary structured decision tree. This tree is either written as in PMML form into a Xindice database or into a file. The DT Service also integrates tree pruning, based on the pessimistic error pruning [6], and an Evaluation Service, which uses as input PMML data and a testing dataset given as WebRowSet data. The output is PMML with additional precision values and is again written either into a Xindice database or into a file. All input data can be transformed by specifying XSLT data loaded from an URL. Input data is also loaded by URLs, so the services are not locally limited. The set of adjustable features of the sequential as well as of the distributed version includes: target class attribute name (the attribute whose values should be predicted), split precision, split mode (binary), amount of records sorted internal, internal sort algorithms, external sort algorithms, tree writer and evaluation writer, and transfer protocols.

5 Performance

A set of tests has been created to estimate the basic performance of the distributed DT Service. All tests have been executed at least thrice and the average values have been used. As dataset a *WebRowSet* file[4] (weather data) was used containing 2 numerical and 3 nominal attributes. The tests were executed on 4 computers running Globus Container 3.2 connected by a 100 MBit network:

Master: Sun Solaris 9 on a Sun Fire 880 with 4 Sparc CPUs at 750MHz and 8 GB main memory

Slave1: Sun Solaris 9 on a Sun Blade 1500 with 1 Sparc CPU at 1062MHz and 1.5 GB main memory

Slave2: Sun Solaris 9 on a Sun Blade 1500 with 1 Sparc CPU at 1062MHz and 1.5 GB main memory

Slave3: Sun Solaris 9 on a Sun Blade 1500 with 1 Sparc CPU at 1062MHz and 1.5 GB main memory

[4] A XML file containing metadata and records of a dataset.

Client: Red Hat Linux 9 on a Toshiba Satellite with 1 Pentium 4 CPU at 2200MHz and 512 MB main memory

The performance of the service was also measured using only 1 and 2 nodes. In the first case, only the Master was used to build the decision tree, in the second case Master and Slave1 were used. Fig. 4 shows the results of the tests using the configuration for sorting a maximum amount of 600000 dataset records internal. For internal sorting, the QuickSort algorithm is used.

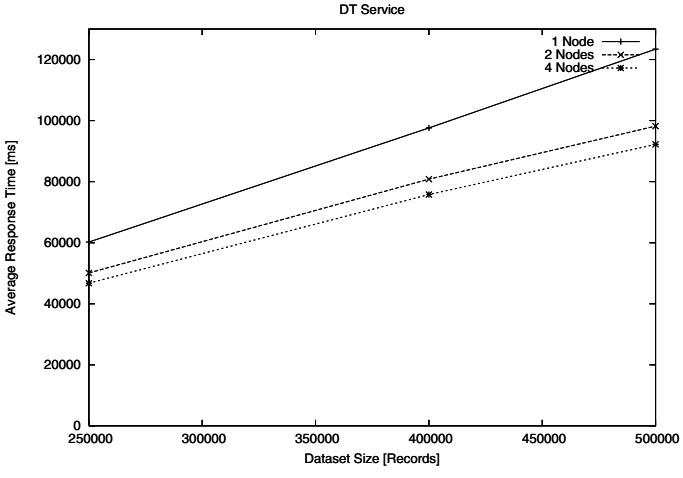


Fig. 4. Average results of response time for 3 runs. The DT Service is configured to sort a maximum amount of 600000 dataset records in main memory before using external sorting method.

6 Conclusion

In this paper we described the Grid service for constructing scalable and distributed classifiers and the components it consists of. We implemented a prototype of this service based on the Globus 3 toolkit. A set of performance tests was carried out to analyze behavior of the service. The prototype is an important intermediate step towards the vision of high-performance data mining in Grid databases.

References

1. P. Brezany and C. Kloner. Programming the decision tree service within the grid data mining framework gridminer-core. Technical Report GridMiner TR 2004-05, University of Vienna Austria, September 2004.
2. P. Brezany, A Min Tjoa, M. Rusnak, and I. Janciak. Knowledge grid support for treatment of traumatic brain injury victims. International Conference on Computational Science and Its Applications, Montreal, Canada, May 2003.

3. Juergen Hofer and Peter Brezany. Distributed decision tree induction within the grid data mining framework gridminer-core. Technical Report GridMiner TR 2004-04, University of Vienna, Vienna, Austria, March 2004.
4. Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.
5. J.R. Quinlan. Induction on decision trees. *Machine Learning*, 1:81–106, 1986.
6. J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.
7. John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, pages 544–555. Morgan Kaufmann, 3–6 1996.

Block Matrix Multiplication in a Distributed Computing Environment: Experiments with NetSolve*

Luisa D'Amore, Giuliano Laccetti, and Marco Lapegna

Department of Mathematics and Applications,
University of Naples Federico II,
via Cintia Monte S. Angelo, 80126 Naples, Italy
{damore, laccetti, lapegna}@dma.unina.it

Abstract. We address synchronization issues of some block matrix multiplication algorithms in a distributed computing environment. We discuss performance behavior of a client/server implementation of these algorithms focusing on the most appropriate version which delivers the minimum synchronization overhead. Numerical experiments are carried out using the NetSolve distributed computing system.

1 Introduction

Distributed computing aggregates computational resources in order to tackle problems that cannot be solved on a single system. Depending on the software and hardware infrastructure, these resources might comprise the majority of the supercomputers in the country or simply all the workstations within a department. Over the last decade, distributed computing received great attention from the scientific computing communities, thanks to a substantial improvement of the networks bandwidth, and it is now feasible the use of geographically scattered computers as a single computational resource. A significant example is the Italian national research network GARR [7], where the backbone bandwidth has grown from 2Mbit/sec in 1994 up to 2.5Gbit/sec in 2002, a growth factor of about 1000 in eight years, much more than the Moore's law about the processors speed. In general, for computationally demanding problems the exploitation of parallelism is a key issue in order to compute the solution within a reasonable time. This means to decompose the given problem into smaller subproblems to be solved concurrently (data or task decomposition).

However, it is important to underline that distributed computing environments are composed by heterogeneous computational resources, both from the static (processors, operating systems, arithmetic,...) and dynamic (workload of the systems, effective bandwidth of the networks,..) point of view, making very

* This work has been partially supported by Italian Ministry of Education, University and Research (MIUR) within the activities of the WP9 workpackage *Grid Enabled Scientific Libraries*, coordinated by A. Murli, part of the MIUR FIRB RBNE01KNFP Grid.it project.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 625–632, 2006.

difficult an efficient synchronization among the nodes. For this reason, one of the main approaches towards the development of distributed applications is based on the client/server programming model where the application is divided into a large number of essentially independent tasks that are dispatched to several servers, and a "coordinator" task managed by the client module. In the parallel computing community these problems are called "pleasingly" or "embarrassingly" parallel.

The most significant example in this sense is the SETI@home project [10]. This project uses idle computers for the analysis of radio signals outcoming from space to find extraterrestrial intelligence, and it has been able to aggregate more than 5 millions of computers, achieving a average performance of about 60 Tflops, about the same of today's most powerful supercomputers [4].

Beyond such example of mere networked computing, new issues arise in the development of algorithms and software able to run efficiently on such a pervasive hardware and software infrastructure. Following [5], in this paper we deal with on-demand computing where remote resources are used to meet short term requirements that cannot be cost effectively or conveniently locally located. These resources may be hardware, software libraries, data repositories, and so on. In contrast to traditional supercomputing, these applications are often driven by cost-performance concerns rather than absolute performance. The challenging of on demand computing derives primarily from the dynamic nature of resources requirements and the potentially large populations of users and resources. Main issues include middleware related topics like resources brokering and management, configuration, authentication and security as well algorithm related aspects like fault tolerance, latency tolerance, heterogeneity management and performance.

In this paper our reference computational environment agrees with a client/server programming model where:

- the servers do not communicate directly each other but with the client only,
- the selection of the resources is in charge to the underlying computing environment by means of their own dynamic allocation strategies.
- the computational information about the servers, including their availability, load, processor speed, are hidden to the client.

In these years several computing environments have been developed with the aim to address these topics, allowing, at the same time, a friendly access to remote resources. Among them there are NetSolve [1] and Condor [8]. NetSolve is the distributed computing environment where we implemented our algorithms. It has been developed at the University of Tennessee to be a simple-to-use middleware system that allows users to access computational resources and to use remote libraries, without the need to locate, configure and install them. However, for the development of algorithms for distributed computing environments, a completely different approach with respect to the classical parallel computing methodologies, is necessary. As case study we consider a block matrix multiplication algorithm because it is a basic linear algebra computational kernel representative of similar other computations. On the other hand, it encompasses a

lot of data movements, so that to minimize the synchronization overhead among the nodes becomes a challenging task.

The paper is organized as follows: in section 2 we analyze the performance behavior of some versions of block matrix multiplication algorithm in a client/server programming model, in section 3 we describe experiments on two different NetSolve systems: a "wide area" system located at the University of Tennessee and a "local area" system installed in our university campus. Finally, conclusions are discussed in section 4.

2 A Client Server Block Matrix Multiplication Algorithm

Starting from PUMMA (Parallel Universal Matrix Multiplication Algorithm) [3], the algorithm on which PDGEMM routine of ScaLAPACK is based, similar algorithms for block matrix multiplication have been developed in the last decade. These algorithms are tuned in order to optimize the synchronization overheads on tightly coupled distributed memory machines by overlapping computations and communications; recently, new issues on heterogeneous networks of workstations have been addressed; in this case the load balancing of processors running at different speed is the challenging task (e.g. [2] and [9]). These implementations are based on the revision of classical parallel algorithms for homogeneous environments and they suppose that key features of the computing resources, as the cycle times of the CPUs, are known, so it is possible to dispatch to each node an amount of work proportional to its computational speed. However, these algorithms are based on the SPMD programming model, suitable for distributed memory multiprocessors with a tightly intra node synchronization but critically liable for the performance decline in a client/server distributed computing environment.

As an example of revision for the classical parallel block matrix multiplication algorithms let us assume that A is an $(m \times n)$ matrix, B is an $(n \times p)$ matrix and C is an $(m \times p)$ matrix divided for simplicity of notations in square blocks of order r, with n, m and p divisible by r. Then the blocks number of the matrix are $MB = m/r$, $NB = n/r$ and $PB = p/r$.

In Figure 1, three variants of a standard blocks algorithm for the matrix multiplication $C = A \times B$, obtained by the permutation of the loops indices are

```
for I=1, MB (in parallel)
  for J=1, PB (in parallel)
   for K=1, NB
    C(I,J)=C(I,J)+
    A(I,K)B(K,J)
   endfor
  endfor
endfor
```
a) (I, J, K) ordering

```
for I=1,MB (in parallel)
  for K=1, NB
   for J=1, PB (in parallel)
    C(I,J)=C(I,J)+
    A(I,K)B(K,J)
   endfor
  endfor
endfor
```
b) (I, K, J) ordering

```
for K=1, NB
  for I=1, MB (in parallel)
   for J=1, PB (in parallel)
    C(I,J)=C(I,J)+
    A(I,K)B(K,J)
   endfor
  endfor
endfor
```
c) (K, I, J) ordering

Fig. 1. Standard versions of parallel blocks matrix multiplication

then shown. Note that the other versions are equivalent to these ones and all versions are based on the same matrix operation:

$$C(I,J) = C(I,J) + A(I,K)B(K,J) \tag{1}$$

In a client/server implementation, for given values of I, J and K, this operation can be computed by sending from the client to a server the three blocks $A(I,K)$, $B(K,J)$ and $C(I,J)$, then the server can update the block $C(I,J)$ and it can sends back the result to the client. It is important to note that in all cases the only possible source of parallelism is along the indices I and J, in the sense that each block $C(I,J)$ can be computed independently from the other ones. This is not possible for the index K, because of the risk of "race condition" accessing a given block $C(I,J)$ for different values of K. Then, in order to reduce the synchronization overhead the main problem in a client/server implementation is to define which of the orderings in Figure 1 has to be used to compute the several matrix operations involving the blocks $C(I,J)$, $A(I,K)$ and $B(K,J)$. For the (I,J,K) ordering (Figure 1.a) the client generates $MB \times PB$ independent threads of computation, each of them managing the sequence along the index K. For the (I,K,J) ordering (Figure 1.b) the client generates only MB independent threads of computation, each of them generating PB parallel tasks at every step of the index K . Finally, for the (K,I,J) ordering (Figure 1.c) at each step of the index K, the client generates $MB \times PB$ parallel tasks, and it has to wait for the completion of all these tasks before generating new ones.

For a computational cost analysis, let t_{ijk} denote the execution time (computation and communication) needed to perform the operation (1) and $T^{(a)}$, $T^{(b)}$ and $T^{(c)}$ the total execution times for the three orderings in Figure 1. It is easy to found that:

$$T^{(a)} = \max_{i,j} \sum_{k} t_{ijk} \quad T^{(b)} = \max_{i} \sum_{k} \max_{j} t_{ijk} \quad T^{(c)} = \sum_{k} \max_{i,j} t_{ijk}$$

so that:

$$T^{(a)} \leq T^{(b)} \leq T^{(c)}$$

Then the (I,J,K) ordering is more oriented to a distributed client/server implementation than the others two orderings. The worst case is the (K, I, J) ordering. Let us assume, for a while, the ideal case, where the environment is homogeneous and dedicated to the computation and $m = n = p$. In this case the execution time $t_{ijk} = t$, is the same for all the values of I, J, K, and

$$T^{(a)} = T^{(b)} = T^{(c)} = NB \cdot t$$

This result shows a linear growth with respect to NB of the total execution time when the matrix dimension n grows while the block dimension $r = n/NB$ is kept constant, and that the ideal scaled efficiency when we multiply by α the matrix dimension n is:

$$S_\alpha = T_n^{(a)} / T_{\alpha n}^{(a)} = \frac{n \cdot t/r}{\alpha n \cdot t/r} = \frac{1}{\alpha}$$

Finally let us compute the communication overhead of each task. In all the cases, for each value of K, the core instruction (1) is executed $MB \cdot NB \cdot PB$ times, each requiring 4 blocks communications of size r^2 ($C(I,J)$, $A(I,K)$, $B(K,J)$ and back the update of $C(I,J)$) between the client and the servers. This means that $4r^2$ double precision floating point data, are exchanged between the client and the servers. Then, the total communication overhead is due to the movement of

$$CO = \frac{32n^2 MB \cdot PB}{NB} \quad \text{byte.}$$

We implemented the three versions of the block matrix multiplication in the NetSolve distributed computing environment. This environment uses a client-agent-server paradigm aimed to dispatch on the most suitable server those calculations that require specialized software and, at the same, to minimize the total communication overhead among the nodes of the environment. More precisely, the agent keeps track of static and dynamic information about all the servers, selects one of these on the basis of their static (hardware and software) and dynamic (load balancing, effective networks bandwidth) features and notifies the choice to the client. The client is then able to send directly to the server the data of the required task, and the server can use its installed software libraries to perform the computation on client data. Finally the server sends the results back directly to the client and the agent is notified of the completion of the task. For instance the core operation (1) is executed on the servers using the DGEMM routine of LAPACK library, available with the release 2.0 of NetSolve.

3 Computational Experiments

In this section we discuss experiments that we carried out, aimed to evaluate performance behavior of the block matrix multiplication algorithms previously described. For each experiment, the reported total execution times are the average of 10 runs launched at different day hours in order to use the networks with different workloads.

To this aim we used two NetSolve infrastructures. The first one is a system located at our department that we call *LAN system*, and a second one is the system located at the University of Tennessee that we call *WAN system*. For

	LAN system	WAN system
Client location	Dept. Math. and Appl. Univ. Naples - Italy	Dept. Math. and Appl. Univ. Naples - Italy
Agent and servers location	Dept. Math. and Appl. Univ. Naples - Italy	Comp. Science dept. Tennessee Univ.
Servers	7 PCs Pentium III and IV	> 50 PCs and WS
Measured latency and bandwith	$\sim$ 200 μsec $\sim$ 50 Mbits	$\sim$ 140 msec $\sim$ 1.5 Mbits

Fig. 2. features of the NetSolve systems used for our experiments

both systems we report in Figure 2 the main features with special regard to the network. Note the client location is the same in both cases.

A first set of experiments is aimed to verify the effectiveness of the (I,J,K) ordering of the block matrix multiplication algorithm respect to the other versions. With this aim we implemented the two orderings (I,J,K) and (K,I,J) in Figure 1 (namely the best expected version and the worst expected version) on the WAN NetSolve system with square matrix of order $n = 250$, 500, 1000, 1500, 2000, and a fixed block size $r = 250$, thus $NB = 1$, 2, 4, 6, 8. In Figure 3 we report the average execution time of the two orderings, where it is evident the better performance of the (I,J,K) ordering of the algorithm. Actually the (I,J,K) versione exploits for every test a smaller average execution time, with a gain of about 50% respect to the (K,I,J) version, because of the smaller syncronization overhead.

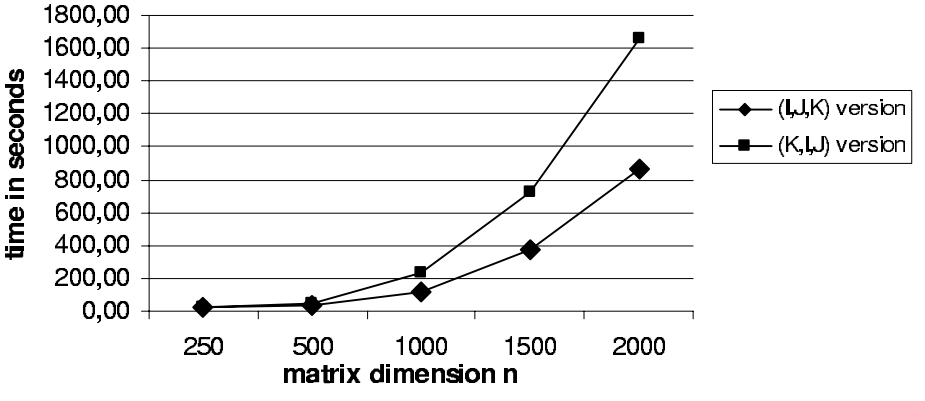


Fig. 3. Average execution time of two orderings of the block matrix multiplication

A second set of experiments is aimed to compare the execution times of the (I,J,K) ordering of the block matrix multiplication algorithm, on the two different NetSolve systems. For these experiments the results are reported in Figure 4. For the same values of n and r used in the previous experiments, we estimate that on the LAN system the total execution time is about 50% smaller with respect to the WAN system, because of the smaller latency and the higher bandwidth of the networks. In order to quantify the performance gain, let us observe the scaled speed up S_2 achieved from $n = 1000$ up to $n = 2000$: on the LAN system $S_2 = 0.23$ whereas on the WAN system $S_2 = 0.14$. These values should be compared with the ideal scaled efficiency $S_2 = 0.5$. Furthermore, we previously stated that, in the ideal case, the execution time grows linearly, but in our experiments we found that the total execution time of the (I,J,K) version grows roughly as NB^2 on the LAN system because of the system overhead. However this asymptotic rate is still smaller than the asymptotic rate of a sequential execution, where the total execution time grows as NB^3. On the other hand, when the (I,J,K) version is executed on the WAN system we found the same grow

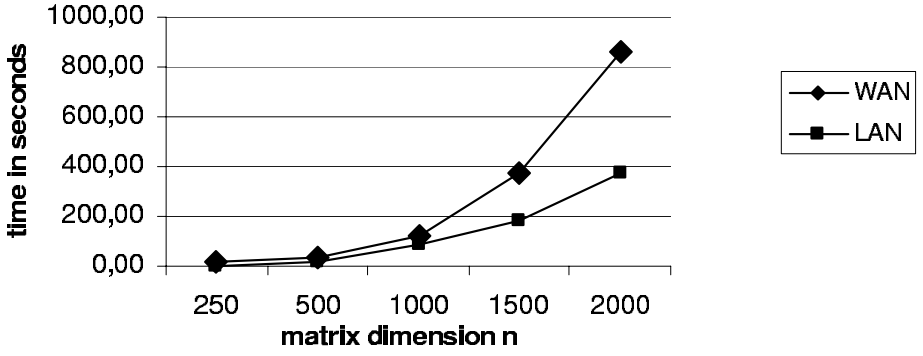


Fig. 4. Average execution time for the (I,J,K) of the matrix multiplication algorithms for the LAN and WAN NetSolve infrastructures

rate NB^3 of a sequential execution, then in this case the proposed approach to the distributed computing is not competitive.

However, in general it is worth to note that efficiency in distributed computing applications is a wrong performance's measure. Actually, the use of a distributed computing infrastructure should not be seen as a high performance computing solution but as a cost / effective solution because it enables the access to computing resources that cannot be conveniently locally located.

4 Conclusions

The development of cost/effective distributed computing environments is enabled by the current advances in networking technologies. The network-based computing environments provides the computational and storage requirements for solving distributed applications. Even thought distributed computing environments can deliver high performances when lot of computations is needed to do embarrassingly parallel tasks, the availability of distributed software infrastructure such as NetSolve and Condor represent a viable and economic solution when other dedicated resources are unavailable. This often requires detailed understanding of the underlying architecture and writing parallel or distributed algorithms needs exploitation of different programming models which are most suitable for addressing communication and synchronization issues. To this aim, we discussed performance behavior of a client/server implementation of some block matrix multiplication algorithms in terms of their synchronization overhead in order to analyze bottlenecks in the algorithms that critically impact performance. While in parallel computing we decompose into parts, in distributed computing we assemble parts [6], and in some cases, composition requires hard synchronization issues.

References

1. D.Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar. User's Guide to NetSolve V. 2.0. Univ. of Tennessee, 2004. See also NetSolve home page - URL: http://icl.cs.utk.edu/netsolve/index.html
2. O. Beaumont, V.Boudet, F. Rastello and Y. Robert. Matrix Multiplication on Heterogene-ous Platforms. in IEEE Trans on Parallel and Distributed Systems, vol. 12 (2001), pp 1033- 1051
3. J. Choi, J. J. Dongarra, and D. W. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms. Concurrency: Practice and Experience, Vol. 6 (1994), pages 543-570.
4. J. Dongarra. Performance of Various Computers Using Standard Linear Equations Soft-ware, (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89-85, 2005. See also Top500 Supercomputers home page at URL: http://www.top500.org
5. I. Foster and C. Kesselman. Computational Grid. in The Grid: Blueprint for a Future Gen-eration Computing Infrastructure. Foster and Kesselman eds., Morgan Kaufman, 1998.
6. G. Fox. Message Passing from Parallel Computing to the Grid. in IEEE Computing in Sci-ence and Engineering. Sept/Oct 2002
7. GARR home page. - URL: http://www.garr.it
8. M. Litzkow, M.Livny and M.Mutka. Condor: a hunter of idle workstation. In Proc. of 8-th Int. Conf. Distributed Computing Systems, pp. 104-111 IEEE press, 1988. See also URL www.cs.wisc.edu/condor
9. A. Kalinov and A. Lastovetsky. Heterogeneous Distribution of Computations Solving Lin-ear Algebra Problems on Networks of Heterogeneous Computers. Journ. of Parallel and Distributed Computing, Vol. 61 (2001), pp. 520-535
10. Seti@home home page - URL: http://setiathome.ssl.berkeley.edu/

Best Practices of User Account Management with Virtual Organization Based Access to Grid

Jiří Denemark[3], Michał Jankowski[2], Aleš Křenek[1], Luděk Matyska[1,3], Norbert Meyer[2], Miroslav Ruda[1], and Paweł Wolniewicz[2]

[1] Institute of Computer Science, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
{ludek, ruda}@ics.muni.cz
[2] Poznań Supercomputing and Networking Center,
ul. Noskowskiego 10, 61-704 Poznań, Poland
{jankowsk, meyer, pawelw}@man.poznan.pl
[3] Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
jirka@ics.muni.cz

Abstract. Scalable and fine-grained Grid authorization requires the move away from gridmap-file based access control and 1-to-1 mappings to individual operating system user accounts. This is recognized and addressed by virtual organization authorization services and user management systems e. g. Virtual Organization Membership Service (VOMS), Local Centre Authorization System (LCAS), Local Credential MAPping Service (LCMAPS) and Community Authorization Service (CAS). They do, however, not address user operating system account management and isolation/sandboxing requirements, such as flexible pooling of accounts while maintaining auditing records. In this paper we compare existing systems which solve the above shortcomings and are currently used in real production grids.

1 Introduction

The main aim of user management systems is to provide controlled and secure access to grid resources. Security requires authentication of the user and authorization based on combined security policy from the resource provider and the virtual organization of the user. Users must be allowed to use the resources to the extent allowed by the user roles and the policy, while resources must be secured against unintentional as well as malicious policy breaks. The second important thing is the possibility of logging user activities for accounting and auditing (security reasons) and then gathering these data or their aggregates both by the resource provider and virtual organization of the user. From the user point of view, an important feature is single sign-on and automatic authorization based on user identity and his/her privileges and roles. The problem of user management is a non-trivial one in an environment of thousands of users participating in many virtual organizations that provide a huge number of computing and storage resources and data. The complexity rises from the point of view of time

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 633–642, 2006.

required for administration tasks and automation of these tasks. While there are many attempts to solve the mentioned problem, none of them, according to our best knowledge, solves it in complex and satisfactory way. The simplest solutions that are scalable and easy to manage are using anonymous or shared accounts and similar insecure approaches. Such solutions do not offer isolation of activities of different users which is necessary for user and resource security as well as for proper accounting and logging user activities. More sophisticated solutions use set of accounts that are one-to-one (at the time) mapped to the users, however the history of mappings is not stored, thus accounting and logging is still impossible.

Few projects like Perun [1, 2] and VUS (Virtual User System) [3, 4] aim to overcome most of these deficiencies. However, these projects originated from different motivations and needs, therefore the concepts and semantics of some basic ideas differs. In this paper we will present the main ideas and architectural features of these existing systems running in real production grids.

2 VO Based Access to the Grid

One of the basic concepts in grid theory is Virtual Organization (VO). A VO is a set of individuals and/or institutions that allows its members sharing resources in a controlled manner, so that they may collaborate to achieve a shared goal [5].

The first step in obtaining an access to a remote resource is authentication. From the user point of view, the remote access should be as much as possible simple and similar to the local access. This may be achieved by single sign-on and credential delegation features and by integration with local security solutions. These requirements are usually fulfilled by most of already implemented solutions.

The next important issue is authorization. The security policy should be combined from few sources, at least from VO and resource providers. This allows for delegation of some administrative privileges and work to VO, which is more scalable. The authorization must be fine-grained, which means the user privileges must be limited according to their VO membership, roles and capabilities.

The system must assure that two different jobs will not interact in unwanted and unpredictable way, e. g. overwriting each other results. Closely related issue is security of data and files and protecting them from unauthorized read or write. Moreover, it must be possible to identify who and when used specific resources, performed or attempted some actions, which is relevant for accounting, auditing, and for security reasons. Usually it is not even enough to know who it was personally, but it is important on whose behalf he acted and in which role. By the virtual environment we understand such encapsulation of user jobs that will both guarantee the limited set of privileges and also provides support for identification of user and organization on behalf he/she acts. Virtual accounts, sandboxes, and virtual machines are examples of different approaches to the creation of virtual environments [6, 7, 8, 4].

Any production grid, especially commercial one, needs an accounting feature. Before issuing any bills however, the accounting data must be collected (possibly from several locations) and tied to users and VOs. From the security point of view, it is important to track user activities (e. g. by analyzing system logs) in order to detect any rule-breaking. It must be possible to identify the user who has performed the action. Proper encapsulation of jobs and history of users' mapping into virtual environment are both necessary to achieve these goals.

3 Perun—from Cluster to VO Management

3.1 Motivation

Project Perun [1] started as a user management tool for the Czech national Grid—*META Centre* project—encompassing heterogeneous resources from supercomputing centers across Czech Republic. *META Centre* consists of a mix of PC IA-32 clusters, SGI and Digital supercomputers and several smaller computers of various architectures (Itanium, Power5, Opteron).

The design of the system was driven by several rather contradictory requirements: e. g. fault-tolerance of managed resources (probability of hundreds of machines being up and reachable at the same time approaches zero) vs. enforcement of integrity constraints (changes of related configuration parameters should be done simultaneously and atomically), or the configuration repository being central (easier to manage) vs. distributed (independent control of distinct administrative domains).

The system itself is not a symmetric component of authorization service. On the contrary, it provides a repository of complex authorization data, as well as tools to manage the data. The data are used to generate configuration of the authorization services themselves (starting from UNIX user accounts to gridmap-files or VOMS database). In turn, these services are used to enforce authorization policies. Hence the centralized Perun architecture does not introduce a single point of failure of the whole Grid authorization infrastructure.

3.2 Architecture

Having analyses the principal requirements (see above) we decided to take the approach of central configuration repository which models an *ideal world*, i. e. how the resources should look like. In this central repository, all the necessary (and possibly very complex) integrity constraints are relatively easy to be enforced. The repository is complemented with a change propagation mechanism which detects the changes, generates consistent configuration snapshots of atomic pieces of managed systems, and tries to deliver them to their final destinations, dealing with resource or network failures appropriately. In this way, the *real world* is forced to follow the ideal one as closely as possible. Details are discussed in [2].

The core of the system is completely independent on the structure and semantics of the configuration data, hence the system is easily extensible. In the

current deployment the managed configuration items include user accounts on UNIX machines, Kerberos realms, and user home directories on both UNIX filesystems and AFS.

Security of such system is critical. First of all, compromise of the central configuration repository implies compromise of all the managed resources. Therefore we attempt to minimize possible impact by following the minimal required privilege principle. In other words, even if a typical update of the managed computer, e. g. adding users to `/etc/passwd`, requires root privilege, the system does not require the credentials of the central server to be granted an unrestricted root access to the managed computers. On the contrary, only the necessary operation is allowed. For example, the central server can manipulate only user accounts in `/etc/passwd` but it can neither touch system accounts nor modify other files.

The managed resources span different administrative domains. Therefore the individual administrators must be isolated when accessing the central repository. Technically this is done by denying direct access to configuration database tables and allowing it via special database views only. These expose only an appropriate set of rows, selected dynamically based on the user's authentication information and access control lists.

The current implementation uses Kerberos 5 as the primary authentication mechanism, for both users and inter-service communication. However, X509 certificates are also supported, being translated to Kerberos identities wherever required.

Both resource administrators and ordinary users access the system via web interface for routine tasks, e. g. initial user registration procedure, annual report submission, temporary locking and unlocking unused accounts etc. In addition, the administrators may use command-line interface, intended for building ad-hoc scripts for solving particular tasks. Finally, the administrators are given full SQL access to the database (applying the security restrictions described above) to resolve exceptional situations.

3.3 Grid Extensions

With our involvement in international Grid projects GridLab and EGEE, Perun was extended to cover additional requirements of these. First, support for binding X509 identities to physical users, and maintenance of a set of trusted certification authorities were added. The data are used for generating *gridmap-file*s (mapping of X509 certificate subjects to local user names) directly. The files are either propagated together with `/etc/passwd` using standard Perun's mechanism or published on a web site from where they are picked by the managed computers actively.

The EGEE project uses two different authorization services: LCMAPS and VOMS. The current Perun implementation supports both. For LCMAPS it generates and publishes LDIF files which are picked by the service and turned into local *gridmap-file*s. VOMS is controlled directly via its administrative interface.

3.4 Deployment

***META Centre* (national Grid).** In the *META Centre* project, the system manages over 200 nodes of IA-32 clusters, 4 SGI supercomputers, and about 50 other, rather heterogeneous machines. The machines are located in several cities in the Czech Republic. There are almost 800 user records in the database, however, a typical number of users authorized to access a single resource is less than 300.

Gridlab (EU 5FP project). Perun was used to manage specific GridLab user accounts on the machines forming the project testbed. In this case, the machines were spread through the Europe, with some machines even overseas. This deployment motivated the described introduction of X509 certificate management and generating *gridmap-file*s.

VOCE (within EU 6FP project). Perun is used for the management of *Virtual Organization for Central Europe* (VOCE)[1] in the EGEE project. Besides controlling the LCMAPS and VOCE authorization services it also manages dedicated VOCE user-interface machines.

Deployment in VOCE brought the described authentication of users and administrators with X509 certificates.

3.5 Further Development

The current integration of Perun and batch system in *META Centre* is limited—the only configuration maintained by Perun is a file with a list of user accounts on particular machines and clusters, serving as a hint for job planning. We plan introducing a more general virtual accounts or virtual machines environment, managed by Perun, more tightly coupled with the batch system.

In a longer timeframe scalability issues should be addressed. The current bottleneck is the update of managed machines—the server opens a network connection to the machine, and waits until the configuration update is finished. This is proven to work with hundreds of machines but will not probably scale further. Instead a hierarchical configuration propagation should be used.

4 Virtual User System

4.1 Motivation

Since 1999 some supercomputing centers in Poland have been connected using the LSF (Load Sharing Facility) HPC Cluster. It was taken into consideration that such solutions help distribute jobs across the multicluster, what enables better machine utilization. However, the queuing systems doesn't care of distributed user account management, and thus Virtual User System was introduced to handle the problem. It was exploited and proved the usefulness of concept. In the recent years the concept of grid computing becomes more and more popular and

[1] `http://egee.cesnet.cz/en/voce/index.html`

Globus Toolkit (GT) become a de facto standard in building grids. However, GT still doesn't address most of the problems connected to user management, so the idea of VUS was adopted and VUS was reimplemented in few grid projects.

Virtual User System (VUS [9, 4])[2] is an extension to the system that runs users' jobs (e. g. scheduling system, Globus Gatekeeper, etc.) and allows running jobs without having a personal user account on a node. The personal accounts are replaced by ""virtual" ones, that are mapped to users only for time needed to fully process a job. In this way, it is no need to maintain user accounts for all users on each computing node, reducing thus administrative overhead associated with such environment. VUS addresses most of the problems mentioned above like user authorization, isolation of work of different users, logging user activities and accounting.

4.2 Architecture

There are groups of virtual accounts on each computing node. Each group consists of accounts with different privileges, so the fine grain authorization is achieved by selecting appropriate group by the authorization module. The authentication module is pluggable and may be easily extended or replaced. For example, the authorization decision may be based on VO-membership of the user: the plugin may call a remote VO service to ask about the user or get his VO name from the credential signed by the VO (see VOMS [10]). The VO manager decides who is member of VO. The grid node administrator (RP) decides which VOs should be authorized, defines connection VO-account group and configures privileges of accounts. The second plugin may check if the user is present in a banned users list, maintained by the (site) administrator. This gives RP enough power while part of this power and most of the administrative work is delegated to VO manager. In that way security policies of VO and RP are combined.

The system assumes, that the VOs form DAG-shaped hierarchies. There are few root-level organizations and they are rather static (long-living). Stepping down the hierarchy there are more and more sub-organizations and they tend to be more dynamic. Thus, it is not easy or even possible to keep an up-to-date picture of the whole hierarchy on each node, so the nodes may reflect only some part of this hierarchy. A VO, that is not explicitly configured, is considered to be mapped to the same group of accounts as its parent VO. Note, that this mapping is similar to the gridmap-file, but is done on a higher level, so it requires considerably fewer configuration work.

Once the user is authorized, he is mapped to a virtual account from the group of accounts suggested by the authorization module. The mapping user-account mechanism assures that only one user is mapped to an account at any given time. The history of user-account mapping is stored in a database, so that accounting and tracking user activities are possible. The basic accounting data types (e. g. CPU, memory, etc.) are the same as offered by the standard

[2] `http://vus.psnc.pl/`

operating system accounting and are gathered automatically by VUS, using the local OS mechanisms. The VUS database is ready for storing also non-standard accounting, defined by the administrator and stored in the database by calling API functions by user applications (e. g. it may be time of usage of scientific equipment in a virtual laboratory driven by the application). VUS offers also options for summarizing accounting data and presenting it in different views via webservice to the RP, VO manager and user.

The mapped accounts are released on demand. It is done automatically, when the account is no longer used (no job is running) and there is currently no free account to be mapped for a user starting a new job.

4.3 Implementation

The first implementation of VUS was an extension to the queuing systems (e. g. LSF) and it was successfully exploited a few years ago in the Polish national cluster which connected several HPC centers in Poland [3]. The current VUS is a Globus 'gridmap callout' and it has been implemented from scratch. A few versions of VUS are used in various configurations in several national and international projects:

SGIgrid[3] (Done in cooperation with Silicon Graphics). In this project VUS uses Resource Access Decision (RAD) service for authorization. Done with cooperation with Academic Computer Center in Gdansk TASK. The implementation is finished, the project is in deployment phase.

Clusterix[4] (National Cluster of Linux Systems). In this project, authorization method will be based on querying VO services about membership of the user. Done in cooperation with Technical University of Szczecin and Academic Computer Center in Gdansk TASK. The implementation is matured, but not completely finished.

GridLab[5] (EU 5FP project). VUS uses Grid Authorization Service (GAS) for authorization. Version implemented in pure C, doesn't require a database on the grid node.

Experience from the VUS system development and deployment will be used for designing the general framework for managing user jobs in grids within the EU 6 FP Network of Excellence CoreGRID[6].

5 Related Work

The simplest solution for scalability and management problems is using anonymous or shared accounts, but they do not offer isolation of activities of different

[3] http://www.wcss.wroc.pl/pb/sgigrid/
[4] http://www.clusterix.pl
[5] http://www.gridlab.org
[6] http://www.coregrid.net

users. A bit more sophisticated solution is the pool account system which consists of a set of accounts that are one-to-one mapped to the users. With the strict one-to-one mapping accounts for users never using the resource must be created and maintained, increasing the administrative overhead and creating a possible security hole (never used accounts should be avoided).

Several Grid environments employ the idea of virtual accounts that are temporarily mapped to a pool of physical accounts when needed. Such accounts are called virtual, scratch, generic, template or shadow accounts. In PUNCH users have their own logical user-accounts and the system manages its own physical accounts on remote resources and dynamically recycles them among users as necessary. Generic accounts can also be used in Condor and Legion, but in both systems it is recommended that users have their own accounts on every machine. Condor uses a nobody UID to run jobs for users that do not have an account in a Condor flock. Legion also manages the pool of generic accounts that are assigned for Legion use.

Simple "Dynamic Virtual Environments" approach allows creation of the job environment like e. g. Unix account and gridmap entry for the user on demand. This solution however requires the client for applying for the account and provides no automatic mechanism for releasing or recycling of accounts, creating potentially large account pools where individual accounts are used with very different frequency.

5.1 VOMS, LCAS and LCMAPS

The most popular system for managing user accounts for VO based access is VOMS with LCAS and LCMAPS. Virtual Organization Membership Service was developed in European Data Grid project in order to solve limitations of classic Globus gridmap-file user management approach. VOMS contains database with information on the user's Virtual Organization and group (sub-organization) membership, roles and capabilities. The service preserves it in a special format—the VOMS credential. The user, before starting a job must acquire the VOMS proxy certificate signed by his VO and valid for limited time. The extra authorization data is placed as a non-critical extension in the proxy, so it is compatible with not VOMS aware services. In order to take advantages of VOMS data, the Globus Gatekeeper was extended by LCAS and LCMAPS.

Local Center Authorization System (LCAS) is a service used on computing nodes in order to enforce local security policies. The authorization decision is based on user proxy certificate and job description and is made by pluggable authorization modules. VOMS-aware modules use user's VO membership, roles and capabilities for making the decision.

LCMAPS maps user to local credentials (UNIX account and group, AFS and Kerberos tokens), depending on user proxy certificate and job description. The mapping decision is based on user proxy certificate and job description and is made by pluggable authorization modules. Similarly to LCAS, VOMS-aware modules use user's VO membership, roles and capabilities for making the decision.

Some plugins implemented for LCAS and LCMAPS allow for the backward compatibility with the classic gridmap-file approach. Other ones allow for mapping users to accounts from pools of accounts, where the pool is selected depending on VOMS data. The plugins care that the user is always mapped to the same account and only one user is mapped to an account at the time. The system doesn't care of releasing unused accounts and doesn't address accounting issues.

6 Conclusion

Large scale Grids serving thousands of users need flexible and dynamic user account management system, that provides virtual environment for running jobs while securing both users and resource providers from unexpected interactions. The management of user accounts is closely related to the management of VOs, as the VO membership and role is the basic information necessary to provide appropriate virtual environment.

The Perun system provides robust user management database and tools to create and maintain user accounts on demand and keeps the basic authorization data (to be used through some authorization service). Although it is a centralized system, using consistently the push model with no on-line real-time query support it has been proved to work smoothly with hundreds of maintained machines, spread over wide area network. The complex constraint tests on the data stored in the database together with consistent termination checks on all remote operations guarantee its high robustness and reliability. The Perun system is VO neutral, able to support several VOs simultaneously.

On the other hand, the Virtual User System is primary focused on the flexible and dynamic management of user account pools and collecting accounting information. It does not keep data about VO membership, using external services to provide such checks. To optimize this process, the VUS assumes DAG-shaped VO hierarchies. Fine grain authorization uses groups of virtual accounts with different privileges, to which user identities are mapped by the authorization module.

Both systems represent different approaches to the user account management. However, they start with different requirements and even do not use the basic concepts (like VO or account) in the same way. To some extent they also cover complementary issues, but their combination and extension is not possible without a common conceptual and semantics basis (e. g. hierarchy in VOs vs. a flat VO structure). It is necessary to agree on more precise definitions of VOs, their membership creation and management, description of roles within VO, interaction of user (account) management system with the authorization systems (including the scalability and robustness issues), definition and management of accounting information etc.

Both systems were also build for a specific user communities, with specific use scenarios. Again, more work is necessary to provide wide coverage of possible use cases, from which new requirements will be derived. Based on the common concepts, this will lead to a new, more general architecture of the user account management system and its interaction with other Grid components. This paper

is the first step towards the general architecture, that is being build as part of the Network of Excellence CoreGRID.

Acknowledgment

The support from the NoE CoreGRID (IST-2002-004265) is highly acknowledged.

References

1. Křenek, A., Sebestianová, Z.: Perun – Fault-Tolerant Management of Grid Resources. In: Cracow'04 Grid Workshop Proceedings. (2004)
2. Křenek, A., Sebestianová, Z., Sitera, J.: Perun (In Czech). Technical Report 1/2004, CESNET, z. s. p. o. (2004)
3. Kupczyk, M., Lawenda, M., Meyer, N., Wolniewicz, P.: Using Virtual User Account System for Managing Users Account in Polish National Cluster. In Hertzberger, B., Hoekstra, A., Williams, R., eds.: HPCN Europe 2001. Volume 2110 of LNCS. (2001) 587–590
4. Jankowski, M., Meyer, N., Wolniewicz, P.: Virtual User System for Globus Based Grids. In: Cracow'04 Grid Workshop Proceedings. (2004)
5. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications **15**(3) (2001)
6. Keahey, K., Doering, K., Foster, I.: From sandbox to playground: Dynamic virtual environments in the grid. In: GRID'04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), Washington, DC, USA, IEEE Computer Society (2004) 34–42
7. Keahey, K., Ripeanu, M., Doering, K.: Dynamic creation and management of runtime environments in the grid. Workshop on Designing and Building Grid Services, GGF-9 (2003)
8. Keahey, K., Doering, K., Foster, I.T.: From sandbox to playground: Dynamic virtual environments in the grid. In: GRID. (2004) 34–42
9. Dymaczewski, W., Meyer, N., Stroiński, M., Wolniewicz, P.: Virtual Users Account System for Distributed Batch Processing. In Sloot, P., Bubak, M., Hoekstra, A., Hertzberger, B., eds.: HPCN 1999. Volume 1593 of LNCS., Springer-Verlag (1999) 1231–1234
10. Alfieri, R., Cecchini, R., Ciaschini, V., Dell'Agnello, L., Frohner, A., Gianoli, A., K.Lentey, F.Spataro: VOMS: an Authorization System for Virtual Organizations. In: 1^{st} European Across Grids Conference, Santiago de Compostela (2003)

Manageable Dynamic Execution Environments on the Grid Using Virtual Machines

Sai Srinivas Dharanikota and Ralf Ratering

Intel Corporation, Software & Solutions Group, Parallel & Distributed Solutions Division, Hermuelheimer Str. 8a, 50321 Bruehl, Germany
{sai.dharanikota, ralf.ratering}@intel.com

Abstract. The service-oriented model of the grid offers a wide range of computational capability that can be shared over the Internet. In this paper we describe a concept to create and configure execution environments that can be dynamically deployed using virtual machines onto bare hardware. This new paradigm allows more flexible usage of available hardware and is expected to decrease the efforts needed to configure and maintain nodes on a grid. A novel aspect of the architecture is the use of Web Services Resource Framework (WSRF) [4] and other emerging standards in the grid ecosystem. Based on these standards a prototype has been implemented that is used to create first results, and figure out challenges and future work following this new approach.

1 Introduction

The grid computing paradigm has occupied an important place in the computing world, offering tremendous computing capability. The goal of grid computing is "to provide flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [1].

To make a system available on one of the current grid infrastructures like Globus [10] or Unicore [11], an administrator would have to install server software and edit configuration files for the available applications. This kind of static system configuration limits the usage of existing hardware to the type and version of operating system and installed applications. Moreover, multiple users from different organizations may share the same operating system, which leads to security issues, and the stability of the overall hardware is affected if a user executes a faulty job.

Virtual Machine Technology [7, 8], which is able to support execution of multiple guest operating systems answers the above mentioned problem. Technologies like Xen proved that performance loss of using virtual machines on various classes of applications is less than 5% [2, 9]. With the Virtual Machine Technology and the proven performance impact on the applications in mind, we suggest creating separate execution environments by mapping a user to an operating system, instead of mapping multiple users to the same operating system. Virtual machines, which are characterized by strong isolation and serialization can

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 643–650, 2006.

run and crash independently, and also encapsulates the data in a protected environment. A virtual machine can also be dynamically configured according to the hardware and software request that has been specified by the user. Advanced features of virtual machines include migration during run-time, which opens new ideas for highly flexible resource management: Since, executing a job is no longer bound to a certain system, execution environments could be migrated to another system during run-time (For instance, if a system is exclusively needed by a user with high priority or if it needs to be shut down for maintenance, currently running jobs may migrate with their execution environments to other systems that are currently available with appropriate hardware).

The idea behind proposing the solution is to achieve the following: provide an execution environment in terms of needed software (operating system, applications) and virtual hardware (main memory, secondary memory), make use of the isolation [6] property of the virtual machines so that the jobs can be executed in an isolated manner without affecting the whole system, improve security, simplify the task of the administrator, and open a new idea of resource management as described above.

The rest of the paper describes our solution and the approach followed to tackle the above mentioned problems. Section 2 discusses the concept of *Manageable Dynamic Execution Environments*. Section 3 presents our architecture of providing execution environments. Section 4 concentrates on the implementation details of our basic prototype. Section 5 summarizes the whole discussion and points to the future work.

2 Concept of Manageable Dynamic Execution Environments

Before going into the underlying principles of manageable dynamic execution environments, let us have a look at how the interactions with grid are perceived. Normally, the grid interactions and the problems associated with it are treated as a *single-fold problem*: mapping jobs to the available resources. This level of abstraction restricts the mapping of jobs to the resources, due to the lack of necessary execution environments, administrative privileges that are needed to provide the environments, security reasons or due to some special requirements. This makes the *mapping problem* more complicated.

To address this issue, we treat the present grid interactions as a *two-fold problem*: mapping environments to the resources and in-turn mapping jobs to the environments. As a solution to this *two-fold problem* we have introduced the concept of *Manageable Dynamic Execution Environments*.

Manageable Dynamic Execution Environments are environments in terms of software (operating system, applications) and virtual hardware (RAM size, disk size), which are custom-made for the job, and those that can be managed according to the needs of the user. The process of customization in our concept involves, capturing the requirements of the user and providing an execution environment for the job. A negotiation [3] may take place during this process. Our

concept is based on the emerging Web Services and grid standards like WSRF, JSDL [5] etc. We proceed further by explaining the requirements description and modeling of execution environments using virtual machines.

2.1 Requirements Description

Identifying the requirements that are needed for the job is the key in providing an execution environment. In order to achieve this goal, we represent the environment that is needed for the job as a *Job Submission Description Language* (JSDL) description.

```
<xsd:element name="JobDescription">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="jsdl:JobIdentification"/>
      <xsd:element ref="jsdl:User"/>
      <xsd:element ref="jsdl:Applications"/>
      <xsd:element ref="jsdl:Resource" maxOccurs="unbounded"/>
        . . .
      </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The pseudo XML schema description that is part of JSDL specifies the needed execution environment, which includes hardware, operating system, applications etc. The required hardware and operating system is described as a *Resource* element and the needed applications are described as an *Application* element in the *pseudo XML schema description.*

2.2 Virtual Target System (VTS) Properties

Based on WSRF as framework for our virtualization concept, we describe the execution environments as Web Service Resources [4] by using the following definitions:

A **Target System (TS)** resource represents a system that can execute jobs or perform file operations. Typically each physical machine that is available on the grid would be represented as a Target System in our approach.

A **Virtual Target System (VTS)** resource is a virtual machine that hosts an execution environment for specific job requirements. As such, it is a specific type of Target System that inherits all Target System properties in addition to the properties concerned to a virtual execution environment.
Every VTS is defined by the so-called resource property document:

```
<xsd:simpleType name="VMStateType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="running"/>
    <xsd:enumeration value="paused"/>
    <xsd:enumeration value="shutdown"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:element name="VirtualTargetSystemProperties">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="VMState" type="VMStateType" default="shutdown"/>
      <xsd:element ref="vts:Name"/>
      <xsd:element ref="vts:Description"/>
      <xsd:element ref="vts:Storage" maxOccurs="unbounded"/>
      <xsd:element ref="vts:Processor"/>
      <xsd:element ref="vts:Memory"/>
      <xsd:element ref="vts:OperatingSystem"/>
      <xsd:element ref="vts:Applications" maxOccurs="unbounded"/>
      <xsd:element ref="vts-rl:CreationTime"/>
      <xsd:element ref="vts-rl:TerminationTime"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

The properties of the Virtual Target System contain the *VMState* element that can be either *running*, *paused* or *shutdown* (VM image containing no running processes). In addition to the actual hardware resources, virtual hardware resources like the virtual machine main and secondary memory are described as *Memory* and *Storage* elements. Proceeding further, the description contains properties related to the operating system (*OperatingSystem*) and the available applications (*Applications*) on the Virtual Target System. Finally, the description contains *CreationTime*, which describes the time when the Virtual Target System has been created and *TerminationTime*, which describes the time when it will be destroyed. This WS-ResourceLifetime [13] mechanism prevents users from accidentally blocking valuable hardware resources with their virtual machines.

3 Architecture

Providing *execution environments* adds a new level of abstraction to interact with the grid. The way of providing dynamic execution environments that can be managed, and the way of mapping execution environments to the actual resources and in turn mapping jobs to the execution environments is sketched below.

3.1 Providing and Managing Execution Environments

Figure 1 gives an overview of the various components and services involved in the architecture of providing dynamic execution environments, which can be managed.

To start with the initial setup, the administrator creates OS Image resources (WS-Resources) for all the OS Images available in the repository by contacting the *OS Image Factory*. During this creation process, the references (*qualified endpoint references* [4]) to the OS Image resources and the properties of the OS Images are stored in the registry.

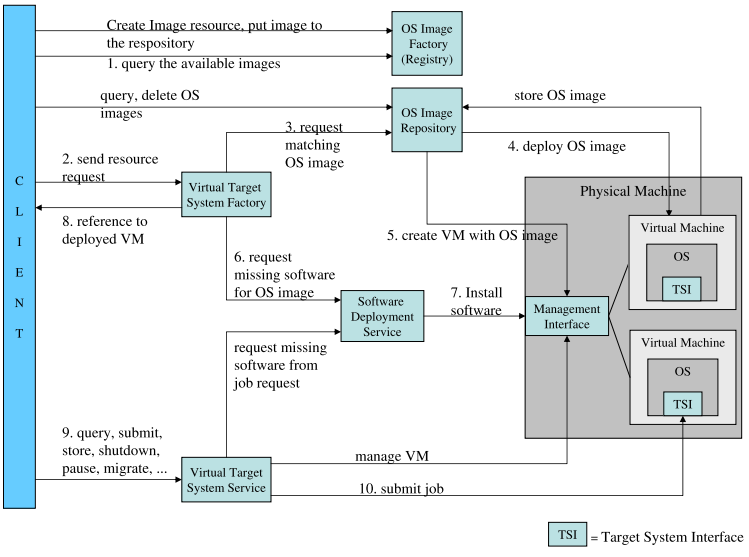


Fig. 1. Overview of Manageable Dynamic Execution Environments architecture

1. The client can query the available images on the repository machine with the help of the registry.
2. In order to create an execution environment for the job, the client sends a request to the *Virtual Target System Factory(VTSF)*. The request contains the needed OS, and is specified using the Job Submission Description Language, as described in Section 2. The factory is responsible to create the Virtual Target System resource (WS-Resource).
3. The *VTSF* analyzes the request and selects the best Target System for the deployment of VM. A request (Target System, OS, RAM etc) is sent to the *OS Image Repository* (service that is available on the repository machine) for the deployment of the OS Image. A set of pre-configured OS images, including information about the applications and a configuration file needed to boot the virtual machine, are maintained on the repository machine.
4. The *OS Image repository* transfers the OS image and a VM configuration file to the Target System.
5. After the image and the configuration file are transferred, the *OS Image Repository* interacts with the *Management Interface* to create the VM with the specified properties and checks its boot status.
6. In case of software request that is not available in the OS image, the required software is requested from the *Software Deployment Service*. This service maintains a repository of packages and configuration files that are needed to make the software available on the Virtual Target System.
7. The *Software Deployment Service* deploys (related work in [14]) the software on the created Virtual Target System through the *Management Interface*.

8. After the deployment is completed, the *Virtual Target System Factory* creates a WS-Resource that represents the new VTS and returns its reference to the client. The reference is used to identify and access the new VTS.
9. The *Virtual Target System Service* is used to submit jobs to the VTS, query its properties and perform other VTS management operations on it. Every request to the service contains the reference that has been returned in step 8 so that multiple VTSs can be managed through the same service. If a job submitted from the client requires software that has not been installed in the initial deployment step, the service will initiate the installation process through the *Software Deployment Service*. Besides migration of VTSs to other machines, it is also possible to store the configured OS image that contains the new applications back to the OS Image Repository for later usage.
10. Finally, the job gets executed using the *Target System Interface* that is preconfigured in every OS image, which gets started on every VTS during the boot sequence.

4 Implementation and Results

This section describes the implementation of the prototype that demonstrates a tight integration of virtual machines into WSRF-based grid infrastructures. It is also being used for benchmarking and to discover possible drawbacks and bottlenecks of the described approach.

4.1 Prototype Implementation

All prototype components have been implemented in Java. The communication is done through web service protocols and standards. The only component using Java RMI instead of web services is the Target System Interface. Opting this way would avoid running a web server on the VM.

The OS images are compressed and maintained on the repository machine. Each OS image is associated with an XML file that describes the properties of the corresponding Virtual Target System before it is deployed and booted, and a configuration file needed to boot the VM. The files are transferred to the Target System by secure copy protocol. As Virtual Machine Monitor we are using Xen, because of its para-virtualization architecture and good performance score.

4.2 Experimental Setup and Results

Experiments have been conducted on three IBM M50 machines with the configuration: Intel Pentium 4, 3 GHz, 1 GB RAM and 120 GB IDE disk. The machines are connected by a 100Mbit/s switched Ethernet network. All the machines are setup with Xen as the virtual machine monitor. For all the tests, the *Virtual Target System Factory*, *Virtual Target System Service*, *OS Image Factory* and the *OS Image Repository* are setup on one machine and the other two machines are used for VM instantiation. We have timed the important steps

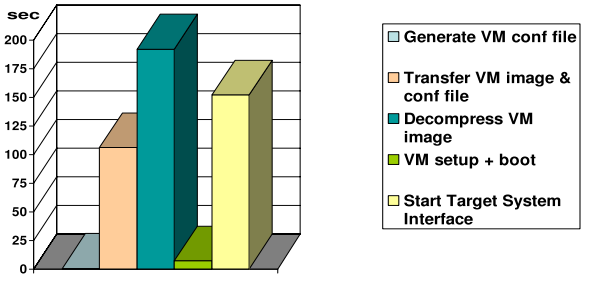


Fig. 2. Graph showing the important steps and the time taken for each step before the job gets executed

involved in our prototype implementation. Figure 2 shows the time elapsed from the moment Virtual Target System Factory receives a request until the Virtual Target System is ready for job execution. In the diagram we can observe that deploying the VM image consumes the most time. Transferring a compressed VM image with a size of around 1 GB takes nearly 106 seconds and decompressing the VM image takes nearly 192 seconds. This seems to be the main bottleneck in the overall concept, in comparison to the VM setup and boot time of 7 sec.

Starting the TSI component at the boot time of a Xen-based virtual machine takes around 152 seconds, which is a factor of 8 in comparison with the startup of the TSI on a machine where Xen is not involved. We observe that TSI object to be available in the RMI registry on a Xen-based virtual machine takes more time (factor of 8), which led to the ongoing discussion with the Xen community [12].

5 Conclusion and Future Work

We have described the architecture of "Manageable Dynamic Execution Environments" using virtual machine technology. We have mentioned how we modeled the execution environments using virtual machines by assigning properties and treating them as Web Service resources that can be managed.

Integrating virtual machines into the grid definitely offers flexibility, efficient utilization of resources and improved security, but also points to the middleware challenges. As we have seen, instantiation of virtual machines doesn't take much time once all the needed data (VM disk, configuration file) is available on the Target System. With this point in mind we have identified the need for efficient techniques to transfer and decompress large virtual machine images which are in the order of gigabytes.

This paper focuses only on the virtual machine instantiation. The directions for future work include deploying the needed environments for the jobs onto the virtual machines, resource scheduling, networking and security.

References

1. I.Foster, C.Kesselman, J. Nick and S.Tuecke.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Int. J. Supercomput. Appl (2001).
2. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield.: Xen and the Art of Virtualization, ACM Symposium on Operating Systems Principles (SOSP) (2003).
3. Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Jim Pruyne, John Rofrano, Steve Tuecke, Ming Xu.: Web Services Agreement Specification (2004).
4. Czajkowski, Donald F Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukin, David Snelling, Steve Tuecke, William Vambenepe.: The WS-Resource Framework (2004).
5. JSDL working group, GGF GridForge (2005), https://forge.gridforum.org/projects/jsdl-wg/document/jsdl.xsd/en/12.
6. R. J. Figueiredo, P. A. Dinda, Jose A.B.Fortes.: A Case For Grid Computing On Virtual Machines, in 23rd International Conference on Distributed Computing Systems (2003).
7. Xen virtual machine monitor, University of Cambridge, http://www.cl.cam.ac.uk/Research/SRG/netos/xen/.
8. VMware workstation, http://www.vmware.com/products/desktop/ws_features.html.
9. Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, Jeanna Neefe Matthews.: Xen and the Art of Repeated Research, in FREENIX 2004.
10. Globus Toolkit (2005), http://www-unix.globus.org/toolkit/.
11. Unicore project (2005), http://unicore.sourceforge.net/.
12. Xen mailing lists archives, http://lists.xensource.com/archives/html/xen-devel/2005-04/msg00860.html.
13. Jeffrey Frey, Steve Graham, Karl Czajkowski, Donald F Ferguson, Ian Foster, Frank Leymann, Tom Maguire, Nataraj Nagaratnam, Martin Nally, Tony Storey, Igor Sedukhin, David Snelling, Steve Tuecke, William Vambenepe, Sanjiva Weerawarana.: Web Services Resource Lifetime (2004).
14. ProActive, http://www-sop.inria.fr/oasis/ProActive/

Semantic-Based Grid Workflow Composition

Tomasz Gubała[2,3], Marian Bubak[1,2],
Maciej Malawski[1], and Katarzyna Rycerz[1]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] Section Computational Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam
{bubak, malawski, kzajac}@uci.agh.edu.pl, gubala@science.uva.nl

Abstract. The work presents a solution to abstract workflow composition in a semantic Grid environment. Along with analysis of the problem of workflow composition and the description of related research in that matter, we present the Workflow Composition Tool. The tool is designed to provide descriptions of abstract (i.e. not executable) workflows of service-based Grid applications. The tool applies novel semantic techniques to deliver meaningful discovery and matching of ontologically described resources. WCT is a part of larger workflow composition and execution system being developed in the K-WfGrid project – the short description of the entire system is also included.

Keywords: scientific workflow, workflow composition, grid services, semantic matchmaking.

1 Introduction

Scientific workflows are an interesting field of research in modern computer science. Projects such as myGrid [9], GriPhyN [5] or Kepler [7] use a workflow-based solution to some scientific, computation and data intensive problems. Here we present the Workflow Composition Tool (WCT)designed to search for solutions for a given problem in form of application workflow.

This paper describes which mechanism of **abstract and dynamic** workflow composition we employ and how our tool is designed. According to our definition, *abstract workflows* represents a Grid application, its internal structure and data/control flow on a functional level with no execution details. By *dynamic workflow* we mean an application described through data and/or control flow structures which is executed by a dedicated engine but the internal structure of which may change during execution. That does not only involve choosing another computational resource to run software when the original one is not available – such dynamic workflow rescheduling may provide the functionality to rethink some parts of workflow which are not executable at the moment in order to devise a better strategy to obtain the demanded results.

This paper is organized as follows: in Section 2 we describe the tool in the context of the K-WfGrid project, in Section 3 the WTC tool is described, and we summarize in Section 4.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 651–658, 2006.

1.1 Similar Work

There are several approaches that aid in application composition. First of them is the idea of code generation tools being able to automatically construct a working workflow for a given problem. Applications of these solutions appear for in the Web services related technologies [3, 14] and in distributed component architectures [8]. The later proposition for automatic workflow generation in the Common Component Architecture (CCA) is applicable together with the concept of application factories which provide ad-hoc application execution in distributed environment as a computational service [12].

Similar research initiatives are providing algorithms for automating service construction based on various algorithms like AI planning [2] or situation calculus [16]. The important feature of these solutions is the semantic service interface description: nowadays scientists tend to use ontological descriptions to add a certain meaning to parts of a service description [1, 11].

There are two main standardization projects in the field of semantic web service description, namely OWL-S [15] and WSMO [10] and both of them are related to the Semantic Web initiative.

2 Scientific Grid Workflows in K-WfGrid Project

In the K-WfGrid project colored Petri nets are used for workflow representation. This representation supports data and control flow, it enables dynamic changing of the flow during runtime, and it also allows several levels of abstraction.

2.1 K-WfGrid Workflow Environment

The K-WfGrid workflow environment was designed to take control over all steps of workflow construction and execution. The main module is the Grid Workflow Execution Service (GWES), which interacts with the user by means of a portal. GWES uses WCT, the Automatic Application Builder (AAB) [6] and the Scheduler components to assist in composition, refinement and execution of workflows in subsequent steps. WCT (Workflow Composition Tool) is used to build an abstract workflow from initial requirements provided by user. Subsequently, AAB maps each of abstract service class operations into a list of concrete Web service operations, providing variants of concrete workflows. Such a workflow is then executed with the help of the Scheduler, which selects the best Web service operation from the list, based on some policy-defined metrics (cost, speed, etc.).

2.2 Levels of Workflow Abstraction

Fig. 1 shows the transformations of workflow abstraction levels during the composition and refinement process in the K-WfGrid environment. The initial version of a workflow denotes very basic facts about future workflows like a description of desired result – such a basic requirement is wrapped into Petri-net notation with a single transition of abstract activity (operation). The first step,

performed by WCT and marked by the grey box in Fig 1, transforms this highly-abstract operation into an abstract workflow, built of abstract operations on Web service classes. In the next step, AAB maps the Web service class operation into a list of concrete WS operations that match the specified class. Subsequently, the Scheduler is responsible for selecting the concrete Web Service operation, concretizing this part of the workflow, which can now be executed.

The *service class* may be associatted with a service interface. It declares a set of operations, their input and output data and provides the semantic meaning of each (see Sect. 3.3). The operation of a service class thus gives a full description of the functionality but it contains no information on implementations of that functionality.

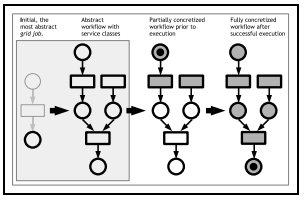


Fig. 1. Stages of workflow construction in K-WfGrid

The process of transformation of abstraction levels is performed in a dynamic way, so different parts of the workflow at a given time may reside on different abstraction levels. Such an approach is called deferred planning, which means the activities in a workflow remain abstract until they are brought to execution once their predecessors are successfully completed. This also allows for dynamic refinement of a workflow when completing some activity may result in alternate subworkflows.

The introduction of different levels of workflow abstraction is caused by the opportunity of workflow reuse. Referring back to Fig. 1 the gray rectangle encompasses the states of workflow which are reusable. The fully-defined abstract workflow produced by WCT is a de-facto description of an application with no details about execution parameters: e.g. it lists operations which are to be invoked but it does not decide which service instance should be contacted in order to use that operation. This means the workflow in its abstract form remains useful for a longer period of time, even taking into account the ever-changing nature of the Grid environment.

2.3 Use of Knowledge

In the K-WfGrid system, all semantic data is stored in the Grid Organizational Memory (GOM) [13], which contains OWL-based descriptions of service class functionality, instance properties and performance records. Fig. 2 sketches the flow of knowledge within the system during workflow construction and execution.

The conditions for the initial workflow are specified within a portal with the User Assistant Agent. WCT uses the knowledge on services class functionality, supplied to GOM by service providers. Subsequently, AAB uses information on service properties which result from the activity of a Knowledge Assimilation Agent, analyzing the monitoring data of resources. Workflows containing multiple possibilities of operations are then executed with the Scheduler, which depends on the knowledge gathered by performance analysis services.

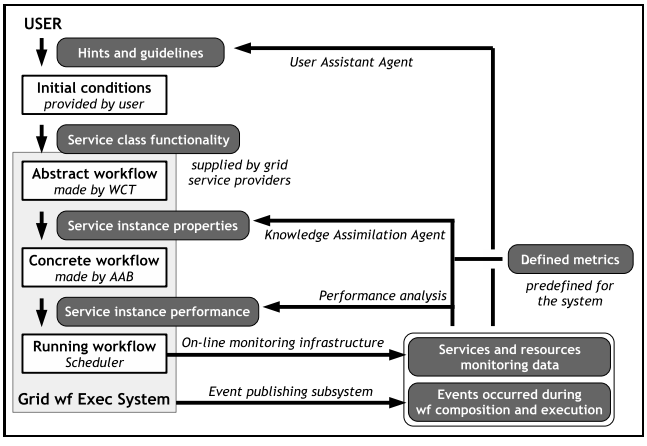


Fig. 2. Circulation of knowledge in K-WfGrid

During the execution of the service, online monitoring data and all events generated by workflow execution are gathered by respective agents, enriching the knowledge base stored in GOM and providing online status information to the user.

3 Workflow Composition Tool

3.1 Purpose of the Tool

The Workflow Composition Tool is a part of the dynamic workflow construction and orchestration environment being built in the K-WfGrid project. WTC transforms the very sketchy requirements provided by the user into a full description of an application workflow in its abstract form (more information on different levels of abstraction can be found in Sect. 2.2).

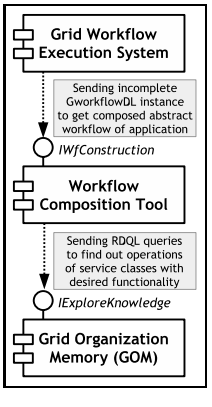


Fig. 3. WCT interfaces

Fig. 3 details the location of WCT inside the K-WfGrid architecture pictured in Fig. 2. There are two main interfaces where the tool connects to other parts of the system. First of all, WCT exposes an IWfConstruction interface which provides the main functionality of workflow composition. It is used by the Grid Workflow Execution System (GWES) when there is a necessity to construct a workflow.

Fig. 3 shows also how the tool depends on an external source of knowledge. The role of the source is attributed to the Grid Organizational Memory, a sophisticated ontology registry dedicated to knowledge storage and publication. The main interface is through a query-based interface which WCT uses in order to search through the GOM store of knowledge. The

queries are formulated using RDQL which is a query language designed for information stored in RDF form. Using that interface WCT tries to find out the published (i.e. available) classes of services needed in the context of certain workflows. More details on how this knowledge is used during the composition process are in the subsequent section.

3.2 Abstract Workflow Composition

In the K-WfGrid project there are various levels of workflow abstraction (as explained in Sect. 2.2). The purpose of the WCT process is to provide an abstract workflow so the execution engine may apply it (see the second stage pictured in Fig. 1). For explanation purposes let us assume that the Workflow Composition Tool is invoked with a sample workflow, still containing some parts that haven't been fully analyzed (e.g. just half of it is defined by the user and the other half is left for the system to generate).

The first step is to identify every section of the input workflow which needs further attention of WCT – the sections where no composition has ever taken place. In terms of our Petri net-based approach these are *transitions* denoting abstract "activity". The role of WCT is to transform these most abstract "activities" into subworkflows of service class operation invocations. The composer finds out the dependencies of these uncertain parts. Most of these places denote dependencies – for instance they may describe what kind of data the "activity" shall produce or accept. Having those requirements the tool forms a proper RDQL query and sends it to the Grid Organizational Memory (GOM) module. GOM parses the query, searches its internal ontology stores and issues a response listing every class of service which declares the required functionality through some operation.

What follows is the analysis of that response and the search for the most suitable solution. Afterwards the chosen service class operation is incorporated in the workflow description and the next iteration of the construction algorithm may take place. If there are several different operations chosen (e.g. all of them have the same suitability factor), all of them may be inserted into the workflow by means of the XOR-split/merge construct – a workflow pattern [17] designed specifically for that purpose.

The next iteration is needed as there can be new requirements generated by the insertion of a new service class operation into the workflow. The whole process ends when there are no more parts of the workflow with no proper service operation found or if there are no suitable service classes published in the registry which may fulfill the lasting dependencies. In the latter case the workflow becomes incomplete and cannot be fully executed, but it may still be stored as there is a possibility of completion in the future (where there are different service classes available) – or it may be executed with some assistance from other tools or the user.

3.3 Resource Semantics Used by Composition Process

The description of the knowledge flow in the K-WfGrid environment pictured in Fig. 2 (see Sect. 2.3) shows that WCT uses *service class functionality* in the course of its operation. That knowledge is based on semantic descriptions of service classes published by service developers and provided in the Grid environment. The service class is described with the OWL-S semantic web standard [15] which allows for description of different aspects of a service, including its interface and its invocation scheme.

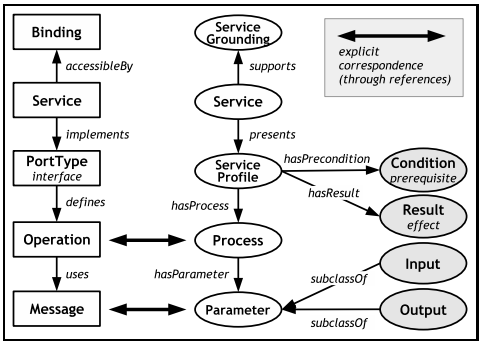


Fig. 4. Correspondence of some OWL-S and WSDL description elements

In Fig. 4 one may note a correspondence between a typical WSDL description of a web service and its counterpart in the ontology-based OWL-S standard. A rough equivalent of a port type in WSDL terminology is a profile element of the OWL-S description: it contains information on the functionality provided by a service.

The most important part for the WCT composer is the one with operation parameters of four possible types: condition, result, input and output. That quadruplet is a direct application of the IOPE (Input-Output-Prerequisites-Effects) paradigm and describes the semantics of input data consumed by an operation, the semantics of the output data produced by an operation and a list of conditions and effects needed or generated by an operation. By taking all that pieces of information into consideration, the tool is able to decide which service class operation is applicable in a certain workflow context.

3.4 Ontology-Based Comparison

The comparison of several possible solutions is a crucial part of the WCT internal workflow composition algorithm. The necessity of a choice of service class operation arises when there are many responses to a query returned by the GOM registry (see Sect. 3.2 for details). The decision on which service class to prefer is based on careful comparison of semantic descriptions of the obtained

services classes and the matching of the abilities of the classes with the actual requirements.

There are several parts of an operation description where the suitability of solutions may be compared: purpose (type) of an operation in terms of a certain taxonomy, input and output parameters of an operation and effects which take place during the operation execution. WCT is concerned mainly on two former options. Both the type of an operation and the semantic types of its I/O parameters may be compared to the requirements and a certain metric can be defined which computes the suitability level for a particular problem-solution pair. That metric (considered to be a method of computation rather than a *metric* in the mathematical meaning of the term) is based on certain rules of similarity between two distinct ontological instances. Many different rules can be applied here and as yet no decision has been made on which to use in the K-WfGrid environment – however from our previous experience [3] we prefer rules based on similarity of types and similarity of property value sets. The former rules compare the ontological classes of both entities and prefer the one semantically *nearer* to the perfect match – for instance, a member of a subclass is more suitable that a member of a superclass, as the former is necessarily inside the preferred class while the latter one isn't. The rules from the second set compare instance properties and find out which one has values more similar to preferred ones. Please note that the comparison of properties may be applied only to members of sufficiently similar classes.

4 Summary

In this paper we present our solution to the problem of workflow composition in a Grid environment. A description of the problem is provided along with a related work section. We also provide an explanation of the K-WfGrid workflow execution system and we picture where our Workflow Composition Tool is used, along with the purpose of the tool and how it provides the abstract workflow construction functionality.

The development of the solution presented in that work is not yet finished, however it is now past its design phase and the implementation of the tool is ongoing. According to the K-WfGrid schedule the first prototype release should be available in the fourth quarter of 2005. The tool is being developed in the Java programming language with the use of modern ontology-based reasoning engine Jena. WCT is devised to work with the mose recent Web service technology standards and in the future it will interface with the WSRF [4] standard and its Grid imlementation, i.e. Globus Toolkit 4.

Acknowledgements. This work was partly funded by the European Commission, Project IST-2004-511385 K-WfGrid and the Polish State Committee for Scientific Research SPUB-M. The authors would like to express their gratitude to Andreas Hoheisel, Lukasz Dutka and Marek Wieczorek for their contribution, and to Piotr Nowakowski for his remarks.

References

1. R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in meteor-s. In *IEEE International Conference on Services Computing (SCC'04)*, pages 23–30. IEEE Computer Society Press, 2004.
2. J. Blythe, E. Deelman, and Y. Gil. Automatically composed workflows for grid environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.
3. M. Bubak, T. Gubala, M. Kapalka, M. Malawski, and K. Rycerz. Workflow composer and service registry for grid applications. *Future Generation Computer Systems*, 21(1):79–86, 2005.
4. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From open grid services infrastructure to ws-resource framework: Refactoring & evolution, 2004.
5. E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in griphyn. In J. Nabrzyski et al., editor, *Grid Resource Management: State of the Art and Future Trends*, pages 99–116. Kluwer Publishing, 2003.
6. L. Dutka, P. Nowakowski, and J. Kitowski. Java based component expert framework for automatic application builder. In *4th Cracow Grid Workshop (CGW04) – Workshop Proceedings*. ACC Cyfronet AGH, 2005.
7. I. Altintas et al. Kepler: An extensible system for design and execution of scientific workflows. In *16th Intl. Conf. on Scientific and Statistical Database Management*. IEEE Computer Society, 2004.
8. M. Bubak et al. Automatic flow building for component grid applications. In *5th Int. Conf. on Parallel Processing and Applied Mathematics*, volume 3019. Lecture Notes in Computer Science, Springer-Verlag, 2004.
9. T. Oinn etal. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
10. D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce: Research and Applications*, 1(2):113–137, 2002.
11. R. Fileto, L. Liu, C. Pu, E.D. Assad, and C.B. Medeiros. Poesia: An ontological workflow approach for composing web services in agriculture. *The VLDB Journal – The International Journal on Very Large Data Bases*, 12(4):352–367, 2003.
12. D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski. Grid web services and application factories. In F. Berman, G. Fox, and A.J.G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley & Sons Ltd, 2003.
13. K. Krawczyk, R. Slota, M. Majewska, B. Kryza, and J. Kitowski. Grid organization memory for knowledge management for grid environment. In *4th Cracow Grid Workshop (CGW04) – Workshop Proceedings*. ACC Cyfronet AGH, 2005.
14. S. Majithia, D.W. Walker, and W.A. Gray. Automated web service composition using semantic web technologies. In J. Kephart, M. Parashar, V. Sunderam, and R. Das, editors, *International Conference on Autonomic Computing (ICAC'04)*, pages 306–307. IEEE Computer Society Press, 2004.
15. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services, 2004.
16. S. Narayanan and S.A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the eleventh international conference on World Wide Web*, pages 77–88. ACM Press, 2002.
17. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.

Towards Checkpointing Grid Architecture

Gracjan Jankowski, Jozsef Kovacs, Norbert Meyer,
Radoslaw Januszewski, and Rafal Mikolajczak

[1] Poznan Supercomputing and Networking Center,
61-704 Poznan, Noskowskiego 12/15, Poland
{gracjan, meyer, radekj, Rafal.Mikolajczak}@man.poznan.pl
[2] Computer and Automation Research Institute, Hungarian Academy of Sciences,
1111 Budapest Kende u. 13-17, Hungary
smith@sztaki.hu

Abstract. Contemporary Grid environments are featured by an increasingly growing virtualization and distribution of resources. Such situations impose greater demands on load-balancing and fault-tolerant capabilities. The checkpoint-restart mechanism seems to be the most intuitive tool that can fulfill the specific requirements. One of the goals of the CoreGRID Network of Excellence is to define the high-level checkpoint-restart Grid Service and to locate it among other Grid Services. We aim to define both the abstract model of that service and the lower layer interface that will allow the service to cooperate with the diverse existing and future checkpoint-restart tools. The paper is the first step leading to achieving this goal. It includes the overall sketch of the architecture of the considered service and its connection with the actual checkpoint-restart tools. Additionally, the work on low-level checkpoint restart tools to be used in the "proof of concept" implementation and integration is mentioned.

1 Introduction

Until now there have been few *checkpointing systems* that can do computing processes' checkpoints, for instance: psncLibCkpt[1], Altix C/R[2], Condor[3], libCkpt[4] and others. Additionally, the checkpointing functionality is shipped together with IRIX and UNICOS operating systems. These *checkpointing systems* always have different capabilities and interfaces and are specifically linked to a particular OS and hardware platform. Mainly, for the historical and technical reasons, the development of *checkpointing systems* is inherently difficult. Also in case of distributed computing the very strong semantics problems appear. Hence the additional challenge is to checkpoint distributed applications based on the PVM/MPI communication models[5]. Therefore *checkpointing systems* are not widely used and the existing ones always have some limitations which vary for different systems.

One can try to employ the aforementioned *checkpointing systems* in the Grid environment. Unfortunately, contrary to the Grid properties[1] expressed by

[1] Here we mean mostly the following: transparent and reliable, open to wide user and provider communities, pervasive and ubiquitous, easy to use and program, persistent, scalable, easy to configure and manage.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 659–666, 2006.

experts within NGG[6] and NGG2[7] such integration would impose high complexity. Then, if we want to use checkpointing functionality in Grids, we have to figure out an abstract *Checkpoint Restart Grid Service (CRGS)* that hides all the complexity and underlying *checkpointing systems*. Moreover, that service has to fit into the more general architecture which will allow bringing into play the diverse existing and future *checkpointing systems*. The idea of such *CRGS* and associated *Grid Checkpointing Architecture (GCA)* is presented in this paper.

We strongly believe that the emergence of such well-defined architecture and associated services will contribute to the popularization of the existing *checkpointing systems*. Moreover, it should prompt research and industry community to work out *checkpointing systems* for new platforms and to extend the functionality of the existing ones. So, thanks to the *GCA* the lack of the wide use of *checkpointing systems* can be overcome.

Please note that we are not concerned with the efficiency and effectiveness issues at the moment. We simply claim that checkpointing can be found very helpful and rational but we do not force anyone to employ checkpointing in every use case.

2 Grid Checkpointing Architecture

2.1 Architecture Outline

The architecture we are going to define comprises four layers. These layers are depicted in Fig. 1. Individual layers are separated by dotted lines. The top layer is composed of two parts. The first is *Other Grid Services* that can be any *Grid Service* which wishes to use the checkpointing functionality exposed by the *Checkpoint Restart Grid Service (CRGS)*. The part, as such, is not defined by the *Grid Checkpointing Architecture (GCA)*. The second part of the top layer is the End User Applications part which also, as such, is not defined by GCA. As it is shown in Fig. 1, *End User Applications* can take advantage of *CRGS*

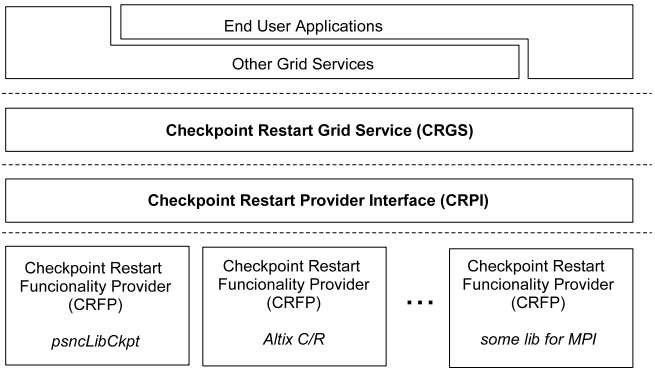


Fig. 1. Grid Checkpointing Architecture (GCA)

directly or through the *Other Grid Services*. If any part of the top layer wants to use the *CRGS* directly, it has to know the interface of this service.

The *Checkpoint Restart Grid Service (CRGS)* layer exposes the abstract checkpointing functionality to the whole world. The interface of this layer is defined by *GCA*. It is described in more depth in section 2.2. The *Checkpoint Restart Provider Interface (CRPI)* is designed for hiding the actual *checkpointing systems* under the abstract interface of the upper layer. The interface of *CRPI* is defined by *GCA*. This interface allows the incorporating of any actual *checkpointing system* to the set of systems that can be accessed through the *CRGS*. The last layer of *GCA* is made of the actual checkpointing systems that are collectively called as *Checkpoint Restart Functionality Providers (CRFP)*. The first emphCRFPs that will be employed in *GCA* are presented in section 4.2.

2.2 Checkpoint Restart Grid Service

Even though the *GCA* can contain many *CRFP* (i.e. *actual checkpointing systems*), all are exposed by the unified interface of the *Checkpoint Restart Grid Service (CRGS)*. Since the number of distinct *CRFPs* with different semantics and interfaces can potentially be infinite, the *CRGS's* interface has to be flexible and general enough to encompass all these cases. Furthermore, the service has to provide access to ontology-based knowledge about the underlying *CRFPs* and nodes that host them. The knowledge should also be available through dedicated, external information services. Therefore the *CRGS* is obliged to cooperate with these information services. The *CRGS* should be able to report some checkpointing-related events and information associated with them. For instance, one can be interested in the notification of each checkpoint that has been done and also in the sizes of the images that contain these checkpoints.

The *CRGS* has to reveal the *do_checkpoint()* function which, if it is only possible with the given *CRFP*, does checkpoint of the involved job. If the given *CRFP* does checkpoints independently of any external requests, then the appropriate information of that behavior has to be available. Regarding the recovering functionality, the *CRGS* offers two approaches. Depending on the ontology of the underlying *CRFPs*, the *CRGS* can provide information to external services on how to recover any given job, or can do it on its own in the *do_recover()* function.

It is also important to explicitly state what *CRGS* is not responsible for. The service does not execute the jobs that are to be checkpointable and does not manage images of checkpointed jobs. Thus, if any job has to be executed with some special argument to activate the checkpointing mechanism, the module or service that is responsible for the jobs executing has to obtain information about the required arguments from the *CRGS*. Similarly, if any service or module wants to replicate or archive checkpointing images, it cannot ask the *CRGS*. The *CRGS* can only give the information on how to obtain the recent image or images.

2.3 Checkpoint Restart Provider Interface

It is obvious that to achieve a general and unified interface for all different *CRFPs*, some kind of mediator or translator is required. Then this role can

be given to the *Checkpoint Restart Provider Interface (CRPI)*. Providing that we deal with a modern object-oriented programming language, the *CRPI* can provide a set of abstract classes or interfaces that are associated with the appropriate abstract functions of the *CRGS*. To incorporate a new *CRFP* the new classes that inherit from suitable abstract classes or implement adequate interfaces have to be created. The way the classes are implemented is not defined and strictly depends on the properties of any considered *CRPI*.

2.4 Checkpoint Restart Funcionality Provider

The actual *checkpointing systems* used within the *GCA* are called *Checkpoint Restart Functionality Providers (CRFP)*. The assumption is that it can be any system regardless of how it was implemented and what platforms and functionality it supports[8]. They can vary from some kernel-level *checkpointing systems* to some highly specialized, application- or user-level solutions for MPI or PVM applications. In fact the *CRFPs* do not have to be aware of the *CRGS* existing. They can be completely independent products that are incorporated to the *GCA* by means of *CRPI* (i.e. the upper layer). Three *checkpointing systems* that are to be employed as *CRFPs* in the "proof of concept" implementation of *GCA* are presented in section 4.2.

3 Surrounding Environment

Modern Grid environments are composed of at least a few interacting services. Individual parts of such systems become meaningful only in the context of other ones. Distinct services often offer some functionality and simultaneously take advantage of the functionality of other services. The semantics of *CRGS* is similar.

The envisioned relationships between *CRGS* and the external *Grid Services* within the *GCA* are depicted in Fig.2. The circles denote the external services. The *CRGS* is represented by the suitably labeled rectangle and the arrows represent interactions between those services (or other involved elements). As the figure expresses only a general idea, most of the arrows are not labeled here. According to the figure, *CRGS* will cooperate with *Grid Services* of the following functionality: *Storage and Transfer*, *Information Service*, *Event Handling* and some kind of *Scheduler or Broker* which will further employ the *Execution Module*. Moreover, the *Authorization and Authentication* service will be utilized by the *Storage and Transfer* and *Scheduler / Broker* services. What is important, the figure implies that external *Grid Services* are aware of the cooperation with *GRGS* and have to adhere to the interfaces and rules defined in advance.

In short, the aforementioned elements work together as follows: the *CRGS* installed on the *Compute Resource* registers the ontology of all available and supported underlying *checkpointing systems* with the *Information Service*. When the *Scheduler* wants to execute the job that is to be checkpointable, it has to find out suitable *Compute Resource* with appropriate *CRGS*. To do this, the

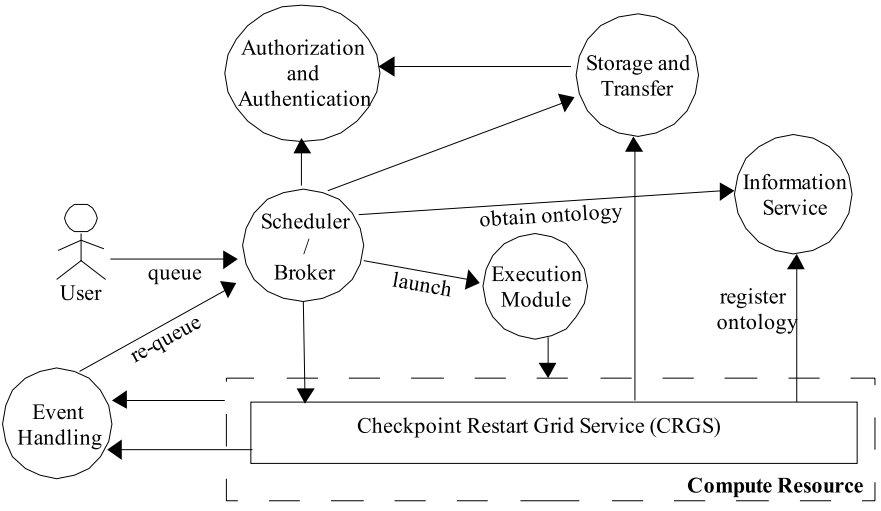


Fig. 2. Relations between WPs

Scheduler asks the *Information Service* about *Compute Resources* that fulfill the required capabilities (i.e. about *Compute Resources* with *CRGS* and appropriate *CRFP*). Further the *Scheduler* chooses a proper *Compute Resource* and launch the job on it. Potential knowledge of how to make the job checkpointable is acquired from the *Information Service*. Depending on the particular *CRFP* the checkpoints can be triggered by the job itself or by the *Scheduler* which can issue appropriate commands to the *CRGS*. After checkpoint is done the *Event Handling* service should be notified about it. Thanks to that, this event can be further delegated to the *Scheduler* which using the *Storage and Transfer* service, can do some replicas and do some bookkeeping activities.

We take an assumption that possible failures are detected by the *Event Handling* service, which sends to the *Scheduler* the re-queue request in such case. After receiving that request, the *Scheduler* has to find a new (or use the same) *Computing Resource* and recover the job. The *Storage and Transfer* service can be required in order to deliver checkpoint images into an appropriate place. The *Information Service* is used to find the new *Computing Resource*, localize the required checkpointing images and to provide knowledge of how to recover a job (e.g. what additional parameters are required to make the job being recovered and not executed from the beginning).

Fig.2 shows the *Authorization and Authentication* service. It was not explicitly mentioned earlier, but that service is used to manage access to the checkpoint images and to determine if the given subject has adequate rights to the given checkpoint image.

4 Proof of Concept Implementation

4.1 Grid Service

The *Global Grid Forum (GGF)* community has worked out the *Open Grid Service Architecture (OGSA)* specification. Nowadays it seems to be the most acceptable and usable specification for the architecture of the Grid environment. The *OGSA* specification is recommended by the EU's experts in NGG[6] and NGG2[7] papers. The *OGSA* is based on other well-known, accepted and managed standards like *Web Services*, *WSDL*, *SOAP* and others. As the *OGSA* provides the general rules and vision according to *Grid Services*, the *Open Grid Service Infrastructure (OGSI)* defines the actual interfaces, behaviors and schema for these services. The *OGSI* uses the *WSDL* to define the environment adhering to the *OGSA*. There are a few products that implement the *OGSI* and provide a framework for building Grid systems. One such product is the open source and Java-based Globus Toolkit.

Since we plan that our "proof of concept" implementation will be compliant with the EU experts' recommendations, we want to utilize the Globus Toolkit. Thanks to that our *CRGS* will be based on the *OGSA* specification and will be ready to cooperate with the most up-to-date Grid systems.

4.2 Integration with CRFP

One of the main features of *GCA* is its open architecture. The *GCA* has to be able to work together with all types of the existing, future and potential *checkpointing systems* (i.e. with *CRFPs*). Since the *CRGS* makes no sense without at least one *CRFP*, the "proof of concept" implementation of *CRGS* will cooperate with some *CRFPs*. The first one will be the *checkpointing system* designed for CONDDOR-PVM applications managed by the P-GRADE environment. The second one will be a general checkpointing tool, named TotalCheckpoint supporting sequential and parallel applications running under Linux 2.4 and 2.6. The third one will be the *checkpointing system* for applications running on IA64 processors under the Linux 2.6 operating system. This *checkpointing system* will be developed as part of our effort in the CoreGRID project.

P-GRADE. The Parallel Grid Run-time and Application Development Environment [10][11] developed by MTA SZTAKI is a graphical tool to create applications using the message-passing paradigm. In its editor the application topology and the communication-related parts of the processes are graphically expressed by icons, while the rest is the normal C code. The attached compiler produces a parallel application based on PVM or MPI. The novelty in its integrated *checkpointing system* is that all checkpoint-related support is built in the application itself and no co-operation of the surrounding modules of the execution environment is necessary. While the application is running, the termination of any child process causes the built-in checkpoint module to activate, and checkpoint files are produced before termination. With this feature the P-GRADE parallel applications behave like a sequential one, since simple jobs

with a built-in checkpointing library can also be saved just by delivering the appropriate signal.

The first prototype is currently available for PVM applications scheduled by the Condor job manager, where Condor is unaffected, not even aware of the application being checkpointed. The application might be monitored using Mercury during migration, too. This solution best fits for Grid environments containing heterogeneous job managers or in cases the checkpointing support is a missing part.

TotalCheckpoint. The well-seen disadvantage of the previous solution is that only parallel applications developed by P-GRADE can be checkpointed. MTA SZTAKI has moved forward to develop a general checkpointing environment called TotalCheckpoint. It operates on a single sequential job as well as parallel applications. It is based on a single process checkpointer, named Ckpt, developed at the University of Wisconsin.

TotalCheckpoint requires setting up a coordination process (one per cluster or site) running as a daemon in the background and relinking of the user application. Files are also saved, and as a result of a checkpointing action, an application description XML file is also produced to let the upper layers know about the internals of the application, for example, the list of working files. Through this description the renaming of the working files for a migrating application and the site migration is also supported.

The TotalChekpoint tool is prototyped and introduced to support the Hungarian nationwide ClusterGrid infrastructure maintained by the National Information Infrastructure Development Office. ClusterGrid is built by PC clusters provided by academic institutes and universities.

IA64-Linux Checkpoint Restart Package. The Linux operating system is increasingly gaining popularity in production-grade systems. Simultaneously, higher interest in IA64 CPUs is noticeable. Therefore, the development of the IA64-Linux Checkpoint Restart Package (IA64-LCRP), the new kernel level *checkpointing system* for Linux 2.6 OS running on the IA64 platform makes good sense. The functionality of that *checkpointing system* will be similar to that of Altix C/R [2], but IA64-LCRP will be aimed at a newer kernel version and will be more portable between different Linux distributions. Additionally, the IA64-LCRP will support checkpointing of multi-threaded applications. The resources virtualization mechanism described in [9] will also be included.

5 Conclusion

The *Grid Checkpointing Architecture* and its layers have been presented in a very general way. The forthcoming work includes more in-depth studies on checkpointing ontology, and a precise definition of the interface of the *Checkpoint Restart Grid Service*. Next, on the road to the OGSI compliant "proof of concept" implementation of the checkpointing service, the interface will be written in the WSDL language and finally implemented by means of the Globus Toolkit. At

the same time, the *Checkpoint Restart Provider Interface* layer has to be defined in a more formal way. Additionally, we believe that thanks to the open architecture of the presented solution, and due to the widely accepted and recommended technologies (OGSA, OGSI) we are going to use, the *Checkpoint Restart Grid Service* can become a very useful and widespread tool.

Acknowledgments

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

References

1. http://checkpointing.psnc.pl/Progress/psncLibCkpt/
2. Checkpoint/Restart mechanism for multiprocess applications implemented under SGIGrid Project, Gracjan Jankowski, Rafa Mikolajczak, Radoslaw Januszewski, CGW2004.
3. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System, Michael Litzkow, Todd Tannenbaun, Jim Basney, and Miron Livny; Computer Sciences Department University of Wisconsin-Madison.
4. Libckpt: Transparent Checkpointing under Unix', Conference Proceedings, Usenix Winter 1995 Technical Conference, New Orleans, LA, January, 1995.
5. Jozsef Kovacs, Peter Kacsuk: "A migration framework for executing parallel programs in the Grid", 2nd European AxGrids Conference, Nicosia, Cyprus, Jan. 28-30, 2004, pp. 80-89.
6. Next Generation Grid(s), European Grid Research 2005-2010, Expert Group Report, 16th June 2003.
7. Next Generation Grids 2, Requirements and Options for European Grids Research 2005-2010 and Beyond, Expert Group Report, July 2004.
8. A Survey of Checkpointing/Restart Implementations, Eric Roman, Lawrence Berkley National Laboratory, CA.
9. G. Jankowski, R. Mikolajczak, R. Januszewski, N. Meyer, M. Stroinski.: Resources Virtualization in Fault-Tolerance and Migration Issues, ICSS 2004, LNCS 3036, pp. 449-452, 2004.
10. P. Kacsuk, G. Dozsa, J. Kovacs, et all: "P-GRADE: a Grid Programming Environment", Journal of Grid Computing Vol. 1. No. 2, pp. 171-197. 2004.
11. PGRADE Parallel Grid Run-time and Application Development Environment: http://www.lpds.sztaki.hu/pgrade

Enabling Remote Method Invocations in Peer-to-Peer Environments: RMIX over JXTA

Pawel Jurczyk[1], Maciej Golenia[1], Maciej Malawski[1], Dawid Kurzyniec[2], Marian Bubak[1,3], and Vaidy S. Sunderam[2]

[1] Institute of Computer Science, AGH, Mickiewicza 30, 30-059 Kraków, Poland
[2] Emory University, Atlanta, USA
[3] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland

Abstract. In this paper, we present a peer-to-peer (P2P) system with remote method invocations, combining RMIX and JXTA technologies, and underpinning the H2O distributed resource sharing platform. We show that the integration of RMIX and JXTA was possible due to extensibility of the former, which allowed to plug in the JXTA-based socket implementations. The result of this integration is a fully operational RMI implementation running on top of the JXTA P2P network, where methods can be invoked on remote objects located behind firewalls or NATs. We present results of tests showing that our implementation can be used to connect peers in different LANs that cannot interact directly, while in the case of direct connection the performance is comparable to that of RMI using standard sockets.

1 Introduction

Scientists and companies are looking for new ways of accessing the computational power needed for solving large-scale computing problems. This issue is addressed by Grid systems development which provides middleware for running applications on distributed resources shared between institutions. On the other hand, huge potential computing power is located inside millions of computers connected to the Internet. These resources can be used in a Peer-to-Peer (P2P) fashion, as was demonstrated by the SETI@home [1] project. The common feature of Grid and P2P systems is the resource sharing and distributed nature of the environment. However, large administrative effort is required to set up the Grid infrastructure and to establish virtual organization security. On the other hand, P2P systems are more suitable for ad-hoc collaborations, characterized by more dynamic participation patterns than those observed in Grid systems.

Avoiding the administrative burden related to using Grid systems was one of the goals of the H2O [2] resource sharing platform. H2O proposes and implements the model, where the roles of resource providers, service deployers and users can be separated. In this model, providers can independently offer the CPU power of their machines by running H2O kernels, but the responsibility of service (application) deployment is delegated to others. This makes resource sharing easier for providers, in the spirit of the P2P model. Communication in H2O is

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 667–674, 2006.

facilitated by RMIX, which is an extensible RMI-based framework offering the RMI programming model to H2O-based distributed applications.

In order to enhance the H2O with the ability to operate within a P2P environment, we are integrating H2O with JXTA P2P system. The goal of this work is to exploit JXTA mechanisms in order to enable resource sharing among peers which may be hidden behind NAT or firewalls, and which may dynamically join and leave the P2P network, possibly changing their locations. Our approach decomposes into two separate tasks: (1) enabling communication in the P2P environment and (2) enabling resource discovery in the P2P network. In our previous paper [3], we presented the basic concepts of our solution. In this paper, we describe in detail the RMIX-JXTA integration, which brings the RMI programming model to P2P systems and provides foundations for peer-to-peer resource sharing via the H2O framework.

This paper is organized as follows: First, we give the background on RMI as a programming model for distributed computing and RMIX as its specific instance. Next, we overview P2P technologies and JXTA as the proposed standard. Subsequently, we describe our approach of integrating RMIX with JXTA and we present results of preliminary tests.

2 RMI as a Successful Programming Model and RMIX

The diversity of distributed programming approaches results in multiplicity and heterogeneity of distributed computing infrastructures. Remote Procedure Calls paradigm, exemplified e.g. by ONC-RPC and XML-RPC [4], gained popularity as it is based on common and well-understood function invocation semantics. Similarly, recent systems such as CORBA [5] or Java RMI [6] extend the notion of object-oriented programming to distributed environments via the concept of *distributed objects* whose methods can be invoked from remote locations.

Such Remote Method Invocation (RMI) model is natural and convenient in object-oriented languages such as Java. However, interoperability between different environments becomes an issue. For example, standard Java RMI implementation is based on the Java Remote Method Protocol (JRMP) that is sophisticated and full-featured, but limited in practice to pure Java systems. There are some other Java RMI implementations based on other protocols (e.g. RMI-IIOP [7] that enable connectivity with CORBA or JAX-RPC [8] using SOAP/HTTP for Web services connectivity) but usually these solutions sacrifice functionality. In these circumstances the RMIX project [9] has been initiated.

The main objective of the RMIX communication library is to enable using multiple, independent RMI protocol service providers within a single, RMI paradigm-based framework. RMIX is flexible and extensible, allowing integration of existing RMI implementations. Service provider implementations are pluggable at run-time and hot-swappable. Additionally, RMIX features several general-purpose enhancements over Java RMI, including dynamic stubs, SSL support, runtime binding and customizable virtual endpoints that allow the same remote object to be accessed via different protocols such as SOAP, JRMP,

SunRPC. The framework opens up a possibility of dynamic access control - the interceptor allows or denies method invocations depending on custom-defined logic. RMIX provides a set of new features in the method invocation model providing an asynchronous calls and a one-way calls [10], with precise semantics.

Importantly from the P2P environment point of view, RMIX communication library enables pluggability also at the byte transport level, via potentially user-supplied socket factories. RMIX socket factory model is more generic than the one defined by standard Java RMI, naturally supporting non-IP networks. This makes it possible to plug in a transport which uses non-host-port based addressing scheme, such as the JXTA P2P overlay.

3 P2P Networks and JXTA P2P Network Implementation

One of the first incarnations of the P2P paradigm was the Napster application [11], used for file sharing purposes. Next generations of P2P networks tried to avoid centralized servers, and they provided mechanisms for distributed lookup (Gnutella [12]), also introducing the concept of peer hubs for more efficient operation (Morpheus [13]).

The idea of exchanging files in a Peer-to-Peer manner inspired people to share free CPU cycles and resources, since it has been realized that there are millions of mostly idle machines that could be linked and used as one big supercomputer. Example projects which tried to tap on that resources are SETI@home [1], GPU Project [14], JNGI [15] and Parabon [16].

There exist many libraries allowing to create custom Peer-to-Peer systems but most of them are platform- or protocol-dependent. They are tailored for specific types of networks and applications, such as e.g. file sharing, and thus lack the necessary flexibility to serve as a general purpose sharing middleware. That is why JXTA [17] has been introduced. JXTA is designed to be a set of open, language independent protocols that allows any connectible devices, from cell phones, PDAs, and notebooks to big servers, to communicate witch each other as peers [18]. Currently, implementations of JXTA protocols exist in PERL, C and, of course, Java.

A *peer* is a central JXTA abstraction. When a peer connects to the network, it publishes its interface (the *endpoint*) for communication with the outside world. Peers can collect information about the network, available resources, and services. They may also gather in *peer groups* to form controlled and focused collaborations. A peer may belong to multiple groups at any given time.

One of the most important features of JXTA are its communication capabilities. Peers can create communication channels called *pipes*, which are used in the JXTA network to exchange messages. These virtual channels do not require a direct connectivity between the participating peers. For instance, if peers are behind NATs or firewalls, they can still communicate with the help of special *routing* peers that propagate necessary messages. Pipe messages can carry any type of data, such as e.g. text messages, binary data, or even Java objects. In

the Java implementation of JXTA, pipes are unidirectional. Peer endpoints can include multiple network interfaces, and JXTA tries to use the best one available. For instance, when peers are in the same subnetwork, pipes typically use direct TCP connections which greatly increases efficiency.

JXTA pipes are very useful, but they represent a relatively low level of abstraction. However, in the recent versions of JXTA, the socket API very similar to that of standard Java sockets has been implemented on top of pipes.

4 Advantages of a P2P System for Distributed Computing

Distributed resource sharing systems like H2O can draw many advantages from using a P2P RMI framework. In particular, such approach allows to extend resource sharing to environments spanning networks with NATs and firewalls. This is achieved by using flat addressing type in P2P systems. Moreover, independence between object address and its host IP address opens up a possibility of location-independent resource references, supporting mobility and dynamic load balancing. These new features in P2P distributed application frameworks will yield new possibilities for building global computing systems:

- simplicity in deploying distributed applications (no need of specialized configurations of routers or firewalls),
- wider distributed application range (users from private networks can participate in any distributed application),
- clients can use resources independently from their location,
- enabling ad-hoc collaborations and virtual computing groups (using peer groups in P2P network).

Motivation for using RMIX and JXTA technologies stem from the flexibility and genericity of JXTA on the one hand, and from the RMIX features, facilitating the integration with the P2P environment, on the other hand. Furthermore, since the JXTA Socket API is similar to Java Socket API used at the RMIX transport level, integration of both systems can be achieved relatively easily, bringing the advantages presented above.

5 Integration of RMIX with JXTA

The integration of JXTA and RMIX is achieved by implementing client socket factory and server socket factory interfaces defined by the RMIX communication library. By using JXTA Sockets abstraction in the mentioned socket factories, RMIX framework will be communicating on to of the P2P network.

To denote RMIX endpoint address in the P2P environment we have decided to use the following addressing model:

```
<string endpoint address>[@<group>[(param-1;param-2;...;param-n)]]
```

In its simplest form, an address can be a simple string such as *rmixEndpoint*. Such addresses belong to JXTA *NetPeerGroup* and are accessible globally.

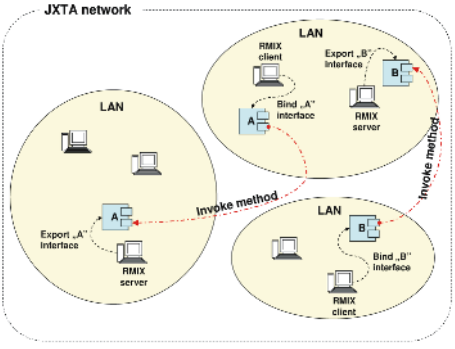


Fig. 1. RMIX communication library in P2P environment

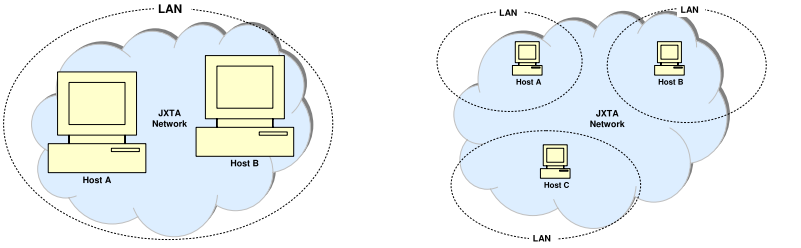
Alternatively, the address can be narrowed to a user-specified group, by providing two optional parts - the first defining a group name and the second providing additional parameters used when joining the group. This approach enables flexibility (e.g. custom security controls) at the group membership level, by supporting groups using custom implementations of Membership Service. For example, address *myEndpoint@cGroup* denotes an RMIX endpoint with address *myEndpoint* that is located in the *cGroup* JXTA group, with no security control. When one would like to use group with the security control, a valid address of RMIX endpoint may look like *myEndpoint@cGroup(groupInterfaceImpl;p1;p2)*. Here, when joining *cGroup*, the custom security control defined in *groupInterfaceImpl* implementation of our *JxtaGroupInterface* interface will be used.

From the RMIX user's point of view, the JXTA functionality is completely hidden. The user responsibility is limited to provide the JXTA socket address (usually embedded into a serialized stub received from a naming service or from another method invocation). The RMIX recognizes the endpoint as JXTA-specific and delegates transport-related activities to JXTA socket factories. The system automatically connects to the JXTA network (becoming a JXTA peer), joins a group (if needed), manages rendezvous status of the current JXTA peer, and connects to the JXTA socket specified via the provided address.

The approach to combining RMI and P2P paradigms, described in this section, creates a possibility of using the RMIX semantics in a P2P environment (as shown on Fig. 1). We argue that this approach creates new possibilities for developing distributed applications, by providing a simple remote method invocation model that can be used across any firewalls, NATs or private networks, in a location-independent fashion.

6 Experimental Evaluation

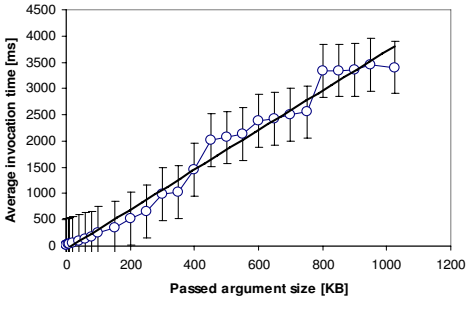
We have executed a simple benchmark to measure the performance of the RMIX communication framework in the P2P environment. We measure the round-trip



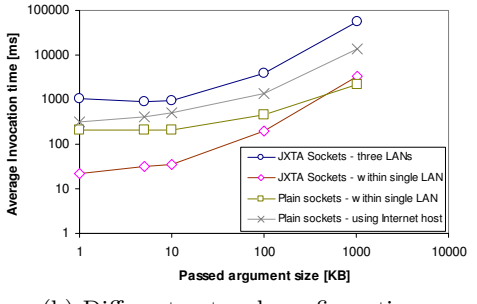
(a) Configuration that uses only a Local Area Network

(b) Configuration that uses the JXTA network spanning several Local Area Networks

Fig. 2. Test configurations of the JXTA Socket Factories



(a) Single Local Area Network



(b) Different network configurations

Fig. 3. Test results of RMIX using JXTA

time of a remote method call that accepts one argument of type `byte[]`, and returns a String object. The size of the byte array argument is a parameter of the benchmark. For each table size, we invoke the method 50 times and take the arithmetic average. The measurement was performed for two JXTA network configurations. In the first one (shown in Fig. 2(a)) JXTA network connects two computers from the same LAN. In this situation JXTA Sockets use direct TCP connections. The second test configuration 2(b) assumes that JXTA spans three LANs. Two of them contain the client and the server of our benchmark, while the third LAN contains a JXTA rendezvous peer providing connectivity between the client and the server. Additionally, to compare JXTA Socket with TCP socket performance, we run the same test using RMIX plain socket factories.

Results for JXTA Socket performance in a single LAN (the configuration shown in Fig. 2(a)) indicate linear increase of invocation time as a function of payload, which suggests that the communication is throughput-bound. Further experiments, illustrated in Fig. 2, show JXTA results for different setups,

compared with invocation times for plain RMIX socket factories in two configurations: the first one is for two hosts inside a single Local Area Network, the second involves a host within a LAN connecting to a server that is somewhere on the Internet. Note that the chart presented in Fig. 3(b) has logarithmic scales for better clarity.

As could be expected, when the payload is large, JXTA Sockets are consistently slower than plain sockets. The reason is that despite using direct socket communication, there is still overhead induced by JXTA pipes and their messaging layer. Surprisingly, however, we observed that for smaller data sizes, JXTA Sockets performed faster than plain sockets. We attribute this phenomenon to the connectionless nature of of the HTTP 1.0 protocol used by the Apache AXIS (SOAP 1.1 implementation used by RMIX), incurring a TCP handshake overhead on each method call. In contrast, JXTA pipes internally use TCP connection pools, reducing the connection establishment overhead. This trend is reversed when we use a connection- oriented RMI protocol, such as JRMP/TCP. Nonetheless, we point it out as an interesting observation that performance of widely used communication technologies and implementations (such as the Apache AXIS) can sometimes be improved by *inserting* an additional layer into the invocation protocol stack.

On the other hand, we observed a significant drop of JXTA communication performance a "three-LAN" communication scenario. The causes of this overhead are twofold: 1) routing through an intermediary, and 2) necessity to perform additional HTTP 1.0 tunneling to traverse firewalls.

7 Summary

In this paper we discussed the role of a P2P-enabled RMIX communication library in the context of distributed resource sharing, and the H2O platform in particular. The need for a generic platform allowing to run arbitrary code on resources shared via a P2P network can be satisfied by the combination of H2O and JXTA technologies. Since RMIX is an underlying communication substrate of H2O, exploiting JXTA communication capabilities in H2O reduces to integrating them with RMIX. We have shown that this integration is possible because of the RMIX extensibility which makes it possible to use JXTA sockets as the transport mechanism. The result of this integration is the fully operational RMI implementation running over JXTA P2P network, where methods can be invoked on remote objects located behind firewalls or NATs, which is not possible in traditional RMI systems. Preliminary test results show that our implementation can be indeed used to connect peers in different LANs that cannot interact directly, while maintaining performance at near-native level when direct connectivity is achievable.

The proposed RMI over JXTA implementation provides a foundation to enable H2O resource sharing in a P2P environment. In particular, it exploits the P2P *connectivity* and allows H2O services to be accessible via JXTA. Our current work is focused on the complementary *discovery* aspects. By reusing

decentralized P2P lookup mechanisms, coupled with global addressing and universal connectivity, powerful general-purpose P2P-enabled distributed computing platform can be enabled.

Acknowledgments. This research is partly funded by the EU IST Projects CoreGRID, KWf-Grid and the Polish State Committee for Scientific Research SPUB-M grant. The authors are grateful to Piotr Nowakowski for his remarks.

References

1. SETI@home: Search for extraterrestrial intelligence at home (2003) http://setiathome.ssl.berkeley.edu/.
2. Kurzyniec, D., Wrzosek, T., Drzewiecki, D., Sunderam, V.: Towards self-organizing distributed computing frameworks: The H2O approach. Parallel Processing Letters **13**(2) (2003) 273–290
3. Jurczyk, P., Golenia, M., Malawski, M., Kurzyniec, D., Bubak, M., Sunderam, V.S.: A system for distributed computing based on H2O and JXTA. In: Cracow Grid Workshop, CGW'04, December 13–15, 2004, Kraków, Poland (2005) 257–268
4. Scripting News, Inc.: XML-RPC Home Page (2005) http://www.xmlrpc.com/.
5. Object Management Group, Inc.: CORBA (2005) http://www.corba.org/.
6. Sun Microsystems, I.: Java RMI (2005) http://java.sun.com/products/jdk/rmi/.
7. Sun Microsystems, Inc.: Java RMI over IIOP (2005) http://java.sun.com/products/rmi-iiop/.
8. Sun Microsystems, Inc.: Java API for XML-Based RPC (JAX-RPC) (2005) http://java.sun.com/xml/jaxrpc/.
9. Kurzyniec, D., Wrzosek, T., Sunderam, V., Slomiński, A.: RMIX: A multiprotocol RMI framework for Java. In: Proc. of the Intl. Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, IEEE Computer Society (2003) 140–146
10. Kurzyniec, D., Sunderam, V.S.: Semantic aspects of asynchronous RMI: The RMIX approach. In: Proc. of 6th Intl. Workshop on Java for Parallel and Distributed Computing, IPDPS 2004, Santa Fe, New Mexico, USA, IEEE Computer S. (2004)
11. Napster, LLC.: NAPSTER Music Service (2005) http://www.napster.com.
12. OSMB, LLC: Gnutella File Sharing Network (2001) http://www.gnutella.org.
13. StreamCast Networks, Inc.: Morpheus File Sharing System (2005) http://www.morpheus.com/.
14. GPU Team: GPU project (2005) http://gpu.sourceforge.net/.
15. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I.: Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. Lecture Notes in Computer Science **2536** (2002) 1–12
16. Parabon Computation, Inc.: Parabon distributed computing platform (2005) http://www.parabon.com/index.jsp.
17. CollabNet, Inc.: JXTA Home Page (2005) http://www.jxta.org/.
18. Sun Microsystems: JXTA v2.3.x: Java Programmers Guide. (2005) http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf.

Transparency in Object-Oriented Grid Database Systems

Krzysztof Kaczmarski[1], Piotr Habela[3], Hanna Kozankiewicz[2], Krzysztof Stencel[4], and Kazimierz Subieta[2,3]

[1] Warsaw Univesity of Technology
k.kaczmarski@mini.pw.edu.pl
[2] Institute of Computer Science, Polish Academy of Sciences
{hanka, subieta}@ipipan.waw.pl
[3] Polish-Japanese Institute of Information Technology
habela@pjwstk.edu.pl
[4] Warsaw University
stencel@mimuw.edu.pl

Abstract. The paper presents various transparency issues that have to be considered during development of object-oriented Grid applications based on virtual repositories. Higher-level transparencies, such as location, heterogeneity, fragmentation, replication, redundancy, indexing and service provider transparency assure new information processing culture greatly supporting the development, operation and maintenance of Grid database applications. The paper discusses some requirements for a virtual repository that is a kernel of a Grid database and a general architecture of such systems. The architecture is based on object-oriented updatable database views that serve, in particular, as adapters of local servers and integrators/mediators on the level of a global virtual repository. Finally, some issues of the development of Grid database applications are presented.

1 Introduction

In distributed businesses digital data is scattered across different file systems, content repositories, databases, legacy applications and web sites, making it difficult to discover, reuse and integrate into content enabled processes [2,8]. In situation when global client software may need to access thousands of distributed resources, their complexity may undermine feasibility of business software goals. In addition, conceptual simplification has direct impact on various vital business factors, such as the cost, time and manpower necessary for software manufacturing, the quality of service (reliability, stability, effectiveness, etc.), the quality and size of software source and documentation, the software maintenance/change cost and others [7].

The mentioned effective data usage problems might be greatly simplified by transparent integration of resources and hiding technical details. Higher forms of transparency assure new information processing culture, where the client of a

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 675–682, 2006.

system may concentrate his/her effort on just services and abstract from technical and business peculiarities of particular service providers and contracts with them. Transparency has various forms, in particular:

- hardware, operating system, communication/transport protocol, file system and database management system transparency – users have no need and no possibility to involve these features into their programs,
- data/service location and access transparency – users need not care for the geographical location of data and services,
- concurrency transparency – users can access resources simultaneously and need not know about the existence of other users,
- heterogeneity transparency – users do not see local data structures and local system implementation but operate on higher and common form of information,
- scaling transparency – servers, data and services may be added or removed without impact on the applications and the users,
- fragmentation transparency – users need not be aware that data is partitioned; the fragments are integrated automatically,
- replication transparency – users need not be aware that data is replicated; replicas may be transparently added or removed to improve the efficiency of processing,
- redundancy transparency – users are isolated from redundant data that may exist among all local systems,
- site or connection failure transparency – most users can still work after some of the nodes or communication links are broken,
- migration transparency – data and services can be moved without any impact on the applications and users,
- optimization transparency – indexing, query cashing and rewriting, pipelining, decomposition and other optimizations are done on the level invisible for users,
- service provider transparency – users are interested in just services, service providers are hidden or are shifted to a secondary scene.

Typical approaches to Grid technologies consider some limited forms of transparency, from the above list. The current tendency is, however, to achieve advanced transparency forms that will simplify and unify the access of the user of global applications to the entire resources of the organization. Full heterogeneity, fragmentation, replication, redundancy, migration, optimization and service providers transparencies are considered in the context of virtual data/service repositories rather than in classical grid-oriented literature. However, achieving all the advanced forms of transparency is a big challenge for developers. Usually existing systems do not solve it to satisfactory degree. Some of transparency forms, although conceivable to achieve, may result in compromising the performance of applications, thus tradeoffs between performance and transparency might be required.

Usually, systems integrating heterogeneous resources propose kind of a middleware solution that could be a basis for achieving many forms of transparency.

The main goal is to create a new layer of abstraction, which could seamlessly integrate any technology - data and service, and perform additional administrative operations in a way invisible for end-users. Fig. 1 presents the general approach to transparency that is already implemented in many systems (CORBA, RMI, WebServices, etc). It is usually realized by an architecture employing three or more layers.

Transparency achieved by virtual repositories has two organizational aspects. The first one could be called functional and focuses on generating virtual data reflecting real, concrete resources in a form acceptable for clients. This must be done according to certain business goals. The design is created by an involved consortium (called a Virtual Organization [3]). Complexity of this task may vary in different applications. Thus, business analysts, Grid designers and managers play a key role here. The second aspect may be called operational (or administrative) and it must assure proper operation of the virtual repository. This part of the system is independent of the chosen data structures and business goals. It is responsible for automatic performing repository reconfiguration, meta-database modification, indexing, caching and other operations in dynamically changing environment.

Both of these aspects accomplished by the system simultaneously may guarantee proper integration of distributed heterogeneous databases. If such a system offers transparency on a satisfactory level, we call it a Grid Database. In this paper, we discuss basic issues of Grid databases such as common (canonical) data model that is to be introduced on the Grid global level, the architecture and general design phases of a Grid database creation. These issues are currently investigated in our prototype implementation ODRA (Object Database for Rapid Application development), which is based on the Stack-Based Approach to object-oriented query/programming language, the language SBQL and updatable views defined in SBQL. ODRA is assumed to be a platform for enhanced Web Services and advanced Grid applications.

The rest of the paper is as follows. Section 2 discusses some issues of a virtual repository - a kernel of Grid database applications. Section 3 is devoted to the architecture, based on the concept of object-oriented updatable database views. Section 4 presents generalities of the creation of a Grid database. Section 5 concludes.

2 Virtual Repository in a Grid Database

A big advantage of virtual repositories is that data and services need not to be copied, replicated and maintained on the global applications side: they live on their autonomous sites and are locally supplied, stored, processed and maintained. In many businesses copying data (in any form) is not allowed due to local security policies. Virtuality of data, however, requires automatic mechanisms allowing one to access the data as easily as if they resided inside one machine. This property is extremely difficult to achieve if virtual data are to be updated and mappings from stored to virtual data are a bit more sophisticated.

Many virtual repositories are already implemented and operating. Usually, they are proprietary solutions; no general standard or conceptual frame is observed. The repositories differ in end-user or programmer interface, security capabilities, forms of transparency and other aspects.

A little step forward was done by Web Services [12] but their capabilities in the field of transparently distributed systems are limited, what was already observed by the OGSA standard [3]. In fact, they achieve only basic forms of transparency. Global applications based on distributed, heterogeneous and redundant data/service resources databases require technical features much beyond the current capabilities of Web Services.

2.1 Common Data Model – Functional Aspect

Transparency can also be considered as a higher abstraction level over distributed, heterogeneous and redundant data and services. From the functional point of view, the abstraction means a common canonical data model, which is universal, simple and minimal. From among many approaches (data models) that can be considered as candidates to build canonical models we have chosen the Stack-Based Approach (SBA) and a corresponding query language SBQL [8, 10]. SBA introduces (as the main notions): complex objects, classes, methods, inheritance, dynamic object roles, encapsulation and other concepts of the object-orientedness. Each object has an internal identifier, external name and may contain a value, a link or a set of objects. The major difference between SBA and other object-oriented database models is that it is based on a fully-fledged query/programming language with precisely defined formal operational semantics. A programmer writes queries and updating statements based on queries, which may manipulate objects in repositories: create new objects, remove, update and insert into other objects, call their methods, create and assign new roles, etc. Queries have all well-known capabilities plus some more advanced like transitive closure. One of the most important features of the SBA data model is virtual objects (updateable views), which from the programmer's point of view are undistinguishable from stored objects. Virtual objects do not exist in any real data storage but are determined by view definitions. A view definition establishes two kinds of mappings: (1) from stored objects into virtual ones; (2) from any operations on virtual objects (reading, updating, deleting, inserting, etc.) into operations on stored objects. This mechanism of updatable views makes it possible to achieve many transparency forms mentioned in the introduction, in particular, location, fragmentation, indexing, replication and redundancy transparency.

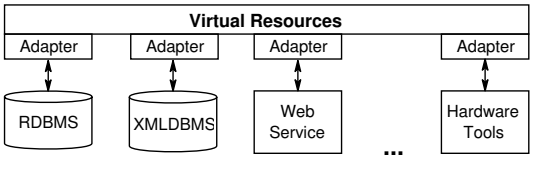


Fig. 1. General approach to transparency

Mappings of stored objects into virtual ones are also known from other approaches, for example, SQL views in relational databases. In all such solutions, however, the programmers must face the problem of reverse mappings if one would like to update information delivered by a virtual object [6, 11]. Some systems simply forbid such possibilities. Other, like Oracle and MS SQL Server, require so called INSTEAD_OF triggers. Our updatable object views define five generic operations, which must be defined for each virtual object kind in order to achieve full transparency between stored and virtual objects. These are: on_update, on_insert, on_delete, on_retrieve and on_create (on_create_pointer). They cover all generic operations that may be performed on objects. Explaining all subtleties of this data model is beyond the scope of this paper; please refer to [5, 6, 10] and other references to SBA and SBQL.

Similarly to normal objects, virtual ones are identified by virtual identifiers. Their construction allows the system to distinct them from normal identifiers and use dedicated methods, while a programmer is not aware of their virtual nature.

2.2 Operational Aspect of Transparency in Grids

Apart from establishing a common data model, which is used in a virtual repository, a Grid Database must also implement features assuring proper operation of the whole system. The general idea is to make such features as invisible as possible for the users. This allows the system to achieve a higher level of transparency, which is the main benefit for administrators, programmers and end users.

However, some solutions, even if very useful, may be very expensive while other may be too complex. A consortium planning to create a virtual repository based on a Grid Database should always carefully analyze possible advantages and drawbacks, finding a reasonable tradeoff between higher forms of transparency, satisfactory performance, programmers/user efficiency and the costs of development, operation and maintenance. Simplified, abstract, encapsulated and better organized data structures usually require additional operations performed by the system behind them. Many stages of data transformations on the way to a client may produce additional and sometimes unacceptable performance overhead. There is obviously the need for automatic optimizers [4], in particular, parallelization of computations [1], indexing techniques, query rewriting mechanisms and special physical data organizations. There may be also the need for special ,,golden rules" for programmers (c.f. Oracle SQL) encouraging or discouraging using some features of the system and its languages.

3 Grid Database Architecture

In our approach, a Grid Database consists of independent but cooperating database systems, which share data: publish and process objects. Clients use its virtual repository, which transparently integrates data and accomplishes a higher abstraction level. The repository is based on two kinds of specialized views: contributory views defined on local servers and global views defined on the global

level. Fig. 2 on the left presents possible (nested) architectures of a Grid database. Local servers (smallest rings) can be integrated into the first-level Grid (larger rings); then these Grids can be integrated into the global Grid (the biggest ring). Because virtual objects delivered by virtual repositories are not distinguishable from stored objects, the number of nesting levels of the Grid is (theoretically) unlimited. Each Grid node has the same conceptual architecture. If only desirable for the consortium, data integration process may also be divided into several stages. Fig. 2 on the right presents the architecture of a Virtual Repository Layer realized by stateless Global Views (GV) supported by a Grid Coordinator and a resources' Adapters Layer implemented by Contributory Views (CV).

A contributory view (sometimes called wrapper, mediator or adapter) is a view by means of which a node shares its own resources with the others. Its main task is to hide heterogeneity of local database systems (object, relational, etc.) within the consortium, by transforming local data models into unified data model specific for the consortium and to control access rights and hide data, which should not be published.

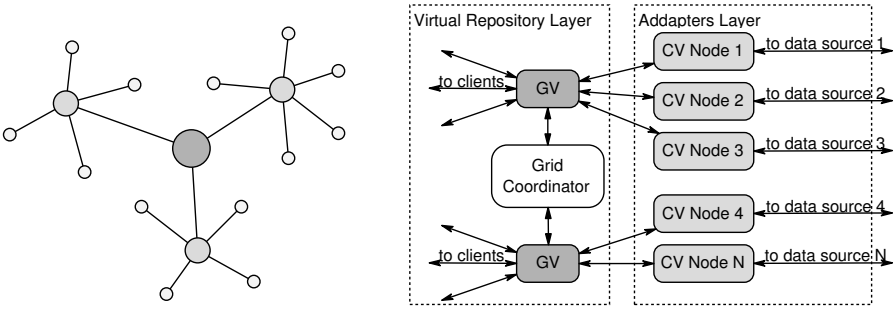


Fig. 2. Architecture of a Grid database

A Grid view (sometimes called mediator, integrator or fuse) is a view delivering virtual objects to users. Through this view, a user sees Grid resources adapted to his/her particular needs. All data transformations, optimizations and additional operations are hidden behind this view. Users see only resulting virtual objects created according to the established consortium's schema. Virtual objects created by the view do not exist in any concrete repository place, but are materialized on the fly, when needed upon distributed resources. Some optimizations (notably query modification [9]) cause that in fact some virtual objects may be never materialized.

A Grid coordinator plays additional role of a global coordination and global information administration. Generally, it is dedicated to: support system-wide optimizations (like query caching, indexing, query decomposition); control global access rights (which probably should not be decentralized due to security requirements) and meta-database coordination (sending additional update information to distributed Global Views). There are many ways to organize this global

coordination to be feasible and optimal for all users. It may be centralized, clustered or distributed according to the particular cost, time, complexity and other constraints.

4 The Process of a Grid Database Creation

The two mentioned aspects, functional and operational, reflect in the whole process of Grid database modeling and construction. The first one influences Updatable Object Views modeling and implementation, while the second one focuses on configuration of a Grid Coordinator. Especially the functional aspect is dependent on certain business strategy and cannot be accidentally solved. For this reason, we are skeptical about the feasibility of ad-hoc (or dynamic) integration solutions (except for perhaps very simple data structures) especially in the context of higher forms of transparency. Thus, for a serious project, a full development cycle is to be initiated by a consortium and precise rules obliging every participant are required. We assume the following Grid construction scenario:

1. Strategic phase, when the decision on creating a Grid is made and an initial analysis of the required content and potential participants are performed.
2. Analysis phase, when the existing resources are elaborated and confronted with the information requirements for the integrated service. The issues such as data heterogeneity, redundancy or incompleteness should be identified.
3. Design phase, which results in the precise definition of the global virtual schema and contributory schemes for each participant. The task of transforming local resources to the agreed contributory schema is delegated to an adapter of a particular node. On the other hand, the specification of mapping the contributions into the global schema needs to be specified as an integral part of the Grid design.
4. Finalization phase, when participants sign the final agreement formalizing the obligations coming from contributory schemes' specifications. Each participant gets unique identification and authorization codes.
5. Implementation phase, when the necessary data adaptation described by contributory schemas and global schema are implemented. Depending on the heterogeneity between a given node's data and global schema, such node may require either appropriate wrapper or may require in-depth restructuring of its original design.

The resulting specifications determine the required form of data provided both by the global virtual repository (the Grid itself), as well as by each of the participants. The task of adjusting local data into the form required by the Grid can be distributed between participants (that is, the administrators and developers of local systems) and the integrator (that is, the developer of the system serving global, integrated data view). Although different ways of balancing this responsibility are possible, we assume that in order to simplify the integration itself, local sources should be obliged to adapt their contribution as far as possible, based on the knowledge of their local resources.

5 Conclusions

Updatable views allow us to reconsider architecture and transparency in distributed databases. Fully operable virtual objects present new quality among many approaches to virtual repositories. Unlimited composition of Contributory Views and Global Views allows us to achieve full transparency of data location, heterogeneity, fragmentation, indexing, replication and redundancy. More forms of transparency are possible if the system is provided with additional global coordination. That means migration, failure and optimizations transparency. Independence of local databases and distributed query evaluation based on additional meta-information provides concurrency transparency and scaling transparency (in case of horizontal fragmentation).

The presented architecture is currently being implemented in an experimental system ODRA, an object-oriented database management system devoted to rapid development of web and Grid applications.

References

1. E. Bertino: Query Decomposition in an Object-Oriented Database System Distributed on a Local Area Network. RIDE-DOM 1995: 2-9
2. S. Ceri, G. Pelagatti: Distributed Databases: Principles and Systems, 1984
3. I. Foster, C. Kesselman.: The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 2003
4. D. Kossmann: The State of the art in distributed query processing. ACM Comput. Surv. 32(4): 422-469 (2000)
5. H. Kozankiewicz, J. Leszczyowski, K. Subieta.: Updateable XML Views. Proc. of Advances in Databases and Information Systems (ADBIS), Springer LNCS 2798, pp. 385-399, Dresden, Germany, 2003
6. M. Lenzerini.: Data integration: a theoretical perspective. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. Madison, Wisconsin. 2002
7. A.Y. Levy, A. Rajaraman and J. J. Ordille.: Querying Heterogeneous Information Sources Using Source Descriptions. Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India, 1996
8. M. Tamer zsu, Patrick Valduriez: Principles of Distributed Database Systems, Second Edition Prentice-Hall 1999
9. K. Subieta, J. Podzie.: Object Views and Query Modification. In: J. Barzdins, A. Caplinskas (eds.) Databases and Information Systems. Kluwer., pp. 3-14, 2001
10. K.Subieta. Theory and Construction of Object-Oriented Query Languages. Editors of the PJIIT, 2004, 522 pages, ISBN 83-89244-28-4 (in Polish)
11. I. Tatarinov et al.: The Piazza Peer Data Management Project. ACM SIGMOD Record, 32(3), 2003.
12. W3C Consortium: Web Services Documents [http://www.w3.org/2002/ws]

Unifying Grid Metadata Representations Through Ontologies

Bartosz Kryza[1], Marta Majewska[2], Renata Słota[2], and Jacek Kitowski[1,2]

[1] Academic Computer Center CYFRONET-AGH, Nawojki 11, Cracow, Poland
bkryza@icsr.agh.edu.pl
[2] Institute of Computer Science AGH, University of Science and Technology, Mickiewicza 30, Cracow, Poland
{mmajew, rena, kito}@agh.edu.pl

Abstract. Grid environments provide now variety of middleware solutions that are supposed to help manage resources available in the Grid environment. In this paper, the need for a more uniform way of defining metadata in the Grid is presented. A proposition for using ontologies as a formalism for unified representation of various kinds of metadata in the Grid is given. Also description of basic requirements that every Grid metadata approach should address and how these requirements can be fulfilled with ontologies is provided. Finally, a brief decription of example of such an approach taken in EU K-Wf Grid Project is described.

1 Introduction

As Grid computing gains more and more attention and grows mature, the need for unification and consequently standardization of basic technologies used throughout the Grid environments is becoming obvious. This was exactly the situation with the OGSA architecture, which brought a uniform way of accessing virtual resources accessible in the Grid by means of emerging Web Services technology, a couple of years ago. Since Grid technology has recently gained substantial attention within the computer science community, vast amount of various solutions in all aspects of Grid computing have been proposed. As of now, there is no standard way of managing distributed hardware resources, no standard way of monitoring the Grid infrastracture and no standard way of defining and accessing metadata, a crucial part of every Grid enviroment. Most current metadata solutions for the Grid have been developed independently of each other for particular purpose, thus leading to serious limitations in finding all necassary information in a uniform and standard way. This issue has been already raised in a few papers [1, 2].

Since metadata of any kind are basically sets of facts about some domain, it is clear that it is possible to find a unified way of describing them. One of the solutions for this problem lies in the use of ontologies. Ontology-enhanced software systems have gained throughout the last years at least as much attention as Grid computing and lots of research has been done in this field that could be reused.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 683–690, 2006.

The Grid and Semantic Web have been developed for the last years as independent technologies, so while trying to merge them a question arises: *Should Grid environment enhance the way knowledge is managed in some organization or rather should Grid environment benefit from the semantic description?* We believe that to some extent both of these statements are true, but in this paper we would like to focus mainly on the latter one. None of these technologies has fully matured yet, and so their too early integration may bring some difficulties.

The basic rationale behind the use of ontologies in the Grid, is to make Grid easier to use for administrators, application developers and end-users and to allow the possibility of seamless integration of future Grid deployments with each other. At the moment, several new research projects address these issues, like InteliGrid [3], OntoGrid [4] or K-Wf Grid [5] to name only few. In this paper we present the ideas and some early achievements of the K-Wf Grid project in this area.

The following sections of this paper provide brief overview of metadata solutions used within Grid systems, a rationale behind using ontologies for providing common metadata representation and finally an example of such approach taken in K-Wf Grid project.

2 Metadata in the Grid

Grid systems are by definition composed of large amount of geographicaly distributed and possibly replicated, heterogenous resources not subject to centralized control. Due to this fact the need for robust searching and choosing between them becomes a critical issue. This can be addressed properly only when sufficient methods for describing these resources exist within the Grid.

In every Grid application we can distinguish two disparate uses for metadata. First one is about describing common, real world concepts and those related with the Grid middleware itself, thus providing a general set of metadata scheme reusable for a particular Grid middleware, independently of the particular deployment of that Grid system. Another use for the metadata is allowing particular virtual organization users to extend the schema and thus let them describe entities that could have not been predicted during the Grid system development or even deployment. This separation is very general, as it cuts the space of possible concepts into the generic ones (that can be used by any application or the Grid middleware itself) and to domain specific ones.

Another possibility of separation is a strictly existential one, which means it is concerned with what these entities in fact are. Here the concept space could be divided into sets describing application related entities, resources composing the Grid, data managed by the Grid's data management subsystems as well as services registered and accessible from the Grid.

Currently available solutions are usually designed with only one of these dimensions in mind. For instance MDS [6] is mostly concerned with managing hardware and software resources throughtout the Grid while UDDI [7] provides service for registering and locating web services. If at some point, a need for

composing the facts stored in both of these registries arises, the client is forced to create two different queries using two different protocols and query languages. Thus it seems obvious that a need for a general way of describing all of the Grid metadata exists.

3 Ontology as Uniform Representation of Metadata

One of the possibilities to unify the metadata representation is the use of ontology. The ontologies seem particularly interesting due to the fact, that special tools called reasoners can even generate facts that were not introduced into the system directly, but are logical consequences of the asserted statements. The basic idea of unification model of metadata representation in the Grid is presented briefly in Fig. 1.

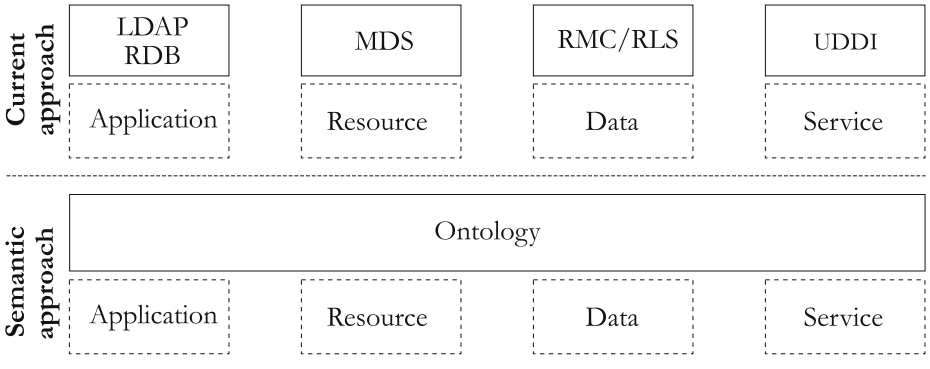


Fig. 1. Current metadata solution versus unified ontological approach

However, to ensure that semantic description is particularly useful for this purpose, the following requirements should be satisfied. First of all, some concepts of the ontology must be fixed (not subject to change) in order to allow Grid middleware developers to depend on it. It means that if some of these basic concepts would change, a modification in the middleware layer might be necessary, which could mean redeployment of many of the systems basic components. This requirement implies the neccessity of minimal possible ontological commitment of the ontology provided with the middleware itself, which means that the ontology should have only a minimal impact on how users understand the system (more users means bigger differences in understanding). These consequently bring the need for allowing the knowledge base extensions by the users of the Grid during its normal usage, maybe with respect to some security constraints, while assuring its consistency. Furthermore, an important issue is to allow automatic generation of some metadata, e.g. through regular logical reasoning but also by using some more refined AI learning algorithms. The metadata repositories themselves, must provide ways for creation and extensions of the schema

(classes of entities and possible relations between them) as well as of the registries (particular instances of these classes).

4 Semantic Approach to Metadata in K-Wf Grid Project

The issues mentioned in the section above are at the core of semantic research performed within the K-Wf Grid project [8]. The first and important choice faced when designing an ontology-based system, is the choice of ontology language. Many good solutions already exists, like WSML [9] and OWL [10]. In the K-Wf Grid project the decision was made to use the OWL, as it is already a W3C recommendation and there is a substantial amount of ontologies already defined that can be easily reused with OWL's import mechanism.

Another problem is the high level design of the ontology, which should enforce maintainability and extensibility. The main idea taken here, was to separate ontology into components organized into two dimensions.

The vertical dimension cuts the concept space into a set of generic ontologies, a set of domain specific ontologies and a set of individuals registries. These registries are responsible for managing the individuals originating from particular ontology. Since these individuals represent entities like users, hardware or files, it can be foreseen that their number will grow significantly throught the life of a particular Grid deployment, so the registries design must be able to cope with this requirement. The generic ontology components set forms a K-Wf Grid core ontology, the same for any application that K-Wf Grid will be used for (e.g. the same for Traffic Monitoring, Flood Crisis Management and ERP, which are the main pilot applications for the project). Although the ontology components mentioned here will be developed particularly for the K-Wf Grid, this separation scheme is general and could be introduced in any Grid middleware system. This

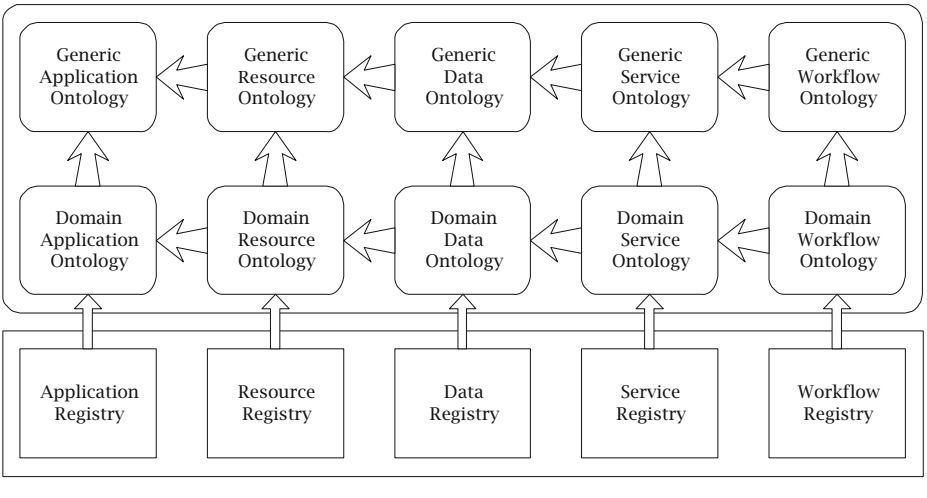


Fig. 2. Separation scheme of ontologies in K-Wf Grid project

ontology set can be considered as an intrinsic part of K-Wf Grid middleware. The domain specific one will be created specifically for the needs of particular application while allowing the users of that particular virtual organization to extend it during the utilization of the system. The Individuals Registry components will hold information about instances of metadata concepts organized according to a scheme set by generic and domain specific ontologies.

The horizontal dimension provides concept separation boundaries within the generic or domain specific ontology sets and registries. The main rationale behind this idea is to make the ontology maintainable as a whole by means of creating a logical dependency structure of otherwise very large concept space. While this separation might seem as a constraint at first, in our point of view it should make users understand better the structure of knowledge underlying the system and more importantly keep it reasonably maintainable.

An overview of the ontology separation scheme within the K-Wf Grid is presented in Fig. 2. The arrows between components represent the import closure.

The horizontal dimension contains the following categories of ontologies:

- *Application ontology* - contains concepts that are a basis for defining concepts in all other components. It can be considered as an ontology describing the environment to which particular Grid deployment is applied, that is in the generic application ontology we can declare for instance spatio-temporal concepts like *Time Period* or *Location*, while in the domain specific we should define the physical entities that relate to physical objects and processes specific to a particular domain and that indirectly influence the Grid higher level services. Examples of general ontologies that could be reused within this component are OWL-Time [11], SUMO [12] and DOLCE [13]. However such decision taken on the generic level which is fixed from the point of view of the middleware forces large ontological commitment.
- *Resource ontology* - this set of components contains concepts that relate to entities that can be directly controlled by the Grid or that generate events directly processed by the Grid. Thus this ontology contains description of both hardware and software resources within the computing domain, and all kinds of sensors or controllers within the particular application domain (e.g. SurveillanceCamera in traffic management application). The registry could store both the entire hardware and software configuration of the Grid, or just support existing solutions like MDS.
- *Data ontology* - this ontology is one of the most challenging since, on the generic level, it should allow integration of variety of different data management systems. The main idea here was to avoid a dominant decomposition of data (e.g. file in Unix operating system) in order to introduce the possibility of describing data independently of its format and storage mechanism. Thus, we have introduced a concept of DataObject which contains three basically independent aspects: Data - *What kind of data it contains?*, StorageObject. - *How it is stored and how can it be accessed?* and Format - *How is the data structured within the StorageObject?* The registry allows for storing all kinds of data source like files, databases, etc.

- *Service ontology* - these ontologies are concerned with a semantic description of Grid services. In most part it is based on the OWL-S standard [14], as up to now the most mature solution for this purpose (another similar is WSML). The generic component provides a hierarchy of Profiles that allows for functional classification of services (e.g. *KWFMiddlewareService*). Additionaly, it has an extended description of Input/Output concepts, in order to allow matchmaking of services into workflows for instance with respect to DataObject concept, to allow passing data virutally (e.g. by LFN). The registry will allow registration of service instances and thus provide the functionality of current solutions analogical with UDDI.
- *Workflow ontology* - the last ontology set is specified for describing workflows, an essential part of the whole K-Wf Grid project. On one hand, the workflows can be described at a general level, with concepts like *Execution Time*, *Author* and on the other hand their internal structure can be also defined semantically to make some more advanced reasoning possible, for instance measuring similarity between particular workflow instances. The registry will hold all past workflows to allow reusing them in the future.

The example presented in Fig. 3 shows a simplified ontology with generic concepts, domain specific (traffic monitoring) and some individuals within the registries. This simple scenario shows semantic description of all aspects of a Grid environment supporting basic use cases in the Grid. In this example, the workflow *Wf001* contains a service that requires on its input a video sequence from *ViaGaribaldi* street from some period of time in *MPEG* format. During the composition of the workflow, it turns out, that the video sequence with requested data exists, that is the data from surveillance camera *SC0004* located

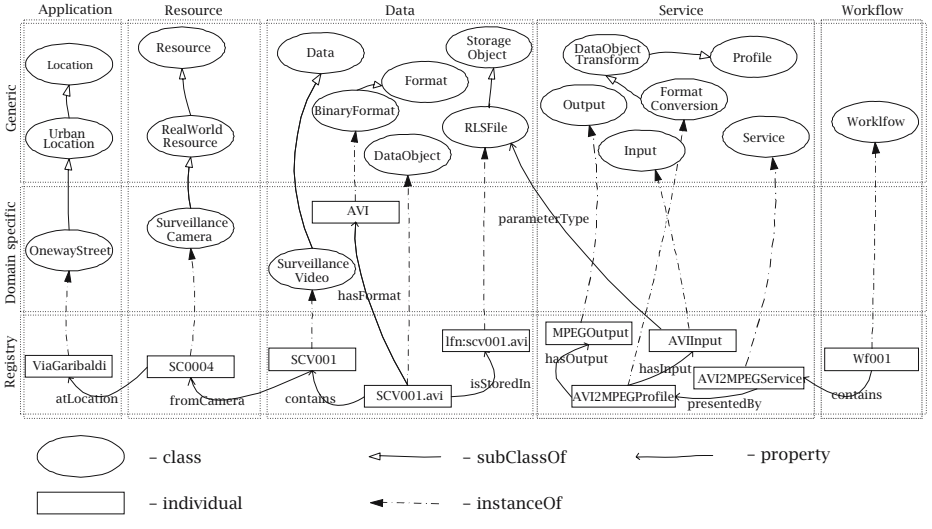


Fig. 3. Example of encouraging inter-domain definitions of concepts with unified metadata model

on the street in question, but it is in different format - *AVI*. Now the system could try to find a service that is be able to tranform some aspects of the existing DataObject into the needed one (here it is conversion of Format aspect from *AVI* to *MPEG* by service *AVI2MPEGService*). Thanks to the semantic description, this conversion can be included in the workflow without any user intervention.

In parallel with the ontology development process, within the framework of K-Wf Grid project, a special software component is being developed called Grid Organizational Memory [15]. This component will play the role of a universal knowledge base, able of storing and publishing the whole presented ontology scheme along with the registries.

5 Future Work

The main area of further research on this approach, will be concerned with excersising the concepts and relations contained in the generic ontology components with more real world scenarios, to ensure its extensibility for any purpose a Grid environment could be used. Also, integration at the level of generic ontologies of common Grid standards like OGSA-DAI [16] will be considered.

Another problem that needs more attention is that of security constrains. One approach could be based on introducing the security within the ontology itself, for instance by attaching some general property (e.g. *AccessRights*) with every concept in the ontology, however this may lead to some major incosistencies within the knowledge base. Different approach is that of restricting access to some users for reading or modifying on the basis of the whole ontology components, which could be effective enough when the level of separation between the components is high.

6 Conclusions

Grid middleware and applications that use it, can strongly benefit from semantic technology. The most natural application of this technology in the Grid is defining and storing of all kinds of metadata, thus making them uniform in terms of the representation language, but also in terms of protocols used to access it. The main problems, that seem to delay this solution, include:

- Still immature tools for manipulating ontologies and for building efficient and scalable repositories of ontological data.
- Until recently, little understanding of semantic technologies within the Grid community and vice-versa.

We believe, that with time, as both technologies become more mature, and their understanding increases, the benefits, like those shown in this article, will drive the work towards the standardization of such integrated Grid architectures.

Acknowledgments

This reasearch has been done in the framework of EU IST-2002-511385 K-Wf Grid project. The authors would like to thank the whole Project Consortium for remarks and feedback during the ontology development process. AGH University of Science and Technology grant is also acknowledged.

References

1. Quix, C. Quality-oriented and metadata-driven integration in information grids. *2nd IST Workshop on Metadata Management in Grid and P2P Systems (MMGPS) - Models, Services, Architectures*, London, UK, December (2004).
2. Foster, I. T., Jennings, N. R., and Kesselman, C. Brain meets brawn: Why grid and agents need each other. *3rd International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS*, New York City, USA, July (2004), vol. 1, pp. 8–15.
3. Inteligrid project homepage. http://www.inteligrid.com/.
4. Ontogrid project homepage. http://www.ontogrid.org.
5. K-Wf Grid project homepage. http://www.kwfgrid.net.
6. Globus Alliance, Globus Toolkit 2.2 MDS Technology Brief. On-line, http://www.globus.org/mds/mdstechnologybrief_draft4.pdf.
7. OASIS consortium, UDDI homepage. Online, http://www.uddi.org.
8. Dutka, L., Kryza, B., Krawczyk, K., Slota, R., Majewska, M., Hluchy, L., and Kitowski, J. Component-expert architecture for supporting grid workflow construction based on knowledge. Cunningham, P. and Cunningham, M. (eds.), *Innovation and the Knowledge Economy: Issues, Applications, Case Studies*, October (2005), vol. 2, pp. 239–246, IOS Press.
9. Fensel, D. and Bussler, C., The Web Service Modeling Framework WSMF. Online, http://www.swsi.org/resources/wsmf-paper.pdf.
10. Smith, M. K., Welty, C., and McGuiness, D. L., OWL web ontology language guide. Online, http://www.w3.org/TR/owl-guide.
11. Hobbs, J. R. and Pan, F. An ontology of time for the semantic web. *ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing*, March (2004), vol. 3, pp. 66–85.
12. Niles, I. and Pease, A. Origins of the standard upper merged ontology: A proposal for the IEEE standard upper ontology. *Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, Seattle, Washington, USA, August (2001).
13. Mika, P., Oberle, D., Gangemi, A., and Sabou, M. Foundations for service ontologies: Aligning owl-s to dolce. *The Thirteenth International World Wide Web Conference Proceedings*, New York, NY, USA, May (2004), pp. 563–572.
14. Martin, D. et al., OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/.
15. Krawczyk, K., Slota, R., Majewska, M., Kryza, B., and Kitowski, J. Grid organization memory for knowledge management for grid environment. *Proceedings of 4th Cracow Grid Workshop*, Cracow, Poland, December (2004), ACK Cyfronet-AGH.
16. Ogsa-dai project homepage. http://www.ogsadai.org.uk/.

The Grid Portlets Web Application: A Grid Portal Framework

Michael Russell[1], Jason Novotny[2], and Oliver Wehrens[1,3]

[1] Max-Planck-Institut für Gravitationsphysik,
Albert-Einstein-Institut, Golm, Germany
[2] University of California, San Diego,
CA, USA
[3] Center for Computation & Technology,
Louisiana State University, Baton Rouge, LA 70803

Abstract. Grid portals build upon the familiar Web portal model to offer virtual communities of users a single point of access to computational resources. The Grid Portlets web application, developed by the GridSphere Project, builds upon the core features in the GridSphere Portal Framework to provide developers with a framework for developing Grid portals.

1 Introduction

The Web has proven to be an effective means for delivering information, business services, and commercial applications to virtual communities of users. Web portals [1], like Yahoo [2] or MSN [3], offer users a single point of access to a wide variety of services and technologies. Grid portals build upon this successful model to provide a single point of access to computational resources: clusters, data servers, applications, scientific instruments and computing services.

2 The GridSphere Project

The GridSphere Project [4] was created by the GridLab Project [5] to leverage the most relevant standards, best-practices and technologies to offer a framework for developing Grid portals. One of the most exciting standards to gain adoption by the general community is the Portlet Java Specification Request (JSR 168) [6]. The Portlet JSR defines an application programming interface (API) and model for packaging and presenting Web content as *portlets*. Portlets are Java classes that have a clearly defined interface and life cycle. Portlets are hosted by a *portlet container* and can be presented in a Web page in any manner supported by the portlet container. The Portlet JSR makes it possible to distribute and share Web applications more easily, creating a means for collaborating on Web portal development on a much larger-scale.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 691–698, 2006.

3 Grid Portlets Web Application

The Grid Portlets web application, released for the first time in June 2005, builds upon the core features in the GridSphere Portal Framework [7] to provide developers with a framework for developing Grid-enabled portlets.

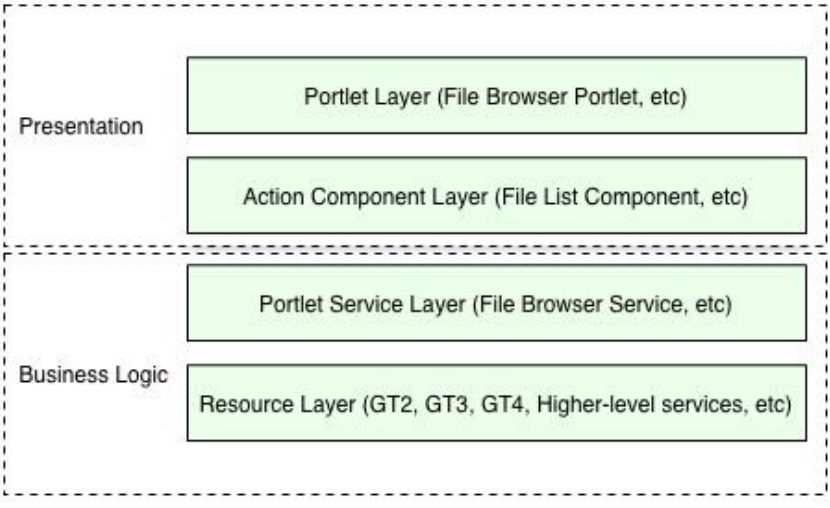


Fig. 1. Grid Portlets has a layered architecture

Grid Portlets has a layered architecture. At the top-most presentation level, Grid Portlets provides a collection of simple, easy-to-use, well integrated portlets that can be used on their own or in conjunction with third-party web applications to provide a complete Grid portal solution. These portlets are built upon reusable Java Server Pages (JSP) [8] based user interface components, called *action components*. The *Job Submission Portlet*, for example, utilizes the *File List Component* to enable users to "browse" for executable files, directories in which to run the executables, where to save job output and so on. The same component is also employed by the *File Browser Portlet* to display two "file browsers" in one window for searching for files and transferring files from one location to another. At the next layer down Grid Portlets offers developers a collection of *portlet services*, Java components, for performing tasks on the Grid. These portlet services define an API for interacting with Grid resources. Finally, the *resource layer* binds the portlet service layer to particular Grid infrastructures and technologies.

4 Grid Portlet Services

The portlet services contained in Grid Portlets offer a high-level API and model of the Grid, enabling developers to reuse the functionality offered in Grid Portlets

to develop custom Grid portal web applications. The Grid Portlet Services API is a collection of Java interfaces and base classes that:

- Abstracts developers from particular Grid technologies and infrastructure.
- Can be supported with Globus Toolkit 2 (GT2), GT3, GT4 and other service-oriented technologies. [9] [10] [11]
- Provides support for persisting information about resources and tasks performed by users on the Grid.
- Is extensible, builds upon simple concepts to provide more complex services, resources and tasks.

4.1 Resources

Central to the Grid Portlet Services API is the notion of a *resource*. Grid Portlets defines a resource as "anything that can be utilized". Resources have a *distinguished name* within the context of Grid Portlets, contain a set of *resource attributes*, and can contain other *child resources*. This very simple concept is quite useful and easy to extend. Grid Portlets defines several key resource concepts that extend from this base definition:

- *Hardware resource* - A hardware resource represents a particular host on the Grid. Hardware resources contain service resources, software resources and hardware accounts, all of which are described below.
- *Service resource* - A service resource is a resource that represents a "service" that is accessible over a network. All service resources have at least one "port" associated with them, through which they can be invoked by clients, and a "protocol" for communicating with the service.
- *Software resource* - A software resource is a resource that represents a "software". At minimum, a software resource has a "path" on a given hardware resource.
- *Resource account* - A resource account represents an "account" on a resource. For example, a *hardware account* represents an account on a particular hardware resource.

4.2 Key Portlet Services

Grid Portlets defines several portlet services interfaces, some of which are described here.

Resource Registry Service. The *Resource Registry Service* is responsible for making resources available to portlets and other portlet services, where resource descriptions are maintained in the Grid Portlets database. Resource descriptions are registered in the database through one of two means, with a file called Resources.xml, located in the WEB-INF directory of the Grid Portlets web application context, and with *resource provider services*, described below. At startup time, the Resource Registry Service will delete any hardware resource entries currently stored in the Grid Portlets database, save the hardware resource entries described in Resources.xml to the database, then activate the resource provider services that have been registered with Grid Portlets.

Resource Provider Services. Grid Portlets supports the ability to discover and monitor resources with resource provider services, where resource provider services are responsible for querying (or registering Grid Portlets for notification) the various *information resources* that have been registered with Grid Portlets. Several resource providers can be active at any one time. Grid Portlets is distributed with an *MDS2 Resource Provider Service* that will periodically query *Grid Resource Information Service (GRIS)* [13] resources registered with Grid Portlets for information about the hardware resources on which they are hosted.

Credential Manager Service. The *Credential Manager Service* is responsible for managing *credential contexts* and making active Generic Security Service (GSS) [14] *credentials* that are registered for those contexts available to portlets and other portlet services. A credential context associates a *distinguished name (DN)* with a portal user, records the last time an active credential with that DN was registered with the Credential Manager Service and other useful information about credentials with that DN. A DN can be registered with only one user, while a given user can be associated with multiple DNs. Essentially, Grid Portlets supports the use of multiple credentials, allowing users to gain access to resources from multiple *virtual organizations*.

Credential Retrieval Service. Grid Portlets supports the ability to retrieve GSS credentials from remote *credential repository resources*, such as MyProxy [15], with the *Credential Retrieval Service*. The Credential Retrieval Service is responsible for managing *credential retrieval contexts*. A credential retrieval context associates information for retrieving a credential with a particular DN to a portal user. A portal user can be associated with multiple credential retrieval contexts, thereby allowing a user to retrieve multiple GSS credentials.

File Browser Service. The *File Browser Service* provides methods for creating *file browsers* to remote *file resources*. A file browser represents a *stateful connection* to a file resource and provides methods for listing files, changing directories, creating directories, transferring files between locations and other basic file operations. The File Browser Service is employed by the *File Browser Portlet* and other portlet services for gaining access to remote files.

Job Submission Service. The *Job Submission Service* is a service for submitting jobs to remote *job resources* and monitoring their status. Job resources are service resources that provide access to one or more *job schedulers*, where job schedulers define one or more *job queues* to which jobs can be submitted. The Job Submission Service can support job submission to multiple types of job resources at runtime. The GridSphere Project and contributors to the GridSphere code base have developed support for job submission to Globus Gatekeeper (GT2) [12], Master Managed Job Factory Service (MMJFS) [16], and the Grid Resource Management Service [17]. Users of the job submission service can opt to allow the job submission service to determine which type of job resource to use for a given job specification or choose on their own.

5 Grid Portlets

The portlets contained in the Grid Portlets web application leverage the Grid portlet services described above to provide a generic, yet powerful set of user interfaces for using resources on the Grid.

5.1 Resource Registry Portlet

Grid Portlets is essentially configured through the *Resource Registry Portlet*. Here, portal administrators can specify the set of resources their Grid portal makes available to its users by editing the Resources.xml file online.

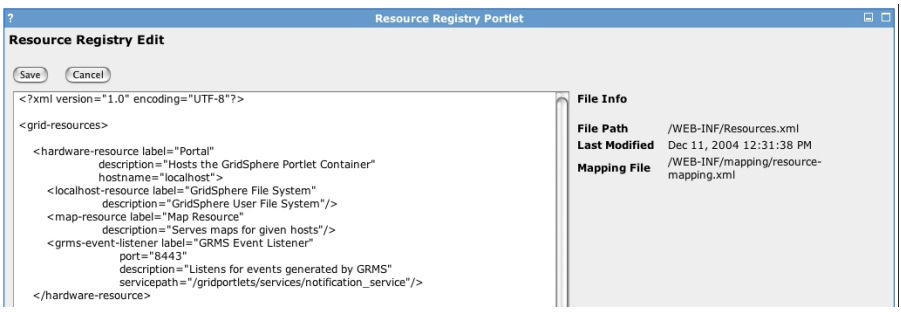


Fig. 2. Edting the resource registry with the Resource Registry Portlet

5.2 Resource Browser Portlet

The *Resource Browser Portlet* enables users to view information about the hardware resources that have been registered with Grid Portlets, including the services, job queues and accounts that are available on those resources.

5.3 Credential Manager Portlet

The *Credential Manager Portlet* enables users to delegate credentials to the portal from credential repositories, such as MyProxy, and monitor the status of those credentials. When a user defines a new credential retrieval context, they can specify whether to enable that credential for *single sign-on*. When single sign-on is enabled, the next time a user logs into the portal, if they enter their credential retrieval password, then their credential will be automatically delegated to the portal on their behalf.

5.4 File Browser Portlet

Users can browse for files on remote file systems with the *File Browser Portlet*. The File Browser Portlet displays two file browser user interfaces side-by-side,

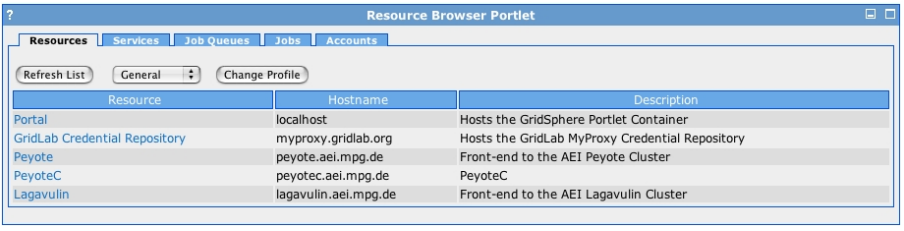


Fig. 3. Viewing resources with the Resource Browser Portlet

enabling users to transfer files and directories between locations. The file transfers are performed asynchronously, so that control returns immediately to the user after starting a file transfer. Users can then monitor the progress of the transfer with the *File Activity Portlet*, described below. The File Browser Portlet also supports the ability to create new directories, delete files and directories, edit text files online, upload files to remote file systems and download files to a user's local host, so long as the files aren't too large.

5.5 File Activity Portlet

The *File Activity Portlet* allows user to monitor the progress of file tasks, such as *file copy tasks*, *file move tasks*, and *file upload tasks*.

5.6 Job Submission Portlet

The *Job Submission Portlet* enables users to submit jobs to remote job schedulers and monitoring their status. Because the Job Submission Service employs the Job Submission Service, it supports the ability to submit jobs to multiple types of job resources. If more than one job resource is registered with Grid Portlets, then Grid Portlets will make each job resource type available as a "job submission service" selection when a user clicks on the *New Job* button.

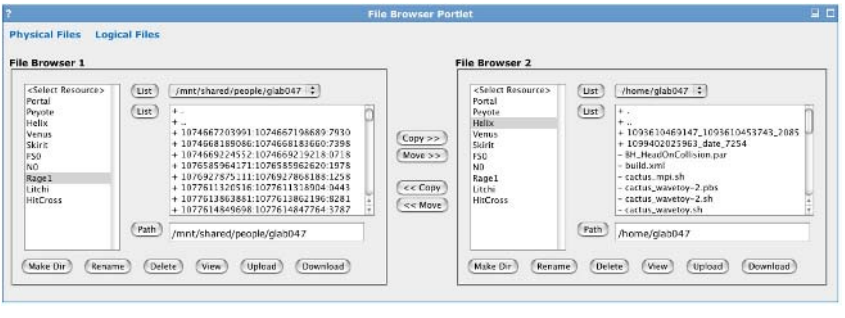


Fig. 4. Selecting files to transfer with the File Browser Portlet

Fig. 5. Selecting a job submission service with the Job Submission Portlet

6 Conclusion

The GridSphere Project is proud to offer the Grid Portlets web application to the general community. This paper covers only a subset of the features Grid Portlets offers, which includes support for logical files, support for plugins for viewing resources and editing jobs, and much more. We have many plans in store for Grid Portlets, including future co-releases of GT3 Portlets and GT4 Portlets projects which offer GT3 and GT4 implementations of the Grid Portlets Services API respectively.

Acknowledgments

The authors wish to thank Ian Kelley for his valuable thoughts and input to the GridSphere Project over the past years. We give special thanks to Dr Ed Seidel and Dr Gabrielle Allen without whom the GridSphere Project would not have been possible. We also would like to thank Dr Ian Foster, Dr Carl Kessleman and Steve Teucke for creating the Globus Toolkit and founding the field of Grid computing. Finally, we would like to acknowledge the contributions and ideas of Jean-Claude Cote, Robert Parrott and many others who have helped to evolve the Grid Portlets web application. The GridSphere Project was supported by the Albert-Einstein-Institut under the GridLab Project, an European Commission 5th Framework program (grant IST-2001-32133), from May 2002 to April 2005.

References

1. Web Portals. A Word Definition from Webopedia. http://www.webopedia.com/TERM/W/Webportal.html
2. Yahoo! Personalized content and search options. Chatrooms, free e-mail, clubs, and pager. http://www.yahoo.com/
3. MSN. Features personalization, channels of content sites like Carpoint, and integration with Hotmail e-mail. http://www.msn.com/
4. GridSphere Project. http://www.gridsphere.org/
5. GridLab: A Grid Application Toolkit and Testbed. http://www.gridlab.org/
6. Java Community Process: *JSR 168 Portlet Specification*. Project Website. Dec 1, 2004. http://www.jcp.org/jsr/detail/168.jsp

7. J. Novotny, M. Russell, O. Wehrens. *"GridSphere: An Advanced Portal Framework*, EUROMICRO, 2004
8. Java Community Process: *JSR 152 Java Server Pages 2.0 Specification.* http://www.jcp.org/en/jsr/detail?id=152
9. Globus Toolkits. http://www.globus.org/toolkit/
10. I. Foster, C. Kesselman, J. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.* Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.
11. I. Foster, J. Frey, S. Gram, S. Teucke, C. Czajkowski, et al. *Modeling Stateful Resources With Web Services* http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.html, March 5, 2004.
12. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems.* Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.
13. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. *Grid Information Services for Distributed Resource Sharing.* Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
14. RFC 2853 - Generic Security Service API Version 2 : Java Bindings http://www.faqs.org/rfcs/rfc2853.html
15. J. Novotny, S. Tuecke, V. Welch. *An Online Credential Repository for the Grid: MyProxy.* Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
16. V. Silva. *The Master Managed Job Factory Service (MMJFS): A custom GRAM client for the Globus Toolkit 3.x.* http://www-106.ibm.com/developerworks/grid/library/gr-factory/
17. K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, J. Pukacki. *Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS.* Scientific Programming, IOS Press. Amsterdam The Netherlands 12:4 (2004) 263-273

A Grid Service for Management of Multiple HLA Federate Processes

Katarzyna Rycerz[1], Marian Bubak[1,2],
Maciej Malawski[1], and Peter M.A. Sloot[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30,30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11,30-950 Kraków, Poland
[3] Faculty of Sciences, Section Computational Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{kzajac, bubak, malawski}@uci.agh.edu.pl, sloot@science.uva.nl
Tel.: (+48 12) 617 39 64, Fax: (+48 12) 633 80 54

Abstract. The subject of this paper is a Grid management service called *HLA–Speaking Service* that interfaces an actual High Level Architecture (HLA) application with the Grid HLA Management System (GHMS). *HLA–Speaking Service* is responsible for execution of an application code on the site it resides and manages multiple federate processes. The design of the architecture is based on the OGSA concept that allows for modularity and compatibility with Grid Services already being developed. We present the functionality of the Service with an example of N–body simulation of dense stellar system.

Keywords: HLA, Grid, distributed interactive simulations, federate management, N-body simulation.

1 Introduction

Distributed simulations often require extensive computing resources. The Grid [2, 3] is a promising means of solving this problem as it offers the possibility to use resources which are not centrally controlled and are under different administrative policies. Simulations built with an implementation of the High Level Architecture (HLA) system [6] allow for merging geographically-distributed parts (called *federates*) of simulations (called *federations*) into a coherent entity. The High Level Architecture is explicitly designed as support for interactive distributed simulations, it provides various services required for that specific purpose, such as time management, useful for time-driven or event-driven interactive simulations. It also takes care of data distribution management and enables all application components to see the entire application data space in an efficient way. On the other hand, the HLA standard does not provide automatic setup of HLA distributed applications and there is no mechanism for migrating federates according to the dynamic changes of host loads or failures, which is essential for Grid applications. Therefore, there is a need for a system that would manage HLA-based simulations on the Grid. The Grid Services concept provides a good starting point for building the Grid HLA Management System (GHMS) for that

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 699–706, 2006.

purpose, as described in [12]. In this paper we focus on the HLA management service called *HLA–Speaking Service* that interfaces actual HLA application with GHMS and is responsible for execution of application code on the site it resides. The service manages multiple federate processes and is based on experience obtained from its prototype for single processes[15]. We also show how the distributed N–body simulation of a dense stellar system can benefit from the Grid by using this service. Such simulations remain a great challenge in astrophysics [17] and there is a need for a computer infrastructure that will significantly improve their performance. We believe that the Grid is a promising environment for such requirements, since it offers the possibility of accessing computational resources that have heretofore been inaccessible.

The paper is organized as follows: in Section 2 related works are presented, in Section 3 we describe the *HLA–Speaking Service*. Section 4 presents the application for the N–body simulation and shows how this application is run with GHMS. In Section 5 we present the migration overhead and improvement of performance of N-body simulation resulting from GHMS, and we conclude the paper in Section 6.

2 Related Works

Execution and management of HLA–based simulation on the Grid is a very interesting and important problem. [1] describes design and implementation of a load management system for running large-scale HLA simulations in a Grid environment based on Globus Toolkit 2. These authors also present a framework where the user can design parallel and distributed simulations without prior knowledge of HLA/RTI [13]. In [14] they presented a protocol that supports efficient checkpointing and federate migration for dynamic load balancing. In order to facilitate the usage of this protocol, an additional layer between HLA and the actual application was proposed[13]. Although this approach presents a more efficient migration protocol, it is not sufficient for porting HLA legacy code to the Grid. In [16] a framework for executing large-scale distributed simulations using Grid services was presented. Currently, this framework addresses dynamic discovery of HLA federates. The approach assumes that each federate is a simulation model encapsulated in a Grid Service. These solutions are complementary to the approach presented in this paper. In [16] there is an assumption that a user is able to build a simulation from existing Grid Services (which encapsulate functionality of HLA federates) or can wrap federates into Grid Services (the paper does not, however, describe how to wrap a federate into a Grid Service, which is a nontrivial issue). This paper describes a *HLA–Speaking* Grid Service that allows a user to simply load legacy federates and manage execution of the simulation. The useability of these approaches depends on simulation requirements.

3 HLA–Speaking Service

The role of HLA–Speaking service in GHMS is illustrated in Fig.1. The group of main GHMS services consists of a *Broker Service* which coordinates

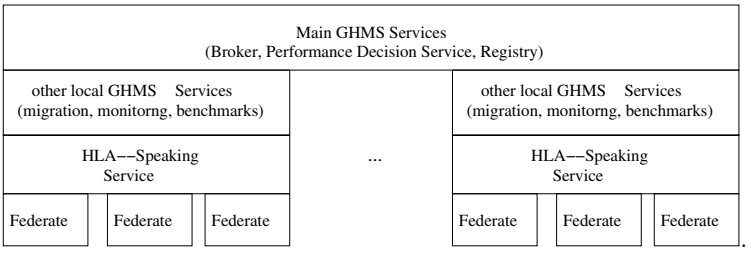


Fig. 1. The *HLA–Speaking Service* in the context of GHMS architecture

management of the simulation, a *Performance Decision Service* which decides when the performance of any of the federates is not satisfactory and therefore migration is required, and a *Registry Service* which stores information about the location of local services. On each Grid site supporting HLA there are local services for performing migration commands on behalf of the *Broker Service* as well as for monitoring of federates and benchmarking. The *HLA–Speaking Service* is one of the local services interfacing federates running on its site to the GHMS system.

3.1 Multiple Processes Support

The *HLA Speaking Service* encapsulates various HLA implementations that are thought of as resources found in a Grid environment. The service is one of the most important parts of GHMS, as it manages the execution of the federates on a particular site. The service starts the execution of a federate code on its site, saves the application state and restores it from a checkpoint file. An HLA-specific

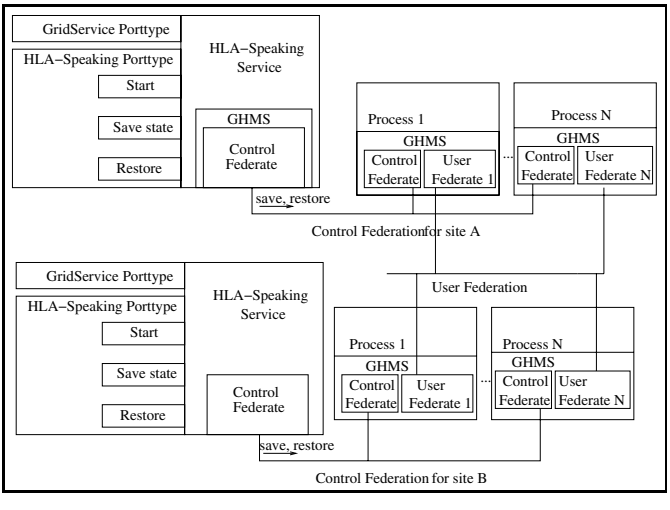


Fig. 2. Conceptual view of *HLA Speaking Service* for multiple federate processes

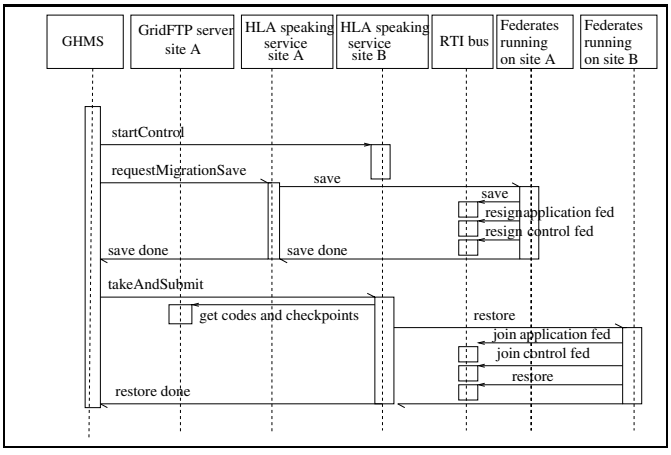


Fig. 3. Sequence diagram describing federate migration for multiple federates

API is hidden in the RTI library functionality and is not exposed as a service interface. Experience gained with *HLA Speaking Service* for single processes[15] has enabled us to design support for multiple federates (see Fig. 2). In this approach, we designed the control federation for forwarding requests from the Grid Service layer to the actual user federates. The Grid Service layer can request saving and restoring specific processes. The user code communicates with the control federate by means of a Grid HLA Controler Library (GHCL). The user federates communicate with each other also using RTI, but this takes place in the scope of user-defined federations.

The operations of the *HLA–Speaking Service* PortType are modified to support multiple federates and use the Resource Specification Language (RSL) from Globus GRAM [4] for description of federate processes to be run and include functions for running, saving and restoring of HLA federates.

3.2 Migration Scenario

The scenario of migration is shown in Fig 3. If there is a need to migrate, the *GHMS* informs the *Broker Service* which decides that the HLA federate from site A has to be migrated to site B and triggers the *Migration Service* to perform this migration. The *Migration Service* creates the *HLA Speaking Service* on the new site, then asks for creation of a control federate that creates the control federation. When this is done, the *GHMS* asks the *HLA–Speaking Service* on site A to save the state. The *HLA Speaking Service* forwards this request with the identifiers of federates that have to be saved to all federates it controls by means of GHCL. The federates invoke an HLA API for federation saving [6] which suspends the other federates. The federates know (by their identifiers) if there is a request to save the state of their federations and resign from them Subsequently, user data is saved in a checkpoint file and the appropriate federates resign from the federations defined by the user (application federations) as well

as from the control federation. Next, the *GHMS* invokes the `takeAndSubmit` operation of the *HLA-Speaking Service* on site B which triggers it to fetch the user codes and checkpoint files from site A to site B and restore federates from the available checkpoint files. The federates join the control federation as well as user-defined (application) federations and invoke the GHCL library which restores user–specific data and uses the HLA API [6] for restoring the application federation RTI internal state.

4 Feasibility Study with N—Body Simulation

The N-body simulation is part of the virtual laboratory called Gravitylab [7] which is a powerful software environment for experiments concerning the dynamics of dense stellar systems. The algorithm of the simulation uses a direct method, where the gravitational force acting on a particle is computed by summing up the contributions from all other particles according to the Newton's law

$$\boldsymbol{F_i} = m_i * \boldsymbol{a_i} = -Gm_i \sum_{j=1, j\neq i}^{j=N} \frac{m_j(\boldsymbol{r_i} - \boldsymbol{r_j})}{|\boldsymbol{r_i} - \boldsymbol{r_j}|^3}$$

The numerical method is based on the Hermite [8] scheme, where higher-order derivatives are explicitly computed in order to construct interpolation polynomials of the force. The performance analysis of different parallelization schemes for direct code used in the simulation of astrophysical stellar systems for different computer architectures, including the Grid, can be found in [5].

The application consists of two modules: the MPI N–body simulation module and the integrated visualization–interaction module. The modules communicate using the HLA RTI bus which connects visualization/interaction federate with the MPI root process of the simulation federate. Those federates form the application federation. Additionally, HLA is used for control of MPI simulation processes on the Grid site as described in Section 3.1. Therefore, all MPI processes join the control federation together with the control federate in the HLA–speaking service. For connection with the control federate, all processes use the GridHLAController library.

5 Results

We have performed two different experiments to show the overhead of various migration stages and its impact on simulation performance. The experiments were performed on the DutchGrid DAS2 testbed infrastructure and at CYFRONET, as shown in Table 1. Because the bandwidth available for testing was broad (10Gbps), communication did not play an important role and calculations were the most time–consuming part of the execution. In order to create conditions in which migration would be useful, we increased the load of the Grid site where the simulation was executed (Amsterdam) by submitting nonrelated,

Table 1. Grid testbed infrastructure

Operating System	RedHat Enterprise Linux Advanced Server, version 3		
Network	10 Gbps (DAS2) + 155 Mbps (DAS2-Cyfronet)		
Role	**Name**	**CPU**	**RAM**
Migration source	DAS2 Nikhef	Pentium III 1 GHz	1 GB
Migration dest	DAS2 Delft	Pentium III 2GHz	2 GB
Interation federate	Cyfronet Krakow	2.40 GHz Xeon	1 GB
RTIexec	Cyfronet Krakow	2.40 GHz Xeon	1 GB

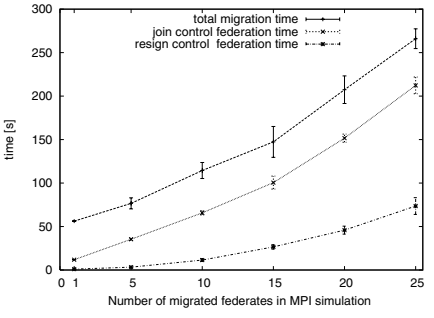


Fig. 4. The time of migration stages dependent on MPI federate number

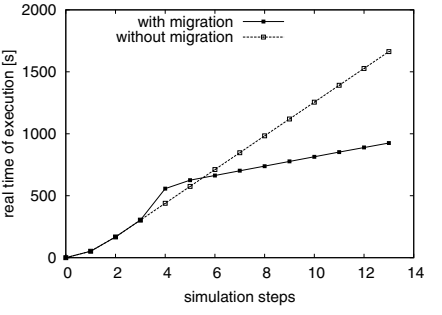


Fig. 5. Impact of migration on performance of N-body simulation

Table 2. Execution time of various migration stages

	$\bar{x}$ [s]	σ [s]
save time	6.4	1.3
application resign time	3.7	0.5
transfer time	0.4	0.05
GRAM	2.5	0.1
PBS waiting time	8.5	0.7
application join time	12.9	1.9
restore time	8.2	0.8

computationally–intensive jobs. Next, we imitated a Resource Broker and migrated the simulation to another site which was not overloaded (Delft). The experiments were performed at night in order to avoid interference with other users and repeated 10 times to check if they are reproducible.

Migration time. Fig. 4 shows migration time as a function of the number of MPI processes in the federation. For our tests, we used GT v3.2 and HLA RTI 1.3v5. The number of stars used in the simulation was 5000. The results were obtained as an average of 10 measurements. The error bars indicate estimated standard deviation denoted by σ. The time includes phases as shown in Fig. 3 in Section 3.2. The overhead of particular phases is shown in Fig. 4 (activities dependent on the number of MPI processes) and Tab. 2 (activities independent

of the number of MPI processes). As can be observed, the largest overhead in the case of the tested application is for rejoining the control federation. This is due to the fact that in the RTI implementation we used, every federate has to open a TCP connection to every other federate in the control federation. Reliable multicast [9] would significantly decrease this overhead. Other activities, such as saving and restoring the state of the federation application, joining and resigning the federation application, transferring files (GridFTP overhead), GRAM and PBS induce overhead which does not depend on the number of MPI federates since only one root MPI process is actually joining the application federation in order to communicate with the visualization federate, as shown in Tab. 2.

Impact of migration on performance. As in the previous experiment, for our tests we used GT v3.2 and HLA RTI 1.3v5. The simulation consisted of 20 MPI processes calculating 3D position and velocity of 24000 stars. Fig. 5 shows the time as a function of number of interactive steps in the loop with a human (for the first 13 steps) In each step, the simulation calculates data and sends them to the visualization and interaction module using HLA. The dashed line shows what would happen if the network stayed busy for some time and the responses took longer. In this situation it is better to spend some time on migration to the other site, which offers better response times as shown by a solid line in the figure. Fig. 5 shows that the user can gain access time between steps 5 and 6 independently of the time lost on migration(performed between steps 3 and 4).

6 Summary and Conclusions

This paper presented the *HLA–Speaking service* which enables efficient management of the execution of HLA code on the Grid. The service is a part of the GHMS system described in [12] and it is used for running, saving and restoring one or more federate processes on the Grid site on which it resides. We have also presented a sample application managed by our service – N–body simulation of a dense stellar system. Our solution gives the ability to not only apply the HLA standard on the Grid, but also to easily adapt legacy HLA applications to the Grid environment in a robust and efficient way.

Acknowledgments. The authors would like to thank Piotr Nowakowski for useful remarks. This research is partly funded by the EU IST Project CoreGRID, the Polish State Committee for Scientific Research SPUB-M grant, and by the Dutch Virtual Laboratory for e-Science project (www.vl-e.nl).

References

1. W. Cai, S.J. Turner, and H. Zhao. A Load Management System for Running HLA-based Distributed Simulations over the Grid. In S.J. Turner and S.J.E. Taylor, editors, *Proceedings of the sixth IEEE International Workshop on Distributed Simulation and Real Time Applications (DS-RT 2002)*, pages 7–14, Fort Worth, Texas, October 2002. IEEE Computer Society.

2. I. Foster. What is the Grid? A three checkpoints list. *GridToday Daily News And Information For The Global Grid Community*, 1(6), July 2002.
3. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002. `http://www.globus.org/research/papers.html`.
4. Globus Project Home Page. `http://www.globus.org/`.
5. A. Gualandris, S.P. Zwart, and A. Tirado-Ramos. Performance analysis of direct N-body algorithms on highly distributed systems. *IEEE Transactions on Parallel and Distributed Systems*. Submitted, available at `http://arxiv.org/abs/astro-ph/0412206`.
6. HLA specification. `http://www.sisostds.org/stdsdev/hla/`.
7. P. Hut and J. Makino. The art of computational science. `www.artcompsci.org/`.
8. J. Makino and S.J. Aarseth. On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems. *PASJ*, 44:141–151, April 1992.
9. D.M. Moen, J.M. Pullen, and F. Zhao. Implementation of host-based overlay multicast to support of web based services for rt-dvs. In *8th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2004), 21-23 October 2004, Budapest, Hungary*, pages 4–11. IEEE Computer Society, 2004.
10. MPI Forum Home Page. `http://www.mpi-forum.org`.
11. RSL – Globus Resource Specification Language v1.0. `http://www.globus.org/gram/rsl_spec1.html`.
12. K. Rycerz, M. Bubak, M. Malawski, and P. Sloot. A Framework for HLA-Based Interactive Simulations on the Grid. *SIMULATION*, 81(1):67–76, 2005.
13. Z. Yuan, W. Cai, and M.Y.H. Low. A Framework for Executing Parallel Simulation using RTI. In S.J. Turner and S.J.E. Taylor, editors, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 20–28, Delft, The Netherlands, October 2003, IEEE.
14. Z. Yuan, W. Cai, M.Y.H. Low, and S. J. Turner. Federate Migration in HLA-Based Simulation. In M. Bubak, G. D. van Albada, P. M.A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 856–864, Krakow, Poland, June 2004. Springer-Verlag.
15. K. Zając, M. Bubak, M. Malawski, and P.M.A Sloot. Towards a Grid Management System for HLA-Based Interactive Simulations. In S.J. Turner and S.J.E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 4–11, Delft, The Netherlands, October 2003. IEEE Computer Society.
16. W. Zong, Y. Wang, W. Cai, and S.J. Turner. Grid Services and Service Discovery for HLA-Based Distributed Simulation. In *8th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2004), 21-23 October 2004, Budapest, Hungary*, pages 116–124. IEEE Computer Society, 2004.
17. S.P. Zwart, H. Baumgardt, P. Hut, J. Makino, and S. L. W. Mcmillan. Formation of massive black holes through runaway collisions in dense young star clusters. *Nature*, 428:724–726, April 2004.

Algorithms for Automatic Data Replication in Grid Environment

Renata Słota[1], Łukasz Skitał[1],
Darin Nikolow[1], and Jacek Kitowski[1,2]

[1] Institute of Computer Science,
AGH-UST, al. Mickiewicza 30, 30-059, Kraków, Poland
[2] Academic Computer Center CYFRONET,
ul. Nawojki 11, 30-950 Kraków, Poland
{rena, kito}@agh.edu.pl

Abstract. Replication methods can be used for shortening the data access time in Grid environment with heterogeneous storage resources. In this paper we describe the algorithms for automatic replication implemented by us in two grid projects having different requirements concerning data storage.

1 Introduction

In Grid environments, data sets are stored in special nodes called storage elements (SEs). The SEs are usually different and this heterogeneity introduces variations of data access time. Optimization of data access is necessary to keep the access time as low as possible. Data replication is one of the methods coping with the problem of data access optimization. It attempts to solve the network bandwidth limitation problem, as well as allows to balance the storage load between many sites and increases the data protection level.

Data management systems developed in previous grid projects offer limited support for data replication so called static replication. In the case of static replication, the decision about creation or removal of a replica is made by the user or by the system administrator. This method is not suitable especially for the future invisible grid environments, which size, dynamics and heterogeneity successfully prevent any human from optimal data management. In the case of automatic (also called dynamic) replication the decision about a replica creation and removal is done by a dedicated grid component – Replica Manager (RM) – usually a part of the data management system.

We have already implemented an automatic replication method in two Grid projects, SGIgrid and Clusterix [1, 2], having different requirements concerning the data storage. Automatic replication algorithms, some implementation details and review of test results are presented in this paper.

The rest of the paper is organized as follows: Section 2 presents state of the art. Sections 3,4,5 give description of the proposed algorithms. Test results are presented in Section 6. Section 7 concludes the paper.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 707–714, 2006.

2 State of the Art

Ranganathan and Foster in [3] evaluate, using simulations, different dynamic replication strategies for managing large data sets in a high performance data grid. Their research has been done within the framework of the GriPhyN project [4]. They define six different replication strategies and test them by generating three different kinds of access patterns. They show how the bandwidth saving and latency differ with these access patterns depending on the applied strategy. The best strategies according to this paper are Fast Spreading and Cascading with Caching, but in the same time these strategies are the most storage consuming ones.

Park et al. in [5] present a simulation based study of a dynamic replication method called BHR (Bandwidth Hierarchy Replication). The BHR strategy takes advantage of the network-level locality which does not always fit to the geographical one. They show that their strategy outperforms other strategies in the case when hierarchy of bandwidth appears in the Internet.

Lamehamedi et al. in [6] investigate the use of highly decentralized dynamic replication services for improving the access time and bandwidth consumption of the overall system. They have assumed a two-tier data grid. They have tested two replication strategies: replicating to level 1 nodes (closer to tier 0) and replicating to level 2 intermediate nodes (closer to clients). Their simulation results show that the performance gains increase with the size of the data.

Izmailov et al. in [7] propose the parallel replicas propagation methods for the Grid environment. The Fast Parallel File Replication tool is presented. The tool creates multiple distribution trees by pipelining point-to-point GridFTP transfers and optimizes the file replication time to multiple sites. Four algorithms for creation of replicas distribution tree are discussed: hierarchical, iterative DFS, iterative BFD and iterative shortest-widest. The performance results show a significant speed up of up to five times using the proposed tool, as compared to the basic point-to-point GridFTP usage.

3 Algorithms for Automatic Data Replication

Automatic replication is a complex task, which requires a set of algorithms concerning the creation, removal, selection and coherency of replicas as well as replica update propagation algorithm.

The replica creation algorithm is responsible for automatic creation of new replicas. This algorithm has to choose files, which need to be replicated and to decide, where to place replicas. Some information is necessary to perform these tasks. The algorithm needs a list of file access read and write requests, information about SEs, like their capacity, type (HDD, HSM, DB) and performance. The information about network, like bandwidths between clients and SEs, is also very important.

The replica removal algorithm is responsible for replicas removal intended to save storage space. There are a lot of replica removal strategies. The most plain

strategy is to remove replicas, which were not used for some, specified time. Also the algorithm can remove replicas, which reside on SEs with small amount of free space. The algorithm can dynamically adjust its parameters depending on the available storage space and replica usage.

The replica selection algorithm is necessary to take a performance gain from replication. The algorithm is responsible for optimal replica selection for the specific read or write operation. Without this algorithm, the only advantage of the replication is the protection from data loss. In the process of optimal replica selection information about current system state is needed, concerning especially network bandwidth that is available between endpoints and SEs. Generally the replica selection algorithm minimizes the cost of data access. The definition of the cost function depends on the implementation and the available information.

The problem of replica coherency is a complex one and it is similar to the cache coherency problem. The strong coherency model assures, that only up-to-date replicas can be accessed. The weak coherency model allows access to older versions of replicas.

The replica update propagation algorithm is responsible for updating out-of-date replicas. The method used for the update task should provide a balance solution between the system load and the time needed to perform replica updates.

The next sections describes the algorithms for automatic data replication implemented in the projects Clusterix and SGIgrid.

4 Clusterix Data Management System

The Polish grid project Clusterix [2] – National Cluster of Linux Systems – is designed for use with computational and data intensive applications. The purpose of the project is to develop a set of tools and procedures allowing to built a productive grid environment based on local PC clusters (64bit or 32bit) located in independent supercomputing centers. The project uses Pionier [8] infrastructure as a network layer.

The Clusterix Data Management System (CDMS) takes advantage of the dynamic replication. Three kinds of replication are used: initial replication, replication on demand and replication based on statistic analysis.

4.1 Algorithms

Replica selection algorithm. The replica selection algorithm is used for selection of optimal SE for data accessing. Elaboration of this algorithm is necessary, to achieve the desired performance gain. By our definition the optimal SE can deliver the data to a destination node in the shortest time. The algorithm takes into account the current system state, obtained from the Network Monitoring System and the JMX-based Infrastructure Monitoring System.

The optimal SE is the element, s_o, for which $W(s_o, d) = \max_s W(s, d)$, where the weight function

$$W(s,d) = \min[(1 - L_{net}(s))B_{net}(s,d), (1 - L_{stor}(s))B_{stor}(s)] \qquad (1)$$

and s - storage element; d - destination node; $L_{net}(s)$ - s network interface load; $B_{net}(s,d)$ - available bandwidth between s and d; $L_{stor}(s)$ - storage system load; $B_{stor}(s)$ - storage system bandwidth.

The weight function finds the bottleneck between SEs and destination node. It takes the smaller value from between the available network and storage bandwidths.

Automatic replication algorithm. The automatic replication is based on the statistical data describing historical file access patterns. The algorithm takes into account the gain from replication $G()$, the cost of replica creation $C()$, the cost of replicas update $U()$ and the administrative factor $A()$.

For every file and every SE (with sufficient free space) the replication profit $P()$ is computed:

$$P(d,R,S,f) = G(d,R,S,f) - \alpha_1 C(d,R,f) - \alpha_2 \frac{S_w}{S_r} U(d,R,S,f) + A(d,f) \quad (2)$$

where: d - storage element, which profit is computed for; f - considered file; R - set of SEs containing replicas of f; S - statistical data - history of file usage; S_w, S_r - number of writes, reads of f; α_1, α_2 - tuning coefficients, specified empirically by the administrator.

The replication is performed, if the profit $P()$ is grater then some threshold value. The function $G()$ prescribes the gain from replication in respect of time. The gain function is defined as follows:

$$G(d,R,S,f) = \sum_{s \in S} \frac{size(f)}{B_{avg}(opt(R,s),s)} - \sum_{s \in S} \frac{size(f)}{B_{avg}(opt(R \cup d,s),s)} \quad (3)$$

where: $size(f)$ - size of file f; s - node, which have accessed f; $opt(R,s)$ - optimal SE from R for node s; $B_{avg}(d,s)$ - average bandwidth between nodes d and s.

The function computes the time for file transfers before replication and the hypothetical time, which would be spent on the file transfer if the file had been replicated. The gain is a difference between these two time values.

The function $C()$ prescribes the cost of replica creation in respect of time. The cost function is defined as follows:

$$C(d,R,f) = \frac{size(f)}{B_{avg}(d,opt(R,d))} \quad (4)$$

The function computes the time, which would be spent to transfer the replica to a new location.

The function $U()$ prescribes the cost of replicas update in respect of time. The cost function is defined as follows:

$$U(d,R,f) = \frac{(n-1) \cdot size(f)}{B_{avg}(d,opt(R,d))} \quad (5)$$

where n is the number of replicas of file f. The function computes time, which would be spent to update all of the replicas of file f.

The administrative factor is set for every SE and allows to control the value of the profit function. It can be used to encourage or discourage the replication to given SE.

Replica removal algorithm. Automatic replication without replica removal can cause high storage usage, so an appropriate replica removal algorithm has to be specified. There are three cases in the CDMS, when a replica can be removed:

1. the profit function for the replica is less than the settled minimum,
2. the free space on storage system is low and the profit function for the replica is less than the average (or other threshold value),
3. the free space on storage system is at critical level.

5 Virtual Storage System for SGIgrid

The SGI Grid project [1] provides services for remote access to expensive scientific equipment, backup computational center and remote data-visualization service. Scientific experiments often produce large amount of data, which have to be stored or archived. Data archivization aspects are addressed by Virtual Storage System (VSS), which main goal is to integrate storage resources distributed among computational centers into a common system. VSS can use a variety of storage resources, like databases, file systems, tertiary storage (tape libraries and optical jukeboxes managed by HSM systems). In order to optimize the access to so different storage resources the automatic replication methods have been implemented in the VSS.

5.1 Algorithms

Replica selection algorithm. The selection of optimal replica is performed using information about network latency between each data container and the client. Because there is none network monitoring system in the SGIGrid project, the network latency is measured by a simple service using *traceroute* utility. The SGIGrid support variety of storage systems, which have different performance and their data access time can vary from milliseconds to ten of minutes, thus the additional information about access time and read/write performance is taken under consideration. The estimated access time is provided by a specialized service. The read/write performance test are performed on every data container.

The optimal replica is the one, which is stored on the container with the minimal weight function d:

$$d(c, u, f) = n(c, u) + r(c) + w(c) + eta(f), \tag{6}$$

where: $n(c, u)$ - network latency between data container c and user location u, $r(c)$ - storage read performance test, $w(c)$ - storage write performance test, $eta(f)$ - storage latency estimation for file f.

Automatic replication algorithm. Automatic replication in SGIGrid is based on statistic data, extracted by the Log Analyzer from system logs. These data contain information about the following events: file creation/removal, replica creation/removal, read/write access.

For each file, which fulfills initial constraints (file size, number of replicas, etc.) a list of pairs (u_i, w_i) is created. u_i is the ip address of requesting node and w_i is the weight, associated with the file usage from this ip address, defined as:

$$w_i = \sum_{a \in L_r(u_i)} f(t_a), \tag{7}$$

where: $L_r(u_i)$ - set of read events performed from u_i address, t_a - timestamp for the read event, $f(t_a)$ - function which controls the influence of the timestamp on the value of the weight.

For each u_i for which $w_i > W_{min}$ an optimal data container, k_i, is selected from the set of containers K. W_{min} is a threshold value. The selection algorithm is similar to the optimal replica selection algorithm (see Section 5.1. The only difference is, that the estimated access time is not taken under consideration. As the result the triples (u_i, w_i, k_i) are produced. The triples are grouped in U_{K_j} sets, where $j = 1..m$ and m is the number of containers. These sets are defined as follows:

$$U_{K_j} = \{(u_i, w_i, k_i), i = 1..n : k_i = K_j\}, \tag{8}$$

where n is the number of triples. Next a set indexes $I_{U_{K_j}}$ is defined:

$$I_{U_{K_j}} = \{i : (u_i, w_i, k_i) \in U_{K_j}\}, \tag{9}$$

and for each data container K_j, a weight z_{K_j} is computed:

$$z_{K_j} = \sum_{l \in I_{U_{K_j}}} w_l, \tag{10}$$

The weight represents the level of usage of the replica from this container. Then the container with maximal weight is chosen to store the new replica.

Replica removal algorithm. To maintain moderate storage usage, VSS removes unused replicas. The algorithm removes replicas, which have been created by RM and have not been used for some (configurable) time. Replicas created manually by the users are not removed.

Replica coherency algorithm. The algorithm is used, when a user updates an existing file, which has been replicated. The algorithm outlines as follow:

1. The user requests file update.
2. RM selects the optimal replica for update and sets its state to `IN_TRANSFER`, all remaining replicas states are set to `STALE`.
3. User performs update (file transfer).
4. The state of updated replica is set to `READY`, RM sends a list of stale replicas and the location of updated replica to RM.
5. RM updates the stale replicas using the updates propagation algorithm.

Updates propagation algorithm. We defined two sets: S - set of up-to-date replicas, initially contains one replica, and D - set of out-of-date replicas, initially contains all replicas but the updated one.

While D is not empty:

Create pairs (s_i, d_j), $s_i \in S$, $d_j \in D$. s_i and d_j can belong to one pair only. For each pair the file transfer $s_i \rightarrow d_j$ is started. After each file transfer, d_j is moved from D to S.

When the D set is empty, all replicas are updated.

6 Test Results

Clusterix and SGIgrid are not large-scale grids with about 5-10 SEs located in Poland. Clusterix uses homogeneous optical network with bandwidth of 10 Gb/s, while SGIgrid uses heterogeneous network with bandwidth between 10 Mb/s and 1 Gb/s. More detailed information about the tests can be found in [9].

Clusterix. CDMS is still in development, thus the real system tests were not possible. Presented results were obtained from CDMS simulator, which has been developed to test algorithms. The typical applications running on Clusterix are simulations which are run many times using the same data set but with different simulation parameters. That is why the case of running grid applications operating on their constant input data files has been tested. The tests have shown that using automatic replication in this case causes a 12% increment of storage usage and a 15% decrement of the overall data access time in compare with the case with only initial replication.

SGIgrid. SGIgrid is already in production, so the tests have been done on the real system. Anyway the assumptions are different from the Clusterix ones. SGIgrid is focused on accessing unique scientific equipment in a virtual laboratory as well as on the further accessing of the data produced during experiments in the laboratory. In this case scientists from different locations access the data files. As a result some data files are more frequently accessed for a given time period.

The performance tests of automatic replication for SGIgrid were done by generating file access requests from different locations. The file access pattern represented the Zipf-like distribution. The result have shown that the increment of storage usage in this case is 49% and the decrement of the overall data access time is 64%.

7 Conclusions

In this paper we have presented automatic replication algorithms for Grid environment. Some implementation details and review of test results for two grid projects Clusterix and SGIgrid have been presented. For these types of grids the tests indicate that the automatic replication can cause decrement of total data access time. The test results have shown that in the case of Clusterix despite the high bandwidth homogeneous network (which makes the gain smaller) and

the initial replication, the automatic replication can bring some profit for the simulation experiments run on a typical computational grid. The gain is due to the better load balancing of the storage elements. In the case of SGIgrid, which is a kind of data grid with heterogeneous network connections and SEs, the gain from automatic replication is higher.

Acknowledgments. The work described in this paper was supported by the Polish Committee for Scientific Research (KBN) projects "Clusterix" 6T11 2003C/06098 and "SGIgrid" 6 T11 0052 2002 C/05836. AGH grant is also acknowledged.

References

1. SGIgrid: Large-scale computing and visualization for virtual laboratory using SGI cluster (in Polish), KBN Project, http://www.wcss.wroc.pl/pb/sgigrid/
2. Clusterix - http://clusterix.pcz.pl/indexEng.php
3. Ranganathan, K., Foster, I., "Identifying Dynamic Replication Strategies for a High-Performance Data Grid", Proceedings of the International Workshop on Grid Computing, Denver, November 2001.
4. Grid Physics Network `http://www.griphyn.org/`
5. Park, S., Kim, J., Ko, Y., Yoon, W., "Dynamic Data Grid Replication Strategy Based on Internet Hierarchy", Lecture Notes in Computer Science Publisher, Springer-Verlag Heidelberg Volume 3033, 2004, pp.838-846.
6. Lamehamedi, H., Szymanski, B., Deelman, E., "Data Replication Strategies in Grid Environments," Proceedings of the 5th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP2002, Beijing, China, October 2002, IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.
7. Rauf Izmailov, Samrat Ganguly, Nan Tu, "Fast Parallel File Replication in Data Grid", Future of Grid Data Environments workshop, GGF - 10, Berlin.
8. PIONIER - Polish Optical Internet, `http://www.pionier.gov.pl/eindex.html`
9. Slota, R., Nikolow, D., Skital, L., Kitowski, J., "Implementation of Replication Methods in the Grid Environment", Lecture Notes in Computer Science, no. 3470, Springer, 2005, pp. 474-484.

A Grid Workflow Language Using High-Level Petri Nets

Martin Alt[1], Andreas Hoheisel[2], Hans-Werner Pohl[2], and Sergei Gorlatch[1]

[1] Westfälische Wilhelms-Universität Muenster, Germany
{mnalt, gorlatch}@uni-muenster.de
[2] Fraunhofer FIRST, Berlin, Germany
{andreas.hoheisel, hans.pohl}@first.fraunhofer.de

Abstract. One approach to Grid application programming is to implement services with often-used functionality on high-performance Grid hosts. Complex applications are created by using several services and specifying the workflow between them. We discuss how the workflow of Grid applications can be described easily as a High-Level Petri Net (HLPN), in order to orchestrate and execute distributed applications on the Grid automatically.

Petri Nets provide an intuitive graphical workflow description, which is easier to use than script-based descriptions and is much more expressive than directed acyclic graphs (DAG). In addition, the workflow description can be analysed for certain properties such as deadlocks and liveness, using standard algorithms for HLPNs. We propose a platform-independent, XML-based language, called Grid Workflow Description Language (GWorkflowDL), and show how it can be adapted to particular Grid platforms. As two example target platforms, we discuss Java/RMI and the current WSRF standard.

1 Introduction

An approach to Grid programming that receives much attention is the use of remotely accessible services which are implemented on Grid hosts and provide commonly used functionality to applications running on clients. Popular examples of Grid middleware following the paradigm of the *Service-Oriented Architecture (SOA)* are the OGSI-compliant Globus Toolkit 3 [1] and the Web Services Resource Framework (WSRF) standard [2] with several implementations, such as Globus Toolkit 4 and WSRF.net.

Grid applications for service-based systems are usually composed of several services working together in an application-specific manner. An application developer has to decide which services offered by the Grid should be used in the application, and he has to specify the data and control flow between them. We will use the term *workflow* to refer to the automation of both – control and data flow – in Grid applications.

In order to simplify Grid programming, it should be possible to describe an application workflow in a simple, intuitive way. Script-based workflow descriptions (e.g. GridAnt [3], BPEL4WS [4]) explicitly contain a set of specific workflow constructs, such as *sequence* or *while/do*, which are often hard to learn for unskilled users. Purely graph-based workflow descriptions have been proposed (e.g. for Symphony [5] or Condor's DAGman tool [6]) which are mostly based on Directed Acyclic Graphs (DAGs). Compared to script-based descriptions, DAGs are easier to use and more intuitive: communications between different services are represented as arcs going from one service

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 715–722, 2006.

to another. However, DAGs offer only a very limited expressiveness, so that it is often hard to describe complex workflows, e.g. loops cannot be expressed directly. High-Level Petri Nets (HLPNs) allow for nondeterministic and deterministic choice, simply by connecting several transitions to the same input place and annotating edges with conditions. Similarly, since DAGs only have a single node type, data flowing through the net cannot be modelled easily. In contrast, HLPNs make the state of the program execution explicit by tokens flowing through the net.

We propose a Grid Workflow Description Language (GWorkflowDL) [7] based on HLPNs. The novelty of our approach is that we do not modify or extend the original HLPN model in order to describe services and control flow: we use the HLPN concept of edge expressions to assign a particular service to a transition, and we use conditions as the control flow mechanism. The resulting workflow description can be analysed for certain properties such as deadlocks and liveness, using standard algorithms for HLPNs. The language itself is platform-independent and provides platform-specific language extensions to adapt the generic workflow to a particular Grid platform. In this paper, we present a Java/RMI as well as a WSRF extension. The GWorkflowDL is intended to provide a common approach for the whole life-cycle of Grid applications, consisting of the workflow orchestration, scheduling, enactment, execution, and monitoring. The GWorkflowDL is currently the basis for the K-Wf Grid project [8], and the Java Grid of the University of Muenster. The Fraunhofer Resource Grid [9] uses a similar approach.

The structure of the paper is as follows: In the next section, we present the underlying Grid infrastructure and describe an example application. In Section 3 we give a brief introduction to High-Level Petri Nets and discuss how HLPNs are used to model service-based Grid applications. Section 4 presents the basic features of the Grid Workflow Description Language and the specific extensions for WSRF and Java/RMI platforms. We conclude our paper in the context of related work.

2 Grid Architecture and Application Example

We assume a Grid system architecture as shown in Fig. 1, where application programs are constructed using a set of services which are implemented on remote high-performance hosts. Services are invoked from a client on remote Grid hosts using a remote method invocation mechanism such as Java/RMI or SOAP. We will use the term *service* for any remotely accessible functionality which transforms a number of input parameters into one or several result values.

As an application example (also shown in Fig. 1), we use the complex convolution function often used for linear filters in signal processing. The convolution function can be computed efficiently using the Fast Fourier Transform (FFT) and its inverse FFT^{-1} as follows:

$$f_{conv}^{(n,n)}(a,b) = FFT_{2n}^{-1}(FFT_{2n}(a) \times FFT_{2n}(b)) \tag{1}$$

where $\times$ denotes pointwise complex multiplication.

We express the convolution according to (1) using three different services: two services for FFT and FFT^{-1} respectively, and a third service, called *zip*, for pointwise complex multiplication. When program (1) is executed on the client, the different

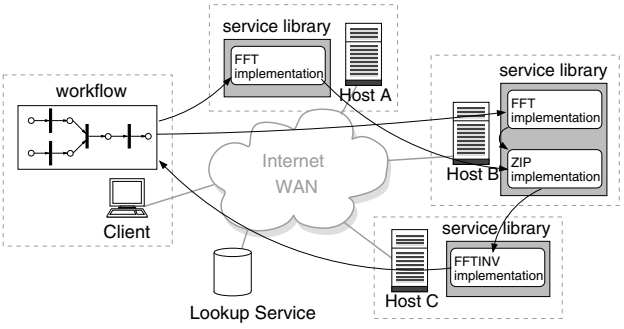


Fig. 1. Prototype Grid architecture

services (FFT, zip and FFT^{-1}) are called remotely on (potentially different) Grid hosts, with parameters a and b being sent across the network.

In the remainder of the paper, we develop a workflow for our application example and show how it can be described using HLPNs.

3 Using Petri Nets for Describing Workflows

Our graphical notation for Grid workflow is based on High-Level Petri Nets (Petri Nets with individual tokens), which allow to compute the value of output tokens of a transition based on the value of the input tokens. An introduction to the theoretical aspects of HLPNs can be found, e.g., in [10]. Van der Aalst and Kumar [11] give an overview of how to describe different workflow patterns using Petri Nets.

Petri Nets are directed graphs, with two distinct sets of nodes: *transitions* (represented by thick vertical lines or rectangles) and *places* (denoted by circles). Places and transitions are connected by directed edges. An edge from a place p to a transition t is called an *incoming edge* of t, and p is called *input place*. Outgoing edges and output places are defined accordingly. Each place can hold a number of individual *tokens* that represent data items flowing through the net. A transition is *enabled* if there is a token present at each of its input places. Enabled transitions can *fire*, consuming one token from each of the input places and putting a new token on each of the output places. The number and values of tokens each place holds is specified by the *marking* of the net. Consecutive markings are obtained by firing transitions.

Each edge can be assigned an *edge expression*. For incoming edges, variable names are used, which assign the token value obtained through this edge to a specific variable. Additionally, each transition can have a set of boolean *condition* functions. A transition can only fire if all of its conditions evaluate to true for the input tokens.

In the next section, we discuss how individual services can be represented as transitions in High-Level Petri Nets, and show how applications built from services are described by more complex nets. We will discuss the workflow concepts using an abstract mathematical notation for edge expressions and show concrete examples for Java/RMI and WSRF in Section 4.2.

3.1 Convolution Example

We will now illustrate the representation of Grid workflows as HLPNs using the convolution example from equation (1). The resulting net is shown in Fig. 2. Individual services, such as the zip and FFT services are represented by single transitions. Transitions $load_a$, $load_b$ and $save$ represent local methods used to load the parameter values and save the result.

The service name is written above the transition, variables for the formal parameters and results are represented as places, with parameter names (e.g. x and y for zip) shown as edge expressions on the incoming edges. The edge expressions for outgoing edges specify which value should be placed on the corresponding output place when the transition fires. Usually, this is the result of the invoked service, but it can also be an error value. In addition to the service itself, a set of conditions may be associated with a transition. For example, the zip service (for pointwise complex multiplication of two arrays, see Sect. 2) is only allowed to be executed if the two input lists are of the same length, which is expressed by $|x| = |y|$. The conditions are shown below the corresponding transition.

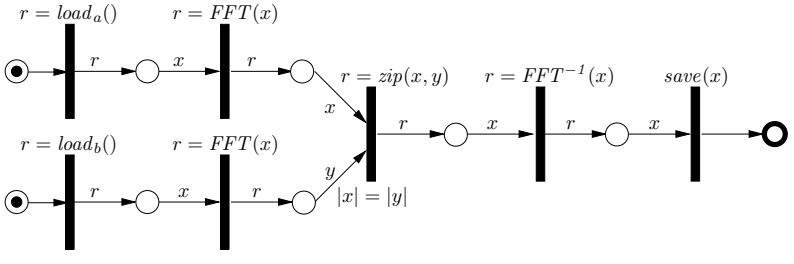


Fig. 2. Petri Net representation of convolution program

In addition to the places and edges for input and output data of transitions, the application developer has the possibility to introduce additional *control places* to the graph. A control place holds simple tokens which do not carry any value. Accordingly, input edges connecting a control place to a transition (*control edges*) have no associated variables, and the token values are not used as parameters for the associated service. Control edges just synchronize the firing of a transition with the corresponding control place. As example, consider the transitions $load_a$ and $load_b$ in Fig. 2. Because the methods for loading the parameters do not have any input parameters, a control input edge is introduced for each of them. The control places initially contain a single control token which is consumed when parameters are loaded. This ensures that the corresponding transitions are executed only once. Similarly, the $save$ transition has an outgoing control edge which signals the completion of the $save$ service.

Conditions can be used to check whether the input data of the service meets certain requirements. Additionally, the application programmer can use conditions to realise standard control flow structures, such as conditions and loops. For example, FFT implementations often can only be used on input lists of length $n = 2^i$. This should be

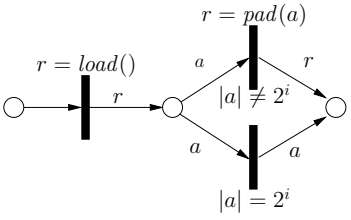


Fig. 3. Conditional padding of the input data realised with user-defined conditions

checked by a condition provided with the FFT service. However, to use the convolution program in equation (1) for input lists of lengths that are not a power of two, the lists can be zero-padded to the appropriate length.

Figure 3 shows a subgraph for loading list a with padding to the required length. The $padding$ transition calls a service to transform the input list, which is only necessary if the list does not have the correct size. Therefore, the user-defined condition $n \neq 2^i$ is used to prevent unnecessary padding. If the list already has the correct length, it is passed through the second transition which has no associated service and does not change its input.

4 Grid Workflow Description Language

The GWorkflowDL is being developed as an XML-based language for Grid workflows, based on HLPNs as described in the previous section. It consists of a generic part, used to define the structure of the workflow, and a platform-specific part (*extension*) defining how to execute the workflow in the context of specific Grid computing platforms.

4.1 GWorkflowDL – XML Schema

Figure 4 graphically represents the XML Schema of GWorkflowDL. The root element is called `<workflow>`, which contains the optional element `<description>` with a human-readable description of the workflow, and several occurrences of the elements `<transition>` and `<place>` that define the Petri Net of the workflow.

The element `<transition>` may be extended by platform specific child elements, such as `<WSRFExtension>` and `<JavaRMIExtension>`, which represent special mappings of transitions onto particular Grid platforms. Elements `<inputPlace>` and `<outputPlace>` define the edges of the net. Edge expressions are represented as attribute `edgeExpression` of `JavaRMIExtension` and `WSRFExtension` tags.

4.2 GWorkflowDL – Platform Extensions

To adapt a generic workflow description to a particular Grid computing platform, we use *extensions*, which describe the meaning of a generic net in the context of a particular platform. We will now present two example extensions, for WSRF and Java/RMI. Platform extensions define: (1) the platform-specific service to be invoked; (2) how

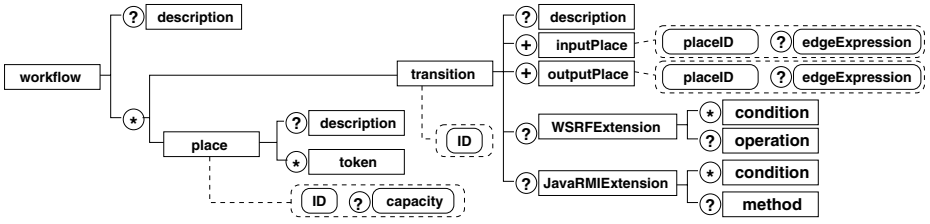


Fig. 4. Graphical representation of the XML schema for GWorkflowDL. Boxes denote elements, rounded boxes represent attributes. Legend: $? = 0, 1; * = 0, 1, 2, \ldots; + = 1, 2, 3, \ldots$

conditions are evaluated; (3) how edge expressions are evaluated. The GWorkflowDL document for the zip service shown in Fig. 2 is as follows:

```
<workflow>
 <place ID="P1"/> <place ID="P2"/> <place ID="R"/>
 <transition ID="ZIP">
  <inputPlace placeID="P1" edgeExpression="x"/>
  <inputPlace placeID="P2" edgeExpression="y"/>
  <outputPlace placeID="R" edgeExpression="result"/>
  <JavaRMIExtension>
   <method name="result = Zip.execute(x,y)"/>
   <condition name="Zip.checkLength(x,y)"/>
  </JavaRMIExtension>
  <WSRFExtension>
   <operation name="zip" owl="gom.kwfgrid.net/zip.xml">
    <WSClassOperation name="zipXY" owl="kwfgrid.net/zipXY.xml">
     <WSOperation name="zipXY@first" owl="first.fhg.de/zip.xml"/>
     <WSOperation name="zipXY@iisas" owl="savba.sk/zip.xml"
                  selected="true"/>
     <WSOperation name="zipXY@cyfro" owl="agh.edu.pl/zip.xml"/>
   </WSClassOperation> </operation>
  </WSRFExtension>
 </transition>
</workflow>
```

Note that this code is not intended to be written by the programmer, but instead can be generated automatically from Java or WSDL interfaces, or by workflow orchestration tools for combining several services. For the Java extension, *edge expressions* assign variable names, and the *method* element captures services and describes how the methods and conditions should be called. The *condition* elements provide conditions which contain a Java expression that depends on the input variables and yields a boolean value. The code also shows the abbreviated GWorkflowDL representation of the *zip* using the `WSRFExtension` as used in K-Wf Grid. The *edge expressions* and *conditions* may be specified as XPath queries. The *operation* element captures several levels of abstraction of web service operations: *operation* describes an very abtract operation without any details, *WSClassOperation* specifies a operation on specific class

of Web Services described by their interfaces and functionality, and *WSOperation* are the concrete instances of Web Service operations that match the class. The *owl* attribute links to external semantic descriptions.

4.3 Workflow Orchestration and Execution

To design a Grid program, the application developer first selects the services required for the application and creates an abstract workflow, such as the net shown in Fig. 2. The resulting abstract workflow description can already be analysed for certain properties such as deadlocks and liveness, using standard algorithms for HLPNs (see e.g. [12]).

After selecting an appropriate service-based Grid computing platform, the application developer has to adapt the abstract Petri Net to the particular platform by assigning particular services and platform-specific edge expressions to transitions. E.g., for the Java platform, a Java method of a remote interface is assigned to each transition, and variable names are assigned to the input and output edges. The resulting specific HLPN for the desired workflow can then be executed on the Grid by assigning an executing host to each service, either manually (to execute the application on a user-selected set of hosts) or automatically, using a scheduling strategy to select hosts.

Execution then starts by selecting an enabled transition. The tokens on the input places are collected and the transition's conditions are evaluated. If all conditions yield true, the corresponding service is invoked, with the data related to the tokens on input places as input parameters. The result is then placed as token on the output places. If any condition evaluates to false, the input tokens are returned to their respective input places. Then the next enabled transition is selected. This process continues until each terminal place holds at least one token, or no enabled transitions remain.

5 Conclusions

We have presented our approach for expressing Grid application workflows as High-Level Petri Nets and described GWorkflowDL, an XML-based language for Grid workflows. Petri nets are widely used for modelling and analysing business workflows in workflow management systems (e.g. [13]). The use of Petri Nets for Grid workflow has first been proposed as Grid Job Definition Language (GJobDL) [14] for job-based Grid systems, where a Grid application is composed of several *atomic Grid jobs* which are sent to the hosts for execution. The GJobDL language is similar to the GWorkflowDL, but it uses a modified HLPN where transitions contain input and output *ports*, representing parameters and results, and edges connect places to ports instead of transitions.

In contrast, our GWorkflowDL specifies Grid workflow as a standard HLPN (as defined e.g. in [10]), using conditions for control flow and edge expressions to assign parameters and results. Adhering more strictly to the standard model of HLPNs allows us to make use of standard algorithms for analysing Nets, e.g. for deadlocks.

The HLPN representation of a workflow serves four main purposes: (1) It is an intuitive graphical description of the program, making communication between services explicit and allowing users to develop programs graphically without having to learn a specific workflow language. (2) Because applications are developed as unmodified

HLPNs, the application's HLPN can be used for analysis and formal reasoning based on the results of previous research in High-Level Petri Nets. (3) The same GWorkflowDL description can be used to monitor and inspect running and finished workflows. (4) Because the GWorkflowDL is divided into an abstract and a platform-specific part, it can be used with different service implementations and Grid platforms.

As future work, we plan to implement a set of tools for workflow orchestration, execution, monitoring, and analysis, based on the GWorkflowDL. For example, we intend to implement performance prediction of Grid applications by using time values as tokens and service functions that add the expected performance of particular services (which can be obtained using an approach discussed in [15]) to the input tokens.

Acknowledgements

Our work is supported in part by the European Union through the IST-2002-004265 Network of Excellence CoreGRID and the IST-2002-511385 project K-WfGrid.

References

1. Foster, I., et al.: The physiology of the grid: An open grid services architecture for distributed systems integration. In: Open Grid Service Infrastructure WG, Global Grid Forum. (2002)
2. Czajkowski, K., et al.: The WS-Resource Framework (2004) http://www.globus.org/wsrf/.
3. von Laszewski, G., Alunkal, B., Amin, K., Hampton, S., Nijsure, S.: GridAnt – client-side workflow management with Ant. http://www-unix.globus.org/cog/projects/gridant/ (2002)
4. Andrews, T., et al.: Business process execution language for web services version 1.1. Technical report, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems (2003)
5. Lorch, M.: Symphony – A Java-based Composition and Manipulation Framework for Computational Grids. PhD thesis, University of Applied Sciences in Albstadt-Sigmaringen, Germany (2002)
6. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience (2004)
7. Hoheisel, A., H.-W.Pohl: Documentation of the Grid Workflow Description Language toolbox. http://fhrg.first.fraunhofer.de/kwfgrid/gworkflowdl/docs/ (2005)
8. K-Wf Grid consortium: K-Wf Grid homepage. http://www.kwfgrid.net/ (2005)
9. Fraunhofer Gesellschaft: Fraunhofer Resource Grid homepage. http://www.fhrg.fraunhofer.de/ (2005)
10. Jensen, K.: An introduction to the theoretical aspects of Coloured Petri Nets. In de Bakker, J., de Roever, W.P., Rozenberg, G., eds.: A Decade of Concurrency. Volume 803 of Lecture Notes in Computer Science., Springer-Verlag (1994) 230–272
11. van der Aalst, W.M.P., Kumar, A.: Xml based schema definition for support of inter-organizational workflow. University of colorado and university of eindhoven report (2000)
12. Girault, C., Valk, R., eds.: Petri Nets for Systems Engineering. Springer-Verlag (2003)
13. van der Aalst, W.: The application of Petri Nets to workflow management. The Journal of Circuits, Systems and Computers **8** (1998) 21–66
14. Hoheisel, A., Der, U.: An XML-based framework for loosely coupled applications on grid environments. In Sloot, P., ed.: ICCS 2003. Number 2657 in Lecture Notes in Computer Science, Springer-Verlag (2003) 245–254
15. Alt, M., Bischof, H., Gorlatch, S.: Program development for computational Grids using skeletons and performance prediction. Parallel Processing Letters **12** (2002) 157–174

Towards a Language for a Satisfaction-Based Selection of Grid Services

Sergio Andreozzi[1,2], Paolo Ciancarini[1],
Danilo Montesi[1], and Rocco Moretti[1,3]

[1] University of Bologna, Department of Computer Science, 40127 Bologna, Italy
{paolo.ciancarini, danilo.montesi, rocco.moretti}@cs.unibo.it
[2] Istituto Nazionale di Fisica Nucleare-CNAF, 40127 Bologna, Italy
sergio.andreozzi@cnaf.infn.it
[3] Dept. of Pure and Applied Mathematics, University of Padova,
35131 Padova, Italy

Abstract. Grid systems enable the sharing of a large number of geographically-dispersed resources among different communities of users. They require a mapping functionality for the association of users requests expressed in terms of requirements and preferences to actual resources. This functionality should deal with a potentially high number of similar resources and with the diversity of the perceived satisfactions of users. We propose XMatch, a query language enabling the expression of the user request in terms of the expected satisfaction over XML-based representation of available resources. This language improves the expressiveness of queries and supports aggregation of an high number of elementary satisfactions.

1 Introduction

Grid systems follow a new paradigm of distributed computing enabling the sharing of resources and services that are not subject to centralized control, are geographically dispersed and can dynamically join and leave virtual pools [1]. Users typically express their requests as constraints and preferences using a well-known set of terms defined in an information model and describing the resource characteristics. A resource request may involve different types of resources or more items of the same resource, thus requiring the capability of expressing interdependencies among them. Given the large number of possible attributes and the need for making an automatic selection, we need mechanisms for the efficient evaluation of the degree to which resources satisfy the request.

In our previous work [2], we have proposed a model for the rigorous representation of service characteristics, for the association of each of their possible values with the user satisfaction and for the aggregation of the single satisfactions in an overall score using a particular logic. In this paper, we propose XMatch, a query language enabling the expression of the user requirements and preferences based on the defined model. This language is inspired by XQuery [3] reusing a set of constructs useful for our goal and providing clauses based on our service evaluation model.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 723–730, 2006.

The paper is structured as follows: in Section 2 we present the background of the service selection in Grid systems and we describe a number of possible approaches; in Section 3, we introduce the XMatch language with a number of examples; in Section 4, we use the language in a use case part of the Grid computing area; in Section 5, we draw up our conclusions and plan for future work.

2 Background on Service Selection

As mentioned in the introduction, a key functionality of a Grid system is the capability of mapping users requests to available resources. As a matter of fact, current production Grid systems offer to users the mechanisms to describe their expectations in terms of constraints and preferences. Such descriptions are used by the selection service provided by the Grid middleware that implements the mapping functionality of abstract resources to physical ones. This service typically is in charge of the optimization of the workload among the available resources.

The description submitted by Grid users rely on a set of terms defined in an information model (e.g., [17, 16]). The formal definition of such terms are expected to comply with the empirical perception of the users. This requirement is presented and discussed in [2]. Users may express expectations over a large number of terms. These expectations may involve not only the values the terms take, but also how much is important that they are satisfied during the selection process.

An approach widely used in Grid systems (e.g., [7]) for the selection of services is based on the Classified Advertisement (ClassAd) language [13, 14]. This language has been designed in the context of the Condor distributed computing system [15] where it is used for discovery and allocation of resources. Its use consists of the following phases: (1) providers of computing resources submit advertisements describing their capabilities and declaring constraints and preferences for jobs that they are willing to run (constraints are boolean expressions involving constants or ClassAd attributes, while preferences are encoded in a rank that consists in defining an arithmetic expression synthesizing values used for sorting the services satisfying the constraints); (2) consumers submit advertisements describing their jobs and the desired execution environment in terms of constraints and preferences; (3) a matchmaker process matches the resource and consumer request advertisements.

It is worth to mention two important works in the databases community. They are relevant in our context because they propose extensions to query languages for the support of preference expressions. Chomicki proposed a logical framework for formulating preferences as strict partial orders by using arbitrary logical formulas [12]. In order to embed such formulas into relation algebra, a single winnow operator that can be parameterized by a preference formula was defined. This enables the rewriting of preference formulas as SQL queries. Our language is targeted at semi-structured data and our approach has been to define a new language. Kießling proposed a formal language for formulating preference queries based on the Best-Matches-Only (BMO) query model [8, 9]. It developed

a set of constructors and combinators that can be used to write preference expressions. An algebra modeling such operators was defined and extensions for both SQL (Preference SQL [10]) and XPath (Preference XPath [11]) were proposed. In our work, we focused on the XML data model and we have proposed a new language based on XQuery.

As regards the first work, we observe that in general it is difficult to define a rank expression aggregating values from different attributes. As regards the second work, the theoretical preference framework is mapped into the relational data model and the SQL query language. Concerning the third work, the theoretical preference framework is mapped into 1) the relational data model and the SQL query language and 2) the XML data model and XML path language (XPath). Our goal is to provide a query language enabling the expression of the user requirements and preferences over resource description based on the XML data model. This language should also support the simultaneous selection of a set of different resources. In the next section, we propose XMatch, a language inspired by XQuery with extensions enabling the expression of attribute relevance and perceived satisfaction based on the model defined in our previous work [2].

3 XMatch: The Language

In this section, we introduce the XMatch language enabling to express queries over XML-based representations of Grid resources by considering also the satisfaction degree that a user perceives as regards the possible values of the attributes of interest. The grammar rules are given only for symbols starting with the prefix `XM`; the other symbols are taken from the XQuery W3C specification [3]. The core part of the XMatch language is an expression represented in the grammar by the symbol `XMExpr`:

Rule 1

```
XMExpr ::= XMForClause XMLetClause+ XMatchWhereClause XMatchReturnClause
```

The rule defining this symbol is inspired by the FLWOR (For-Let-Where-Order by-Return) expression of the XQuery language. The clause `XMForClause` generates an ordered sequence of tuples of bound variables, called the tuple stream. This is a simplified version of the XQuery `ForClause`, where only the basic constructs are maintained to select elements from an XML document and to generate set of elements by using joins. For each XMatch expression, only one `XMForClause` is allowed with one or more variables to be bound to different types of nodes. The `URILiteral` is defined in the XQuery specification and should refer to a URI that can be resolved to a file containing the data in XML format from which the set of important fragments are extracted. The `OrExpr` is also part of the XQuery specification.

Rule 2

```
XMForClause ::= <"for" "$"> VarName "in" XMDocCall ("," "$" VarName "in" XMDocCall)*
XMDocCall   ::= "doc(" URILiteral ") XMPathExpr? XMPredicate?
XMPathExpr  ::= ("/" Literal)+
XMatchPredicate ::= "[" OrExpr "]"
```

The symbol `XMLetClause` is the fundamental construct to define the elementary criteria of satisfaction as presented in our previous work [2]. As mentioned above, the three main categories are: an enumeration of all possible values returned by a measurement of an attribute, an absolute classification of these values and a relative classification [5]. This clause is designed according to this classification.

Rule 3

```
XMLetClause    ::= ( XMSimpleEnum | XMCompEnum | XMRange )
XMSimpleEnum   ::= XMPathExpr ValueComp XMElement "satisfies" XMSatLiteral
                   ( "," XMElement "satisfies" XMSatLiteral )*
XMCompEnum     ::= XMPathExprList ValueComp XMElementList "satisfies"
                   XMSatLiteral ( "," XMElementList "satisfies" XMSatLiteral )*
XMRange        ::= XMPathExpr  "in" XMElement "to" XMElement "satisfies" "with"
                   ( "linear" "increment" | "linear" "decrement" | "around")
XMPathExprList ::= "(" XMPathExpr ( "," XMPathExpr )+ ")"
XMElementList  ::= "(" XMElement ( "," XMElement )+ ")"
XMElement      ::= ( Literal | XMPathExpr | XMFunctionCall )
XMSatLiteral   ::= ( "0"? "." Digits | "1" )
XMFunctionCall ::= ("max"|"min"|"avg"|"sum"|"count") "(" XMPathExpr ")"
XMPathExpr     ::= ("/" Literal)+
```

The first category of elementary criterion of satisfaction can be used when the number of possible values for which the user wants to express an explicit satisfaction is finite. For instance, a user may want to express a number of acceptable possibilities for the operating system type where its job should be executed; the most preferred one is `Scientific Linux`, but `RedHat Enterprise Linux` is acceptable with less satisfaction. This could be expressed as in Example 1 by using the `XMSimpleEnum` clause.

Example 1.
```
let $e1 := $CS/OSName eq "Scientific Linux" satisfies 1,
"RedHat Enterprise Edition" satisfies 0.8
```

A more complex use case supported in XMatch is the association of an elementary satisfaction by enumeration to a compound comparison predicated. For instance, if a user wants to express a satisfaction associated to both the name and version of an operating system, this can be done as in Example 2 by using the `XMCompEnum` clause.

Example 2.
```
let $e1 := ($CS/OSName, $CS/OSVersion) eq ("Scientific Linux", "3")
satisfies 1, ("RedHat Enterprise Edition", "3") satisfies 0.8
```

The second category of elementary criterion of satisfaction can be used when the expression of satisfaction is based on parameters independent from attribute values under investigation. The possible functions that map the range of values into a satisfaction are infinite. Our choice is to express in the language construct three meaningful of them. They are linear and capture an increasing satisfaction, a decreasing satisfaction or a satisfaction centered around a value. The first function (linear increment) can be used, for instance, when a user requires a storage service offering at least 50 gigabytes of available disk space with an increasing satisfaction up to 70 gigabytes. After that, everything is considered to be equal and fully satisfying. This can be captured in XMatch as in Example 3 by using the `XMRange` clause. The satisfaction is 0 if the value of the attribute is less than or equal to the lower bound of the range, while it is 1 if the value of the attribute is equal to or greater than the upper bound.

Example 3. `let $e2 := $SS/FreeSpace in 50 to 70 satisfies with linear increment`

The second linear function (linear decrement) is similar to the previous function, except that for values less than or equal to the lower bound, the satisfaction is 1 and for values equal to or greater then the upper bound, the satisfaction is 0. For attribute values in the range, the satisfaction linearly decreases. For instance, let us consider a user that is fully satisfied if the response time of a service is less than or equal to 5 ms, but he will accept values up to 30 ms (see Example 4).

Example 4. `let $e3 := $CS/EstimantedResponseTime in 5 to 30 satisfies with linear decrement`

The last function should be used when a high satisfaction is associated to values close to the one reputed to be the optimum. For instance, let us consider a user requiring a bandwidth for a network service around 200 Mbit/s with a 10% tolerance (see Example 5).

Example 5. `let $e4 := $NS/Bandwidth in 180 to 220 satisfies with around`

The third category of elementary criterion refers to a relative comparison among the attribute values of all entities in the evaluation context. This criterion requires the introduction of aggregation functions (e.g., min or max). They can be used in both the first and second elementary criteria given above. For instance, let us consider a user requiring the storage service offering the highest free storage capacity (see Example 6).

Example 6. `let $e5 := $SS/FreeSpace eq max($SS/FreeSpace) satisfies 1`

The use case of a relative elementary criterion concerning a range of values can be expressed by enabling the possibility of using aggregation functions to define the range bounds. For instance, a user may be fully satisfied when he is assigned for the service with the lowest response time among the available ones, but his satisfaction linearly decreases up to 0 when the highest is given.

Example 7. `let $e6 := $CS/EstimantedResponseTime in min($CS/EstimantedResponseTime) to max($CS/EstimantedResponseTime) satisfies with linear decrement`

Finally, we remark the assumption that the structure of the XML element that can be bound to a variable of the `XMForClause` must not contain in the same level elements with the same qualified name. This requirement will be removed in future evolution of the XMatch language. By means of the `XMWhereClause`, we explain how the association of a relevance category to an elementary satisfaction is modeled in the XMatch language. Potentially, the relevance categories can be infinite, but only three of them are introduced as they are sufficient for meaningful use cases. They are defined in the XMatch language grammar by using the following string literals: `essential`, `desirable` and `optional`. The advantage of such a definition is the improvement of the legibility of XMatch queries. A possible approach to generalize the language to an high number of relevance classes is to use natural numbers to label the relevance categories. The lower is the natural number associated to a relevance category, the more important is the satisfaction.

Rule 4

```
XMWhereClause   ::= XMRelevanceExpr ("and" XMRelevanceExpr)*
XMRelevanceExpr ::= ( "$" VarName "is" | XMVarNameList "are" )
                    ( "essential" | "desirable" | "optional" )
XMVarNameList   ::= "(" "$" VarName ( "," "$" VarName )+ ")"
```

Given a set of `XMLetClause` expressions defining elementary satisfactions, each of them can be associated with its relevance category by using the `XMWhereClause`. This provides the meaningful information for building the aggregation pattern. Weight and power parameters used in the aggregation pattern [2] are considered to be part of the query processor. The last rule introduced is `XMReturnClause` and describes how the result of the query is constructed and returned:

Rule 5

```
XMReturnClause ::= "return" "top" digits ( "with threshold" XMSatLiteral )?
```

Our decision is to define a clause that does not provide any transformation capability. If it is needed, the result can be transformed in a postprocessing phase using languages like XQuery or XSLT. The `XMReturnClause` returns an XML document with a predefined structure as presented in Example 8.

Example 8.

```
<Results>                                  <Result E="0.94">
 <Result E="0.98">                         ...
 ...                          ...          </Result>
 </Result>                                 </Results>
```

Each `Result` element contains a set of elements as generated by the `XMForClause` and an attribute named `E` with the overall satisfaction associated to the solution. The number of results can be limited in two ways: by asking the 'Top K' results and by dropping all solutions that do not reach a minimum overall satisfaction.

4 Use Case

In this section, we present a meaningful use case in the area of Grid computing. We consider a simplified scenario where core services referring to computing, storage and network resources are defined as follows: the *computing service* is a uniquely identified Grid service that can provide a user software application for computing power in a certain execution environment; the *storage service* is a uniquely identified Grid service that manages storage extents to be used for permanent data; the *network service* is a uniquely identified service that offers unidirectional communication capability between network domains that are meant as sets of services sharing the same connectivity (we refer to the model defined in [6]). The following XML fragments represent a computing, a storage and a network service. They are based on a schema and represent the mapping to a formal relational system. In a real scenario, a broker service maintains and continuously updates a cache with the representation of the available services [7]. The broker also receives requests from users and, based on their requirements and preferences, performs the matchmaking phase during which suitable solutions are prepared and one is selected.

```
<data>
 <CS>
  <URI>https://...</URI>
  <DomainURI>/INFN/CNAF</DomainURI>
  <Type>CS</Type>
  <OSName>SL</OSName>
  <OSVersion>3.0.3</OSVersion>
  <ProcFamily>IA32</ProcFamily>
  <AssignedJobSlots>30</AssignedJobSlots>
  <WaitingJobs>10</WaitingJobs>
  <RunningJobs>30</RunningJobs>
  <FreeJobSlots>0</FreeJobSlots>
 </CS>
<NS>
  <URI>/NS/1</URI>
  <Type>NS</Type>
  <SourceDomainURI>/INFN/CNAF</SourceDomainURI>
  <DestDomainURI>/INFN/MI</DestDomainURI>
  <Bandwidth Unit="Mbit/s">300</Bandwidth>
  <RoundTripTime Unit="ms">3</RoundTripTime>
</NS>
 <SS>
  <URI>https://...</URI>
  <Type>SS</Type>
  <DomainURI>/INFN/MI</DomainURI>
  <Durability>0.6</Durability>
  <AvSpace Unit="GB">300</AvSpace>
   <DataAvl>0.8</DataAvl>
 </SS>
</data>
```

Let us consider a user that requires a computing service offering the Scientific Linux (SL) operating system in its version 3.0.3, but also RedHat Enterprise Server (RHES) version 3.0.3 is acceptable with less satisfaction. Then, the user requires an Intel Architecture 32 bit (IA32) processor family. The application is expected to store permanent data for around 200 gigabytes in a storage service with a possible variation of 10%. A network service with around 70 Mbit/s of bandwidth and a small RTT (round trip time) is desirable. Finally, the minimization of the waiting time at the computing service is optional. A possible XMatch query can be written as follows:

```
for $NS in doc("data.xml")/data/NS,
    $CS in doc("data.xml")/data/CS[DomainURI=$NS/SourceDomainURI],
    $SS in doc("data.xml")/data/SS[DomainURI=$NS/DestDomainURI]
let $e1 := ($CS/OSName, $CS/OSVersion) eq ("SL", "3.0.3") satisfies 1,
           ("RHES", "3.0.3") satisfies 0.8
let $e2 := $CS/ProcFamily eq "IA32" satisfies 1
let $e3 := $SS/AvSpace in 180 to 220 satisfies with around
let $e4 := $NS/Bandwidth in 50 to 100 satisfies with linear increment
let $e5 := $NS/RoundTripTime in 0 to 10 satisfies with linear decrement
let $e6 := $CS/FreeJobSlots in 0 to max($CS/FreeJobSlots) satisfies with linear increment
where ( $e1, $e2, $e3 ) are essential and ( $e4, $e5 ) are desirable and $e6 is optional
return top 10 with threshold 0.6
```

Each `let` clause is used to express an elementary criterion of satisfaction, while the `where` clause describes a specific instance of the general aggregation pattern given in [2]. Together, they have an equivalent equation based on the weighted power mean.

5 Conclusions

Grid systems require a mapping functionality for the association of users requests expressed in terms of requirements and preferences to actual resources. Our work started by proposing a model for the evaluation of the satisfaction perceived by a potential user for a set of services. In this paper, we have presented a mapping of such a model to a language for querying XML-based representations of available resources. This language provides a bi-directional mapping with our model, thus offering a tailored solution for its application. Future work will be targeted at defining a method for rewriting XMatch in terms of XQuery in order to exploit the available XQuery processor implementations. Another important aspect is to remove the requirement concerning the structure of the input data.

References

1. Németh, Z., Sunderam, V.: Characterizing Grids: Attributes, Definitions, and Formalisms. Journal of Grid Computing **1** (2003) 9–23
2. Andreozzi, S., Ciancarini, P., Montesi, D., Moretti, R.: An approach to the quantitative evaluation of grid services. To appear in Journal of Concurrency and Computation: Practice and Experience (2005)
3. Boag, S., Chamberlin, D., Fernndez, M., Florescu, D., Robie, J., Simeon, J.: XQuery 1.0: An XML Query Language. (W3C Working Draft, 11 February 2005)
4. Fenton, N., Pfleeger, S.L.: Software Metrics: a Rigorous and Practical Approach, 2nd edition. Course Technology (1998)
5. Dujmovic, J.: A method for evaluation and selection of complex hardware and software systems. In: Proceedings of the International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems (CMG96), San Diego, CA, USA, Dec 1996. Volume 1. (1996) 368–378
6. Andreozzi, S., Ciuffoletti, A., Ghiselli, A., Vistoli, C.: Monitoring the Connectivity of a Grid. In: Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 2004), Toronto, Canada, October 2004. (2004)
7. Prelz, F. et Al.: Pratical Approaches to Grid Workload and Resource Management in the EGEE Project. In: Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2004), Interlaken, Switzerland. (2004)
8. Kießling, W.: Foundations of Preferences in Database Systems. In *Proceedings of the 28th Very Large Database System (VLDB) Conference, Hong Kong, China*, 2002.
9. Hafenrichter, B., Kießling, K.: Optimization of Relational Preference Queries. In *Proceedings of the 16th Australasian Database (ADB05) Conference, Newcastle, Australia*, 2005.
10. Kießling, K., Köstler, G.: Preference SQL - Design, Implementation, Experiences. In *Proceedings of 28th International Conference on Very Large Databases (VLDB), Hong Kong, China*, Aug 2002.
11. Kießling, W., Hafenrichter, B., Fischer, S., Holland, S.: Preference XPATH: a Query Language for E-Commerce. In *Proceedings of 5th Internationale Tagung Wirtschaftsinformatik, Augsburg, Germany*, Sep 2001.
12. Chomicki, J.: Preference Formulas in Relational Queries. *ACM Transaction on Database Systems*, 28(4):427–466, 2003.
13. Solomon, M.: The ClassAd language reference manual. Computer Sciences Department, University of Wisconsin, Madison, WI, Oct 2003.
14. Litzkow, M. J., Livny, M., Mutka, M. W.: 2003, 'Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching'. In: *Proceedings of the 12th IEEE International Symposium on High-Performance Distributed Computing (HPDC 2003), Seattle, WA, USA, June 2003.*
15. Litzkow, M., Livny, M., Mutka, M. W.: 1988, 'Condor - a Hunter of Idle Workstations'. In: *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS 1988), San Jose, CA, USA, June 1988.*
16. Anjomshoaa, A., Brisard F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: 2005, 'Job Submission Description Language (JSDL) Specification, Version 1.0'. (GGF Working Draft, 15 June 2005)
17. Andreozzi, A., Burke, S., Field, L., Fisher, S., Balazs, K., Mambelli, M., Schopf, J.M., Viljoen, M., Wilson, A.: 'GLUE Schema 1.2'. (GLUE Collaboration, Working Draft, 24 Sep 2005)

HMM: A Static Mapping Algorithm to Map Parallel Applications on Grids

Ranieri Baraglia[1], Renato Ferrini[1], and Pierluigi Ritrovato[2]

[1] ISTI - Institute of the Italian National Research Council
{ranieri.baraglia, renato.ferrini}@isti.cnr.it
[2] CRMPA - University of Salerno
ritrovato@crmpa.unisa.it

Abstract. In this paper we present a static mapping heuristic, called Heterogeneous Multi-phase Mapping (HMM), which allows a suboptimal mapping of a parallel program onto a metacomputer to minimize the program execution time. HMM allocates parallel tasks by exploiting the information embedded in the parallelism forms used to implement an application. Moreover, it uses a local search technique together with the tabu search meta-heuristic. The experimental results show that the proposed approach performs well promising a significant potential to develop efficient mapping solutions for metacomputers.

1 Introduction

Heterogeneous Computing (HC) consists in the use of machines with different capabilities, connected by high-speed networks, as a single heterogeneous distributed system [1]. HC is also known as *Metacomputing* [2] or as *Grid Computing* [3]. To exploit the full potential of a metacomputing environment an application has to be decomposed into components, and each component has to be allocated on a machine most suitable for its execution. To efficiently assign application tasks to computing resources it is an important goal to minimize the execution time of an application running on a HC system. The general problem of mapping tasks to machines is NP-complete, and mapping algorithms are usually classified as static or dynamic [4]. This paper deals with static graph-based mapping algorithm. In its general form static algorithms model a parallel application by a directed acyclic graph (DAG) in which nodes and edges represent application tasks and intertask data dependencies, respectively. Moreover, it is assumed that task communication and computation costs, and task dependencies are known before the task execution.

In the past several static graph-based mapping heuristics for HC environments have been developed and evaluated to approximate optimal solution. In [5] an augmented Cluster-M mapping algorithm is described. It is an extension of Cluster-M which is suitable for heterogeneous computing. It takes into account both the decomposition of a parallel application into a number of tasks with different forms of parallelism, and also the different types of parallelism available in a heterogeneous architecture. Its time complexity is $O(MP)$,

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 731–740, 2006.

where M is the number of tasks belonging to an application graph, and $P = max(M, N)$, with N numbers of processors belonging to a metacomputer graph.

A Max-Flow Min Cut algorithm was proposed in [6]. It finds mappings of task modules on heterogeneous processors in $O(M^2N|E_p|logM)$ time, where M, N and E_p are the number of task modules, processors and communication links among the processors, respectively. This algorithm presents two main drawbacks: it has a high time complexity and it does not manage the task data dependencies.

In [7] C. Shen and W. Tsai propose the A^* Searching algorithm which is a Branch & Bound algorithm. Therefore, due to the complexity of these algorithms, A^* Searching algorithm is suitable for small problems.

In [8] a modified Levelized-Min Time (LMT) Algorithm based on the list scheduling heuristic is proposed. The algorithm is structured according to two phases. The time complexity of LMT $O(n^2 \times p^2)$ where n and p are the number of tasks and the number of processors, respectively.

In [9] Haluk Topcuoglu *et al.* propose a list scheduling-based algorithm, called Heterogeneous Earliest-Finish-Time (HEFT) to map parallel applications, modeled by DAGs, on a bounded number of heterogeneous processors. HEFT has time complexity equal to $O(e \times p)$ (e number of edges and p number of processors). For dense application graphs, where the number of edges is proportional to $O(n^2)$, with n is the number of the tasks, HEFT time complexity is equal to $O(n^2 \times p)$.

In this paper we describe Heterogeneous Multi-phase Mapping (HMM), a static mapping algorithm, which finds a suboptimal mapping solution of a parallel program onto a metacomputer by minimizing the program execution time. HMM allocates parallel tasks by exploiting the information embedded in the parallelism forms used to implement an application, and considering an *affinity* code-machine parameter [5] identifying which machine in the metacomputer is most suitable for executing a component of a parallel application. Moreover, it uses a local search technique together with the tabu search meta-heuristic [10]. Most parallel applications are designed using parallelism forms ($farm$, $pipeline$, $geometric$, etc.) [11] which can be used by adopting unrestricted programming model [12] and also restricted programming model [13]. The use of parallelism forms allows the implementation of parallel applications in which the communication patterns are structured and well defined. The tasks inside a parallelism form can be seen as naturally grouped in a cluster that can be allocated on processors of the same machine to reduce task communication costs. Each cluster can embed one or more nested parallelism forms to be handled in a top-down manner. This permits us to find the size of the application grain that better exploits the computation and communication characteristics of the target metacomputer.

This paper is organized as follows. Section 2 sets out the problem, and Section 3 describes our algorithm. Section 4 outlines and evaluates the experimental tests. Finally, we summarize our work in Section 5.

2 Description of the Problem

A parallel application, composed by N parallel tasks, is represented with a DAG denoted as AG. It is assumed that task computation and communication costs are known before the application execution. These costs can be obtained by estimation using source code profiling information [14].

Each AG's node t_i, with $0 \leq i \leq (N-1)$, represents an *atomic node* (a sequential module) or a *hierarchical node* (a parallel module). Atomic nodes are indivisible units of execution and they have an associated weight representing its computational cost. Hierarchical nodes are set of atomic and/or hierarchical nodes represented with a DAG as well. The structure of a hierarchical node respects the order in which the parallelism forms are used in a parallel program. Parallelism forms can be nested each other. From the task mapping point of view a hierarchical node is considered to be atomic. Each edge of AG denotes the precedence relations between parent and child nodes, and it has an associated weight representing the amount of data exchanged between them.

An application DAG is considered structured in S *layers* representing the depth of nested levels. The DAG representing a layer is denoted as AG_s with $1 \leq s \leq S$. In the first layer AG_s consists of only a hierarchical node which is considered to be atomic, in the next layers it could consist of atomic and/or hierarchical nodes, and in the last layer all nodes are atomic. In Fig. 1 an example of a three-layer expansion of a hierarchical node is given. The first layer consists of one hierarchical node including all the nodes; the second layer is represented by a task graph of four nodes, two atomic and two hierarchical (a pipeline and a task-farm), and in the last layer the task graph includes ten atomic nodes. An atomic and/or hierarchical node belonging to a graph AG_s is identified by i^s with $1 \leq i \leq N_s$ where N_s is the number of nodes at layer s.

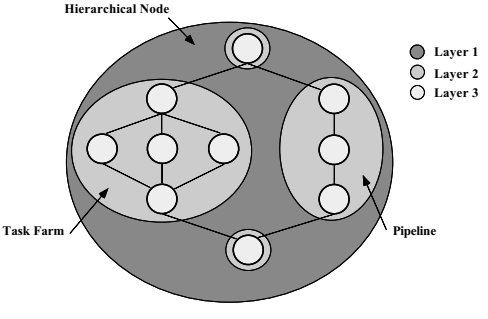


Fig. 1. AG graph representing a parallel application

Managing the layers in top-down order permits us to find the layer at which the application grain size that better exploits the computation and communication characteristics of the target metacomputer corresponds to. The mapping of this layer guarantees the shortest execution times of the nodes in it contained. To reduce the mapping complexity, AG_s are horizontally divided into F levels, called

phases, which are executed sequentially (top-down). The nodes in the first phase do not have parents, and the nodes in phase f, with $2 \leq f \leq F$, are those that have at least one parent among the nodes in phase k with $1 \leq k \leq f-1$. Structuring AG_s into phases the mapping problem is simplified because HMM finds the initial suboptimal mapping solution by operating on each phase separately. The initial problem is thus decomposed into more manageable subproblems. A node in a phase is executed when all its predecessors have been completed and it has received all the data needed for its execution. The node in a phase can be executed in parallel.

A metacomputer is represented by a fully connected indirect graph denoted as MG. Each node of MG represents a metacomputer machine which is denoted as m with $1 \leq m \leq M$, where M is the number of machines in the metacomputer. To each node is associated a number which specifies the machine computational power, and each edge, representing a link between two machines, has an associated value which represents its communication bandwidth. To identify which machine in the metacomputer is most suitable for executing a task of a parallel application an *affinity* parameter was used. The *affinity* parameter is useful to characterize both applications and metacomputer machines by relating them to some real-world parameters. In this work, without loss of generality, we associate to each node the computational model adopted for its implementation (e.g. SIMD, MIMD, sequential), and the amount of memory and software required for its execution. The same parameters are used to characterize a hierarchical node. A hierarchical node has also associated the number of processors needed to run its atomic nodes in parallel. We assume that these parameters can be specified by the programmer and/or provided by the programming tools used to implement a parallel application.

3 Algorithm Architecture

The HMM algorithm is structured according to three main steps: Initial setting, Estimation of the initial mapping solution S_{init}, and Refinement of S_{init}.

Initial setting. In order to find the best machine to execute a node, the affinity code-machine parameter $Aff(i^s, m)$ is computed for all the i^s nodes belonging to each layer with respect to all the M machines. For each node, $Aff(i^s, m)$ is computed by considering both the required computational resources and the adopted computational model. When a machine m does not provide the memory and/or software requirements as well as the computational model to run a node i^s, $Aff(i^s, m)$ is set equal to 0. Otherwise, it assumes a value in the range $0 \div 1$ computed as ratio between the number of the processors available on m and the number of the processors required to run i^s. The algorithm ends if at least one atomic node has affinity equal to 0 with all the metacomputer machines.

Estimation of the initial mapping solution. To do this a AG_s is divided into F phases. All nodes in a phase f of a AG_s are sorted in descending order with respect to their computational workload. Then, according to this order a

node i^s is selected and applying formula (1) the most suitable machine for its execution is found. In (1) W_m is the workload due to the previous allocated tasks, C_{i^s} is the computational workload of i^s and S_m is the computational power of m. $D_c(i^s, \bar{i}^s)$ is the communication cost of i^s with $\bar{i}^s$ (parent with the highest communication cost) belonging to its backward star $BS(i^s)$. $D_c(i^s, \bar{i}^s)$ is zero if the nodes i^s and $\bar{i}^s$ can be allocated on the same machine, otherwise it is equal to the amount of data exchanged between i^s and $\bar{i}^s$. $B_{m,n}$ is the bandwidth of the communication link connecting the machines m and n selected to allocate i^s and $\bar{i}^s$, respectively.

$$\min\left(\frac{W_m + C_{i^s}}{Aff(i^s, m) \times S_m} + \max\left\{\frac{D_c(i^s, \bar{i}^s)}{B_{m,n}}\right\}\right) \tag{1}$$
$$m \in \{1, 2, .., M_{Aff}\} \; and \; n \in \{1, 2, .., M\} \; and \; \forall \, \bar{i}^s \in BS(i^s)$$

The machine which the minimum value computed by (1) corresponds to is selected to allocate i^s. When the minimum value corresponds to several machines, HMM selects the machine m which the smallest value of the ratio $\frac{W_m}{S_m}$ corresponds to. If no machine is found, the node i^s is flagged "not allocated", and the next node is analyzed.

On the selected machine the execution time of i^s is computed according to the expression (2), where $T_w(i^s, m)$ and $T_x(i^s, m)$ are the wait before executing and the execution time of i^s on m, respectively.

$$T_e(i^s, m) = T_w(i^s, m) + T_x(i^s, m) \tag{2}$$

If i^s is a hierarchical node, to compute its execution time the equation (2) is recursively applied to all the atomic nodes in it contained. When the list of nodes to analyze is empty, it is checked if there are some nodes flagged "not allocated". If there are, the search for the most suitable machine to allocate a not yet allocated node starts from the first layer for which there is a machine that can execute each task belonging to it.

Elaborated all the nodes the process ends by carrying out the initial partial solution of the phase analyzed, and the execution continues to elaborate the next phase. All initial partial mapping solutions are analyzed to find the layer completion time which corresponds to the last AG_s node ending its execution. The layer completion time is then compared with the best initial mapping solution S_{init} carried out in a previous layer. The best one becomes the new S_{init}, and execution continues to analyze the next layer. When all the layers have been processed, HMM carries out the initial solution S_{init}. This solution was found by keeping the tasks communication cost as low as possible.

Refinement of S_{init}. To improve the quality of S_{init}, a local search algorithm which exploits the tabu search technique was adopted. This algorithm considers all the nodes in the S_{init} as atomic, and analyzes the neighborhood of S_{init}. We say that the neighborhood of a solution is the set of solutions obtainable by moving a node of the AG_s's critical path, to which S_{init} corresponds to, from the machine of current allocation to another machine. The new machine is selected using the following criterion.

$$\max\{Aff(i^s, m) \times S_m\} \qquad with\ m \in \{1, 2, .., M_{Aff}\}$$

If several machines are selected, the one with the smallest value of the fraction $\frac{W_m}{S_m}$ is selected. If there is no machine able to allocate i^s, the node is not moved, and the process continues by selecting the next node in the critical path. Otherwise, if the selected node is moved, HMM estimates the cost of a new solution. This cost represents the application execution time, and it is obtained by adding the computing and communication times of the nodes belonging to the critical path associated with the new solution. When all the nodes on the critical path have been analyzed the local search algorithm carries out the new solution Sol_{new}. If Sol_{new} is better than the initial solution, it becomes Sol_{init}.

The search for a suboptimal solution is an iterative process. HMM ends by returning the solution Sol_{best}, when one of the following conditions is verified: 1) the total number of iterations performed reaches a user-defined threshold, 2) the number of iterations performed without improving Sol_{init} reaches a user-defined threshold.

In the average case (the worst case in which the number of the layers is equal to the number of atomic nodes is rare) in which each AG_s is structured in $log_2 N + 1$ layers and each layer includes $2^{(i-1)}$ nodes with $1 \leq i \leq log_2 N + 1$, the algorithm complexity is equal to $O(N^3)$. For more details about the calculation of the algorithm complexity see [15].

4 Experimental Results

To evaluate HMM we conducted some experiments by running the algorithm on several tests. In this paper we describe three of the most interesting ones, the other ones can be find in [15]. The first test was conduced to analyze the HMM's behavior when mapping a structured application. In the second test HMM was compared with *Augmented Cluster-M Mapping* (ACM), *Lo's Max-Flow/Min-Cut* (MFMC), and *Shen and Tsai's A* Searching* (AS) algorithms [1]. Moreover, HMM was compared with an exhaustive mapping algorithm (EMA). In the third test HMM was compared with the $HeterogeneousEarliest-Finish-Time$ (HEFT) and the $Levelized-MinTime$ (LMT) algorithms when used to map an application implemented using the task farm parallelism form on clusters of workstations.

In the following figures the produced results are shown in a Gantt chart format. Each Gantt shows, for each task t_i, both the machine processor p_j on which it was allocated and the unit of time required for its execution (the numbers at the top of the chart).

Test 1. In Fig. 2(a) and 2(b) the AG and the MG graphs used in tests 1 are shown, respectively. The application contains two parallelism pattern: a farm identified by the hierarchical node t_{10} and a pipeline identified by the hierarchical node t_{11}. In this test the application was mapped onto three different HC systems by using HMM and the results were compared with those obtained by running

[1] The application DAGs and metacomputer graphs are taken from [5].

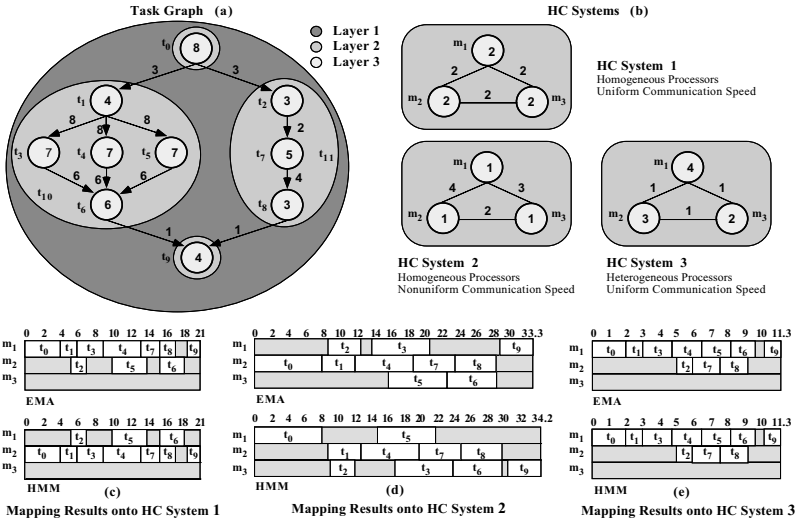


Fig. 2. Application DAG (a), metacomputer graph (b), and Gantts (c) of the test 1

EMA. In two cases HMM computed the optimal mapping solution (Fig. 2(c) and Fig. 2(e)) while in the last case (Fig. 2(d))the HMM mapping solution was worse than the EMA solution by about 2 %.

Test 2. Fig. 3(a) and 3(b) show, respectively, the application and the metacomputer graphs used in test 2. The application consisted of a MIMD code and a vector code. The metacomputer used in the test models both an MIMD and vector machines. As in [5] we performed HMM so that the MIMD code was allocated on the MIMD machine and the vector code on the vector machine. We put together in the same Gantt the execution time of both the code types, even if MFMC, AS and EMA do not treat heterogeneity in computation and machine types. Their mapping results were obtained by mapping separately the MIMD and Vector codes onto the correspondent type of machine. To map the MIMD code HMM converged to the suboptimal solution by analyzing only 21 of the 19,683 possible mappings. The HMM's suboptimal solution corresponding to a total application execution time of 20.5 units with a worsening of 3.6% with respect to the optimal mapping. The HMM and EMA executions required 0.2 and 19.75 seconds, respectively. To map the vector code, HMM converged to the suboptimal solution by analyzing only 19 of the 16,384 possible mappings. The HMM's suboptimal solution corresponding to a total application execution time of 25.17 units with a worsening of 1.6% with respect to the optimal mapping. HMM executions required 0.51 seconds, the EMA execution required 47.92 seconds. As shown in Fig. 3(c), the mapping carried out by HMM and ACM led to the same total application execution time. However, the mapping carried out by HMM reduces the execution time of the MIMD code leading to a better utilization of the MIMD machine.

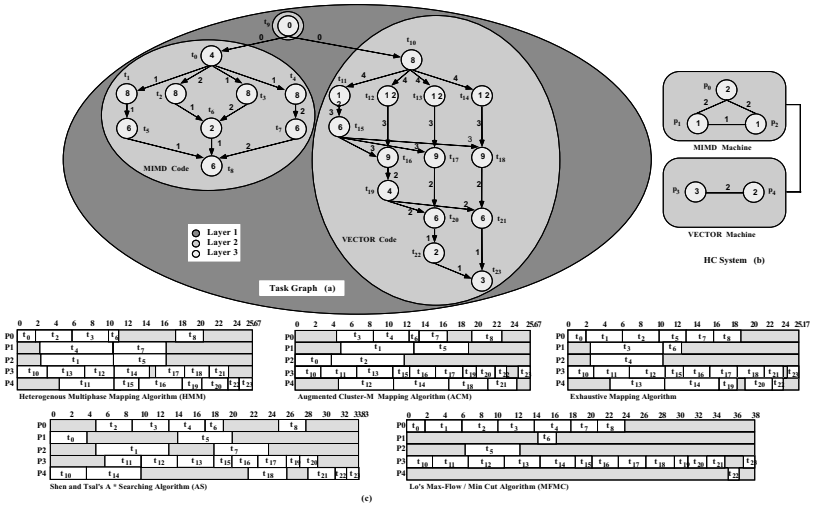


Fig. 3. Application DAG(a), metacomputer graph (b), and Gantts (c) of the test 2

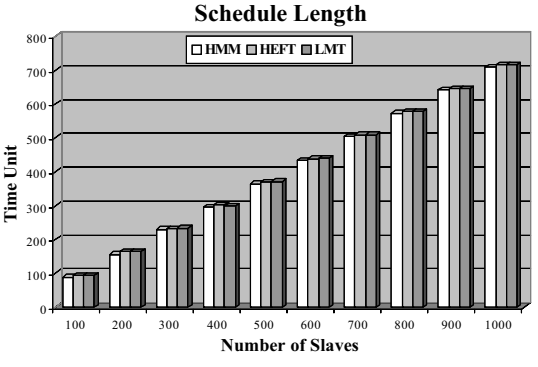


Fig. 4. Bar chart of the test 3

Test 3. This test was conducted to compare HMM, LMT and HEFT when used to map an application implemented by using the task farm parallelism form. The number of the slaves has been increased from 100 to 1000, and computation and communication costs were randomly generated varying the Communication Computation Ratio (CCR) from 0.1 to 1 with an increment step of 0.1. For each CCR value ten application graphs were generated. Therefore, several hundred graphs were mapped in each run, and the related average values were reported on the bar chart of Fig. 4. The system used in the test consisted of two clusters with the following configurations: (a) 3 workstations with processor power of 65, 20 and 20 Mflop/s connected by a 1 Mbit/s Ethernet, (b) two workstations with processor power of 25 and 15 Mflop/s connected by a 1 Mbit/s Ethernet. The clusters were considered to be connected by a 0.25 Mbit/s LAN. Although

in [9] it was showed that HEFT performs better than the Critical Path-based mapping algorithm, on this kind of applications, HMM obtained better results than both HEFT and LMT.

All the experiments were run on an HP 9000/700 workstation.

5 Conclusions

In this paper we have presented a new static mapping algorithm, which allows us to find a suboptimal mapping of a parallel application onto a metacomputer in order to minimize the application execution time.

The experimental results show that our algorithm is able to allocate the application tasks in a way that optimizes the execution of the parallelism forms used within a parallel application. The comparisons with other leading techniques show that our algorithm reaches good mapping solution. This is promising a significant potential to develop a more efficient mapping solutions for heterogeneous system.

Acknowledgements

This work was funded by the Italian Ministry of Education, University and Research (MIUR) as part of the National Project MURST 5% 1999 Grid Computing: Enabling Technology for eScience.

References

1. Freund, R. F., Siegel, H. J.: *Heterogeneous Processing*, IEEE Computer, 26(6): 13-17, 1993.
2. Cattlet, C. E., Smarr, L.: *Metacomputing*, Communication of the ACM, 53(6):45, June 1992.
3. Foster, I., Kesselman, C.: *The Grid: blueprint for new computing infrastructure*, Morgan Kaufmann Publishers, 1998.
4. ALI, H. H., El-Rewini, H., Lewis, T. G.: *Task Scheduling in Parallel and Distributed Systems*, PTR Prentice Hall, June 1994.
5. Eshaghian, M. M.: *Heterogeneous Computing*, Artech House Publishers, 1996.
6. Lo, V. M.: *Heuristic Algorithms for Task Assignment in Distributed Systems*, IEEE Transaction on Computers, 37(11), pp. 1384-1397, November 1988.
7. Shen, C., Tsai, W.: *A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minmax Criterion*, IEEE Transaction on Computers, C-34(3), pp. 197-203, March 1985.
8. M. A. Iverson, F. Ozguner, and G. J. Follen: *Parallelizing Existing Applications in a Distributed Heterogeneous Environment*, Proc. 4th IEEE Heterogeneous Computing Workshop (HCW '95), pp. 93-100, 1995.
9. H. Topcuoglu, S. Hariri, and M.-Y. Wu: *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3, March 2002.

10. Glover, F., Laguna, M.: *Tabu Search*, Kluver Academic Publishers, Boston, MA, June 1997.
11. Hey, A. J. G.: *Experiments in MIMD Parallelism*, Proceedings of Int. Conf. PARLE 89, June 1989.
12. Gropp, W., Lusk, E., Skjellum, A.: *Using MPI Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, 1999.
13. Skillicorn, D. B.: *Models for Practical Parallel Computation*, International Journal of Parallel Programming, 20(2), pp. 133-158, April 1991.
14. H. Jiang, L.N. Bhuyan, and D. Ghosal: *Approximate Analysis of Multiprocessing Task Graphs*, Proceedings of International Conference on Parallel Processing, vol. III, pp. 228-235, 1990.
15. Ranieri Baraglia, Renato Ferrini, and Pierluigi Ritrovato: *A Static Mapping Heuristics to Map Parallel Applications to Heterogeneous Computing Systems*, Concurrency: Practice and Experience, 17, pp. 1579-1605, published online, June 2005.

Agent-Based Grid Scheduling with Calana

Mathias Dalheimer[1], Franz-Josef Pfreundt[1], and Peter Merz[2]

[1] Fraunhofer Institut für Techno-und Wirtschaftsmathematik, Kaiserslautern, Germany
{dalheimer, pfreundt}@itwm.fhg.de
[2] Department of Computer Science, University Kaiserslautern, Germany
peter.merz@ieee.org

Abstract. Grid resource allocation is a complex task that is usually solved by systems relying on a centralized information system. In order to create a lightweight scheduling system, we investigated the potential of auctions for resource allocation. Each resource provider runs an agent bidding on the execution of software with respect to local restrictions. This way, the information system becomes obsolete. In addition, each provider can implement different bidding strategies in order to reflect his preferences.

1 Introduction

Grid Computing as a way of virtualizing computational resources is becoming more relevant in research and industry [1]. It can be foreseen that grid computing will become important in a commercial scenario: service providers will sell computational power and storage. Users will buy the required amount of processor cycles and disk capacity on a per-use basis.

In the *Fraunhofer Resource Grid* (http://www.fhrg.fhg.de), we have a strong bias towards commercial grid applications. Our goal is to provide users from both industry and research with a stable, general purpose grid infrastructure. As a part of this effort, we developed a prototype of Calana, an agent-based grid scheduler.

The paper is organized as follows: section 2 provides an overview of the related work. In section 3, we introduce the architecture and design principles of Calana. Section 4 discusses the results from experiments which have been done to evaluate the system. Section 5 concludes with discussing the advantages of the architecture.

2 Related Work

The problem of grid scheduling is often referred to as *resource management*. In addition to the five challenges of resource management [2], another challenge arises from the different *stakeholder's objectives*: a grid user wants to calculate fast and cheap, but a resource provider's interest is to earn money or utilize the unused computing power of his own resources [3][4].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 741–750, 2006.

Currently, many projects include the development of a grid scheduler. Globus Toolkit 4.0 integrates the Community Scheduler Framework (CSF) created by Platform Computing [5]. Other schedulers include Condor [6] and Gridbus [7], which is the successor of Nimrod-G [8]. However, the projects mentioned do not tackle two very important issues: Currently, resource information is pushed periodically in information systems like the Globus Meta Directory Service [9]. During scheduling, potentially outdated information is used by the scheduler. Furthermore, stakeholders are not able to express complex preferences and adjust their strategies quickly, which is a strong prerequisite in order to establish a computational economy [10]. In real economies, these problems are solved by the established market structures [11].

Ernemann and Yahyapour describe an architecture based on economic principles in order to solve the problem of different stakeholders' objectives [12]. They use objective functions in order to find an equilibrium of interests. Although it should be possible to implement complex strategies within this system in general, they do not discuss it. Furthermore, this system allows various forms of collusion which may not be tolerated.

Various authors have discussed the use of auctions for resource allocation problems [13][11][12]. Although Wolski and colleagues [14] prefer commodities markets over auctions, the common understanding of most authors is that auctions are capable of solving a multicriteria resource allocation problem. The setup of Wolski et al. is not applicable to this work: They separated processor and disk allocation, reducing the auction's reliability to allocate all needed resources.

Taking the stakeholder's objective problem into account, we believe that economic scheduling provides a suitable solution to the grid scheduling problem. As we show in the remainder of the paper, we can incorporate different preference structures and eliminate the need for a centralized information system for dynamic information.

3 The Architecture of Calana

The design goal of Calana is to provide a lightweight and flexible scheduling system. Basically, the architecture consists of two software components: The broker and the agent. Each resource runs a small software agent. The agent registers itself to the broker. When a job needs to be scheduled, the broker uses an auction to determine which resource is available. The registered agents receive a request to bid on the application execution. They need to check the feasibility of the request:

1. The auction announcement includes a link to the software's description. The agent may now compare the application's prerequisites with its resources specification. If all prerequisites are fulfilled, the job can be further considered by the agent.
2. Then, the agent checks the local resource usage directly by trying to get an advance reservation for the job. The local resource system usually optimizes

the reservation details, e.g. by using a backfilling algorithm [15]. The result are fixed start- and end-times for the job.
3. The reservation can be used to create a bid in the auction. The process of bid creation is influenced by the provider's strategy.

After the auction ends, the broker determines which bid fulfills the user's requirements best. Each resource provider can implement own agents, enabling them to specify all kinds of strategy and integrate various local resources.

In the following sections, we will describe the scheduling process in detail.

3.1 Using Auctions for Resource Allocation

Auctions can be classified as multilateral negotiations [16]. An auction is always based on a well-defined scheme: During the bidding phase, all *bidders* submit bids to the *auctioneer*. During the transaction phase, the auctioneer uses the *scoring rule* to evaluate all bids and select the best. This makes the implementation of a software system for auctions easy [11].

There are many different auction types. Each one has certain rules for both bidding and transaction phase and a certain impact on the fairness of the market. For the prototype implementation, we use a first-price sealed-bid auction [17]: the bids are not visible to other bidders. The best bid wins the auction, and the price of the auction is also determined by the best bid. Another possible auction is the *vickrey auction* which is known to deliver pareto-optimal allocations as well as it prevents some forms of collusion [17][16][18].

In order to consider all stakeholder's preferences, resource selection must implement a *multi-criteria optimization* as defined by Kurowski et al [19]. We use multicriteria bids: a bid may contain multiple values [16]. In general, all auctions can be adopted to work with multicriteria bids: An announcement of valid properties of a bid has to be made. For the comparison of bids, the auctioneer can use a scoring rule that delivers a relative value of a bid. The bid with best relative value will win the auction.

A serious problem of auctions is collusion: a bidder may try to break auction rules in order to increase his benefit. Although a bidder may only submit bids for himself, there are ways to influence the market [18]: Pools of bidders may influence auctions, the auctioneer may trade with information in sealed-bid auctions and offerors can use phantom bids to increase the market's competition. Creating fair auctions is a complex task. Basically, the use of sealed bid auctions in combination with a trusted auctioneer seems to be reasonable [16]. Further research concerning auction design for computational markets is needed.

3.2 The Calana Prototype

An overview is given in figure 1. The broker works as an auctioneer for scheduling requests, while the agent submits bids for the auctions. The agents are located at the provider's sites, while the broker must be hosted by a trusted third party. The user accesses the broker not directly but by using a portal or a workflow tool, e.g. the GridJobHandler [20].

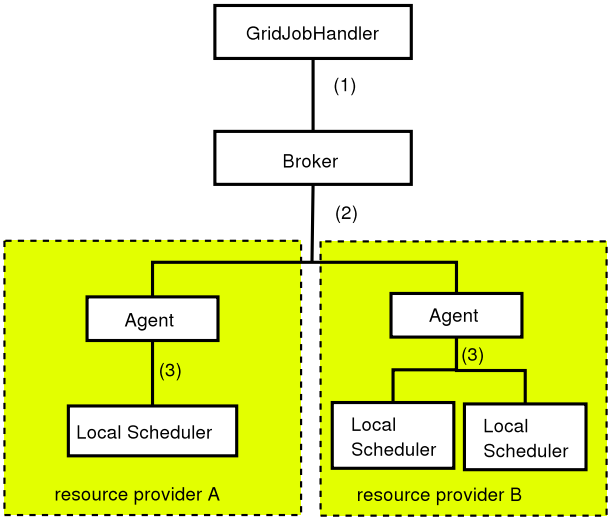


Fig. 1. Overview of the architecture: the user workflow tool uses the broker component to create a valid schedule (1). Resource providers use an agent in order to connect their local scheduling systems to the architecture (3). The broker and agents interact to create the schedules (2).

For each job, the user's tool calls the broker to retrieve a schedule. Each call consists of a description of the software, possibly the size of the datasets and some weights for the scoring function in order to reflect the user's preferences. One may also consider scoring functions completely specified by the user.

The broker receives the request and creates a new auction. A *call for bids* is sent to each registered agent. This call contains a software description, provided by the manufacturer of the software, and the auction deadline. The software description contains a description of the runtime behaviour, along with other properties such as the number of cluster nodes the software should run on. The agent can use this information to predict the overall runtime of the application. Of course, this prediction is not reliable, only for certain types of software the runtime may be calculated, e.g. depending on the input data [21][22]. As a fallback, user-specified walltimes can be used.

Based on this information, the agent composes a bid. This can be influenced by the local bidding strategy: Jobs may be cheaper on weekends or promotions during holidays may be considered. A bid consists of the estimated job finish time t^{f} and a monetary price p for the execution. When a bid is submitted, an advance reservation for the software must be made in order to be able to satisfy an auction won.

When the auction deadline has passed, the broker judges all bids by applying a scoring rule. This enables the broker to consider user preferences modeled as weights for the scoring rule. For the value v of a given bid, the broker calculates

the scoring rule with the parameters given by the users. For example, the value of the bid i, $0 \leq i \leq n$, may be calculated as follows:

$$v_i = f(p_i, t_i^{\mathrm{f}}) = g \cdot \frac{p_i}{p_{max}} + (1 - g) \cdot \frac{t_i^{\mathrm{f}}}{t_{max}^{\mathrm{f}}} \tag{1}$$

with a weight g for the price preference, $0 \leq g \leq 1 \in \mathbb{R}$, maximum finish time $t_{max}^{\mathrm{f}} = \max\{t_i^{\mathrm{f}} | 0 \leq i \leq n\}$ and maximum price $p_{max} = \max\{p_i | 0 \leq i \leq n\}$. The best bid b is the bid with the minimal value:

$$b = \min\{v_i | 0 \leq i \leq n\} \tag{2}$$

This way, a user has the opportunity to express even fuzzy personal preferences, like "I want my results fast, but I don't want to pay that much." In this example, one may choose $g = 0.2$. The scoring rule can be generalized in order to consider more variables or to respect user thresholds like "I prefer fast execution, but it should not cost more than €11.2".

Finally, after a scheduling decision has been made, the broker notifies the user's tool, which may use the advance reservation to execute the job. In parallel, all agents that haven't succeeded will be notified to cancel the advance reservation. If a notification is not received, the agents drop reservation after a timeout. Since all transactions are made by a central broker, an accounting service may also be provided.

The providers register themselves at the broker. There must be an announcement of available brokers, but since their number is comparatively low, this should not have negative effects. The auction announcement is broadcasted to all registered agents. Only the interested agents answer, and only these will receive a message about the auction result. Agents may also choose to resend a bid, e.g. if they didn't win an auction at another broker. When a sealed auction is used, the current bid must not be broadcasted to all agents. If n agents are attached, we expect the number of messages to be propotional to n. Compared to a system with a central information system, more messages are needed during scheduling.

But when comparing to other schedulers, the periodical messages to update the dynamic information in centralized information system are often not taken into account. In this architecture, no centralized dynamic information is needed: since the agents reside beneath the resources, they can directly access the batch queues, getting up-to-date information.

The amount of messages can be reduced further by the introduction of a tree-like hub structure: The broker announces all auctions to its hub peers, which have the resources attached. They propagate the auction request to the resources and collect the bids. Finally, results are handed back to the broker. This infrastructure can also be used to reduce information: for example, a hub peer might choose only to propagate the pareto-dominant bids and drop the rest. This way, the messages of the broker are proportional to $\log n$. Note that all hubs must belong to a trustworthy institution in order to avoid collusion.

4 Experiments

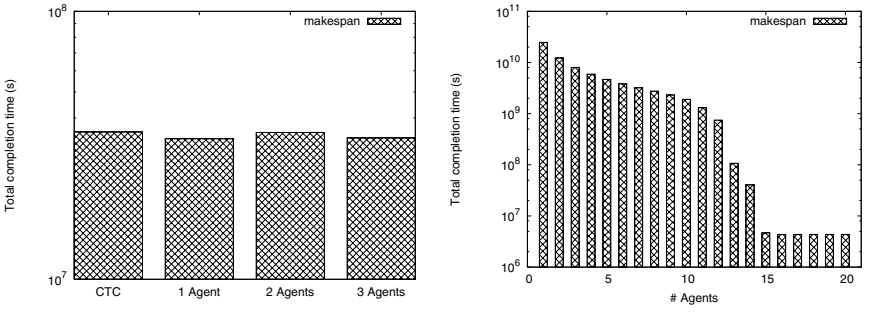
Unfortunately, there is no common benchmark for grid schedulers so far. Lacking a common model for workload creation [23], we used parallel workload sets from the Cornell Theory Center (CTC)[24] and created our own job sets for non-parallel workloads. A model for grid resources has been proposed by Kee and colleagues [25]. In future work, we will use this information to define and run further experiments. In addition to our prototype, we developed an event-based simulation with Gridsim [26] which we use to run most experiments. We denote if we used the prototype. We use two measurements for the evaluation: The total completion time and the overall price. The total completion time C provides the time needed to run n jobs in total, including queuetime and runtime of each individual job:

$$C = \sum_{i=1}^{n} (queuetime_i + runtime_i) \tag{3}$$

The overall price is the sum of the prices of all single software executions.

4.1 Comparison to Other Schedulers

The lack of a common benchmark makes the comparison of schedulers difficult. Since the CTC workload is freely available and delivers the decisions of a centralized scheduler based on backfilling, we can directly compare our parallel simulation runs, see figure 2a: We compared the allocation of the CTC scheduler with three runs of the prototype. In the first run, one agent was responsible for a cluster as big as the 420 node CTC cluster. In the second and third run, we added agents that managed one (114 node) resp. two smaller (56 node) clusters in addition to a cluster with 306 nodes managed by the first agent. This is necessary because the biggest job in the CTC's workload needs 306 nodes. As the



(a) Total completion time comparison (b) Scalability for nonparallel jobs

Fig. 2. The left figure shows a comparison of the total completion time for the CTC scheduler and Calana, on the right side results of our non-parallel workload test are shown

figure shows, the runtimes are almost the same, independent of the number of attached agents. Therefore, we do not expect centralized scheduling systems to perform better in general.

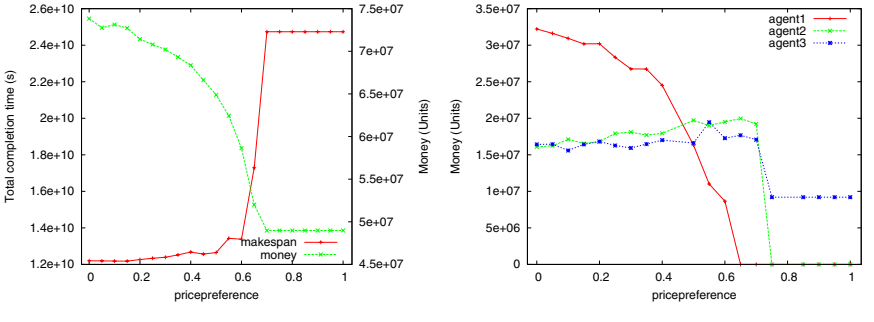
4.2 Scalability of the Architecture

Our prototype was able to handle 500 concurrent agents, bidding on parallel jobs. Since the EGEE project has about 150 sites, we would be able to schedule more than three times the EGEE project with a single broker. Lacking a model to create appropriate workload, we used the CTC workload as input. Of course, the workload is too low to utilize this number of sites, so the queuetime was zero for all runs. For the following experiments, we used a synthetic workload which contains nonparallel jobs. We simulated successively 1 to 20 agents, see figure 2b. Since each agent manages a non-parallel resource, the total completion time for the workload decreases continuously until all jobs can be processed instantly.

4.3 Influence of User Preferences and Provider Strategies

In our basic model, we assume each job has a fixed setup price p^s and a constant price p^t for each CPU-second. The price p of a computation running f^{t} seconds can be calculated as $p = p^s + f^{\mathrm{t}} \cdot p^t$. A provider may now choose different values of p^s and p^t. At the same time, the user wants to specify his own preferences. In order to show Calana's ability to deal with this, we ran an experiment with two agents.

The first agent uses a setup price p_1^s which is half of the setup price of the second agent ($2p_1^s = p_2^s$), but the computation cost is more expensive: the price per CPU-second p_1^t is twice times of the second ($p_1^t = 2p_2^t$). As figure 3a shows, the overall price drops while the total completion time increases when the user's preference changes to cheaper execution. When the price preference reaches 70 % ($g \geq 0.7$), the values doesn't change any more: The price dominates the broker's decisions.



(a) Influence of user preferences (b) Influence of exponential smoothing

Fig. 3. Influence of user preferences and bidding strategies on the allocation. When the price component of the bids become more important, the behaviour of Calana changes.

Of course, this experiment raises the question how a resource provider can adjust its agent to the market. We used the settings above to evaluate a third agent that generates p_3^t based on the last auctions: It uses exponential smoothing ($\alpha = 0.15$) to wheight the last 10 winning bid's amount $\hat{p}$ and uses this value to create a bid:

$$p_3^t = \alpha \sum_{i=0}^{10} (1-\alpha)^i \cdot \hat{p}_i \tag{4}$$

As shown in figure 3b, the third agent dominates the scenario completely if the price influence becomes important.

5 Summary

In this paper, we presented an auction-based grid scheduler that is capable of taking both user and resource provider preferences into account. No central information system for dynamic data is necessary, providing a major advantage compared to other schedulers. A provider may implement all kind of strategies and change them frequently. By creating a bid, it is possible to reflect local restrictions. User preferences can be taken into account by using multicriteria bids.

The architecture is scalable, portable and extensible. In general, the software can be adopted to fit other environments. The architecture will be developed further so that it will be usable with other grid setups. In future, the architecture will be evaluated in a real industry setup and enhanced to include more market aspects. An evaluation of different agent strategies will be made, along with a production-ready implementation in the Fraunhofer Resource Grid.

Acknowledgement

The Competence Center High Performance Computing at the Fraunhofer ITWM (`http://www.itwm.fhg.de`) is supporting this work.

References

1. Foster, I., Kesselman, C.: Computational Grids. In: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1998)
2. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A resource management architecture for metacomputing systems. In: Job Scheduling Strategies for Parallel Processing. Volume 1459 of LNCS., Springer (1998) 62–68
3. Moreno, R., Alonso-Conde, A.B.: Job scheduling and resource management techniques in economic grid environments. In et al., F.F.R., ed.: Across Grids 2003. Volume 2970 of LNCS., Springer (2004) 25–32
4. Ernemann, C., Hamscher, V., Yahyapour, R.: Economic scheduling in grid computing. In: Job Scheduling Strategies for Parallel Processing. Volume 2537 of LNCS., Springer (2002) 128–152

5. Smith, C.: Open source metascheduling for virtual organizations with the community scheduler framework (csf). Technical report, Platform Computing, Inc. (2003)
6. Roy, A., Livny, M.: Condor and preemptive resume scheduling. In Nabrzyski, J., Schopf, J.M., Weglarz, J., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
7. Venugopal, S., Buyya, R., Winton, L.: A grid service broker for scheduling distributed data-oriented applications on global grids. Technical Report GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia (2004)
8. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Future Generation Computer Systems **18** (2002) 1061–1074
9. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press (2001)
10. Kenyon, C., Cheliotis, G.: Grid resource commercialization. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
11. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) **14** (2002) 1507–1542
12. Ernemann, C., Yahyapour, R.: Applying economic scheduling methods to grid environments. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
13. Wellman, M.P., Walsh, W.E., Wurmann, P.R., MacKie-Mason, J.K.: Auction protocols for decentralized scheduling. In: 18th International Conference on Distributed Computing Systems, Amsterdam. (1999) Revised and extended version of "Some economics of market-based distributed scheduling".
14. Wolski, R., Plank, J.S., Brevik, J., Bryan, T.: Analyzing market-based resource allocation strategies for the computational grid. International Journal of High Performance Computing Applications **15**(3) (2001) 258–281
15. Lifka, D.A.: The ANL/IBM SP scheduling system. In Feitelson, D.G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Number 949 in LNCS, Springer (1995) 295–303
16. Peters, R.: Elektronische Märkte - Spieltheoretische Konzeption und agentenorientierte Realisierung. Physica Verlag (2002)
17. Schimmel, K., Zelewski, S.: Untersuchung alternativer auktionsformen hinsichtlich ihrer eignung zur koordination verteilter agenten auf elektronischen märkten. Technical Report 19, Institut für Produktionswirtschaft und industrielle Informationswirtschaft, Universität Leipzig (1996)
18. Corsten, Hans; Gössinger, R.: Auktionen zur marktlichen koordination in unternehmungsnetzwerken. In Corsten, H., ed.: Unternehmungsnetzwerke. Oldenbourg (2001)
19. Kurowski, K., Nabryski, J., Oleksiak, A., Weglarz, J.: Multicriteria aspects of grid resource management. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
20. Hoheisel, A.: User tools and languages for graph-based grid workflows, 2004. In: Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience. (2004)

21. Merten, D.: Integrated performance analysis of distributed computer systems. Workshop on Performance Characterization, Modeling and Benchmarking for Existing and Emerging HPC Systems (2004)
22. Russell, M., Allen, G., Goodale, T., Nabrzyski, J., Seidel, E.: Application requirements for resource brokering in a grid environment. In Nabrzyski, J., Schopf, J.M., eds.: Grid Resource Management. Kluwer Academic Publishers (2003)
23. Chapin, S., Cirne, W., Feitelson, D., Jones, J., Leutenegger, S., Schwiegelshohn, U., Smith, W., Talby, D.: Benchmarks and standards for the evaluation of parallel job schedulers. In Feitelson, D., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Volume 1659 of LNCS., Springer (1999) 66–89
24. Feitelson, D.: Parallel workloads archive (2005) `http://www.cs.huji.ac.il/labs/parallel/workload`.
25. Kee, Y.S., Casanova, H., Chien, A.A.: Realistic modeling and synthesis of resources for computational grids. In: Proceedings Supercomputing 2004, IEEE (2004)
26. Buyya, R., Murshed, M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. The Journal of Concurrency and Computation: Practice and Experience (CCPE) **14**(13-15) (2002)

Towards an Intelligent Grid Scheduling System

Ralf Gruber[1], Vincent Keller[1], Pierre Kuonen[5], Marie-Christine Sawley[4], Basile Schaeli[2], Ali Tolou[1], Marc Torruella[6], and Trach-Minh Tran[3]

[1] LIN-STI, EPFL, CH-1015 Lausanne, Switzerland
[2] LSP-I&C, EPFL, CH-1015 Lausanne, Switzerland
[3] CRPP, EPFL, CH-1015 Lausanne, Switzerland
[4] CSCS, CH-6928 Manno, Switzerland
[5] Ecole d'Ingénieurs et d'Architectes, CH-1705 Fribourg, Switzerland
[6] Universitat Politecnica Catalunya, E-08034 Barcelona, Espana

Abstract. The main objective of the Intelligent GRID Scheduling System (ISS) project is to provide a middleware infrastructure allowing a good positioning and scheduling of real life applications in a computational GRID. According to data collected on the machines in the GRID, on the behaviour of the applications, and on the performance requirements demanded by the user, a heuristic cost function is evaluated by means of which a well suited computational resource is detected and allocated to execute his application. The monitoring information collected during execution is put into a database and reused for the next resource allocation decision. In addition to providing scheduling information, the collected data allows to detect overloaded resources and to pin-point inefficient applications that could be further optimised.

1 Introduction

The development of GRID technology for computational purposes is promising. By harnessing a great number of different systems in a transparent manner, a user can have access to the computer architectures that are well suited to the constraints of his applications.

The different communication needs of applications demand a GRID that can offer different parallel computer architectures: SMP and/or NUMA machines for shared memory parallel applications, a NoW interconnected by a bus for embarassingly parallel applications, scalable but cost-effective networked clusters for applications dominated by point-to-point communications, and more expensive machines with faster networks for communication intensive applications.

There is currently little feedback about applications that are not adapted to the hardware infrastructure, and little incentive to do so: if for instance a user notices that the network is too slow and hampers the performance of its application, he may try to find another machine to run it. On the other hand, when he runs an embarassingly parallel application on a costly NUMA machine, he will probably not recognise this as a problem. In the future, one would like to choose a well suited hardware for the application, and this in a most automatic manner. The ISS project is precisely aimed at solving this latter problem. The ISS middleware will be built on top of existing GRID middleware infrastructures such as

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 751–757, 2006.

Globus [1], Unicore [2], EGEE (http://egee-intranet.web.cern.ch/egee-intranet), or GridLab [3]. The information on the behaviour of the application during execution is collected and put into a database. Scheduling decisions are then made in an automatic manner, taking constantly the monitored data into account. The accumulated data can later be interpreted statistically to recognise overloaded resources that have to be complemented. The long term goal is to determine the suitability of a platform using a more general cost function, which would not be restricted to computation costs. Other indirect costs could include for instance the waiting time of an engineer, or the licence of a commercial application.

Within this paper, we present ideas on how to parameterise the GRID hardware and the parallel applications [4], and how to use these parameters to decide on which machine a given application is to be executed. Moreover, a first statistical study on the CPU usage of the Pleiades.epfl.ch cluster is presented together with two application profiles coming from CFD and plasma physics. Such measurements will be used to automatically parameterise the applications in the next phase of the project.

2 Parameterisation of Clusters and Applications

2.1 Different Types of Machines in a GRID

Let us consider a cluster with P computational nodes, each node has a processor peak performance of R_∞ [Gflops/s], and a peak main memory bandwidth of M_∞ [Gwords/s] (1 word = 64 bits). The nodes are interconnected by a communication network with a total peak bandwidth of C_∞ [Gwords/s]. Then, one can define the following quantities

$$V_M = \frac{R_\infty}{M_\infty} \qquad (1)$$
$$V_C = P\,\frac{R_\infty}{C_\infty}.$$

These two parameters measure the number of floating point operations the processor can make during the transfer time of an operand from main memory to cache (V_M) or from one computational node to another one (V_C).

Some typical machines are listed in Table 1, with their respective parameters in Table 2. The data corresponds to machines with one (NoW, Pleiades, Horizon) or two (Mizar, Blue Gene) processors per node. Specifically, the parameter V_M distinguishes between a vector machine ($V_M \approx 1$) and a RISC processor ($V_M \approx 7$). One also sees that the quantity V_C can vary from 1 for a vector machine to 100000 or even more for a bus-based machine. The cost of a machine often increases with decreasing values of V_C.

2.2 The Γ Parameter

In the following analysis, we will assume that the tasks of a parallel application are well balanced, and that computations and communications do not overlap.

Table 1. Some typical clusters

Cluster	*Vendor*	*node*	*procs/ node*	*network 1*	*network 2*
NoW		Pentium 4	1	FE bus	
Pleiades1	Logics	Pentium 4	1	FE switch	
Pleiades2	DELL	Pentium 4	1	GbE switch	
Mizar	Dalco	Opteron	2	Myrinet	
Blue Gene	IBM	Power 4	2	Grid network	Fat Tree
Horizon	Cray	Opteron	1	3D Torus	
SX-5	NEC	vector	1	Switch	

Table 2. Characteristic parameters of some clusters

Cluster	P	R_∞ [Gflops/s]	M_∞ [Gwords/s]	V_M	C_∞ [Gwords/s]	V_C
NoW	25	6.4	0.8	8	0.0016	100000
Pleiades1	132	5.6	0.8	7	0.2	3600
Pleiades2	120	5.6	0.8	7	1.8	360
Mizar	160	9.6	1.6	6	5	300
Blue Gene	4096	8	1	8	192	170
Horizon	1100	5.2	0.8	6.5	1760	3.3
SX-5	16	8	8	1	128	1

Let assume that the execution time T on each computing node can be divided in two parts:

$$T = T_P + T_C \,, \tag{2}$$

where T_P is the time spent to compute and T_C the time spent to communicate and synchronise on each processor. Thus the speedup A of an application running on P processors can be expressed as:

$$A = \frac{PT_P}{T_C + T_P} = \frac{P}{1 + \frac{1}{\Gamma}} = eP \tag{3}$$

and e is the average CPU usage of the application or the efficiency (e=A/P). We define Γ as the ratio T_P/T_C and decompose T_P and T_C into application and hardware specific parameters. This allows to separate the two contributions:

$$\Gamma = \frac{T_P}{T_C} = \frac{O/r_a}{S/b} = \frac{O/S}{r_a/b} = \frac{\gamma_a}{\gamma_M} \,. \tag{4}$$

The quantity O denotes the number of operations per processor [flops] one has to perform during the execution of the application, and S is the amount of data (in 64-bit words) that has to be sent through the internode network by each processor [words]. The quantities b and r_a measure the peak effective bandwidth of the network for each processor [Gwords/s], and the peak performance of the application per processor [Gflops/s], respectively. If the data can be kept in

cache, the value of r_a can be close to the peak performance R_∞ of a processor, or r_a can be related to the main memory bandwidth if the data has to be continuously loaded from main memory to cache or stored back to memory. In scientific applications, r_a varies between 10% and 100% of R_∞. The smaller r_a/R_∞, the bigger Γ, and the communication needs diminish.

We thus see that Γ is a parameter which expresses how a given hardware is suitable to efficiently run a given parallel application. For instance, a value of 1 means that the application spends as much time in communications than in processing, and is equivalent to a speedup of $P/2$, or $e = 0.5$. Γ should thus be as large as possible but experience shows that a value greater than 1 (around 2 or 3) corresponds to an acceptable match between the application and the hardware. Let us describe a few cost-effective application/machine combinations with values of $\Gamma > 1$.

2.3 The Γ of the Different Application/Machine Combinations

Embarassingly parallel applications. Embarassingly parallel applications are dominated by master/slave communications. No data is exchanged between slave nodes. In this case, $T_P >> T_C$ and thus $\Gamma >> 1$. As a consequence, very high γ_M communication networks such as a bus or the Pleiades1 cluster (see Table 2) can be used. A typical example is the *seti@home* project that collects computational cycles over the Internet. Other examples are massive data interpretation as it occurs in high energy physics, or the sequencing algorithms in genomics and proteomics.

Applications with point-to-point communications. Point-to-point communications typically appear in finite element or finite volume methods when a huge 3D domain is decomposed in subdomains [5] and an explicit time stepping method or an iterative matrix solver is applied. If the number of processors grows with the problem size, and the size of a subdomain is fixed, γ_a is constant, and, consequently, Γ does not change. The per processor performance is determined by the main memory bandwidth. The number O of operations per step is directly related to the number of variables in a subdomain times the number of operations per variable, whereas the amount of data S transfered to the neighboring subdomains is directly related to the number of variables on the subdomain surface. For huge point-to-point applications using many processing nodes, $\Gamma << 1$ for a bus, $2 < \Gamma < 10$ for the Pleiades1 cluster with a Fast Ethernet switch, $10 < \Gamma < 50$ for the Pleiades2 and Mizar clusters, and $\Gamma >> 100$ for Horizon. Hence, that kind of applications can run well on a cluster with a relatively slow and cost-effective communication network.

Applications with multicast communication needs. The parallel 3D FFT algorithm is a typical example with important multicast communication needs. Here, γ_a decreases when the problem size is increased, and the communication network has to become faster. In addition, $r_a = R_\infty$ for FFT, γ_M is big, and, as a consequence, the communication paramater b must be big to satisfy $\Gamma > 1$. Such an application has been discussed in [4]. It has been showed that with a

Fast Ethernet based switched network, the communication time is several times bigger than the computing time, even when the problem size is small. Such an application needs a faster switched network such as an efficient GbE, a Myrinet, a Quadrics, or an Infiniband network. If thousands of processors are needed, a special vendor specific machine such as Horizon or Blue Gene might be required.

3 Monitoring Data from Parallel Applications

Each application has its own well suited parallel machine. This characteristic will in future be used to decide on which machine an application should be executed, which implies that the behaviour of an application has to be monitored for each run. To verify our model, data on the CPU usage was collected on the Pleiades1 cluster using the *sysstat* tool (http://perso.wanadoo.fr/sebastien.godard/). The gathering was made during the first 3 months of 2005, with snapshots being taken on each node every 10 minutes.

The top part of Fig. 1 shows the histogram of the 1682806 collected snapshots. The 10% zero CPU usage is due to non-allocated processors when the scheduler blocks resources for a large job, to resources that are reserved for interactive testing and not used, to lost cycles due to a blocking in a parallel application, or to intensive I/O operations. The 100% usage peak is mainly due to single processor applications that represent about 20% of the total CPU time.

Parallel jobs running on Pleiades1 share their time between computations and MPI and I/O communications, and use on average 10 processors. The average utilization of CPUs is 64%, with two peaks around 55%, and 82%. This can be considered as a fair score by a low-cost cluster with a Fast Ethernet switch with V_M=3600 (see Table 2).

For the application analysis, we chose two user applications that consumed 17% and 9% of the total computing time during the considered period. Fig. 1 shows the distribution of CPU usage for one run of each application. The first application (middle of Fig. 1) comes from fluid dynamics. It used 32 processors and ran for 5570 minutes, leading to a profiling with 17824 (=557*32) snapshots. About 10% of the snapshots show a CPU usage of 0%, and 15% show a 100% usage. This application shows an average CPU usage of e=0.56, i.e. following eq. 3 a Γ of 1.27. It could run more efficiently on a machine with a better internode communication system, but we would need to determine whether the price/performance ratio would improve when going on a more expensive machine.

The second application (bottom graph of Fig. 1) comes from plasma physics. It also used 32 processors and ran for 1690 minutes, giving 5408 (=169*32) snapshots. Processors were idle for about 15% of the time. The efficiency was 75.5%, i.e. $\Gamma = 3.1$. This is a typical application that contributes to the peak around 82% CPU usage in the upper graph. The Pleiades1 cluster seems to be a well-suited machine for this application.

In the future, we shall need characteristic CPU usage profiles such as those given in Fig. 1 for each parallel application and machine in a GRID. Together

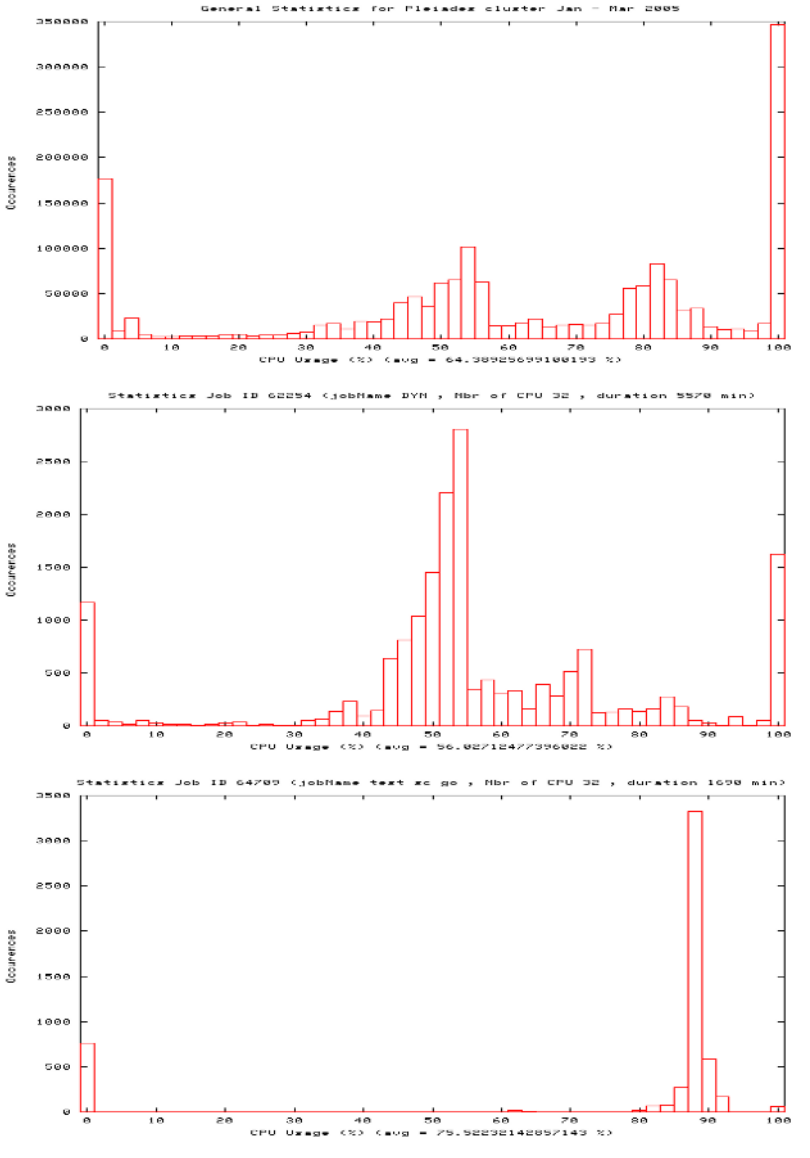


Fig. 1. Up: CPU usage of all the 132 processors of the Pleiades1 cluster (V_M=3600) during the first 3 months in 2005. Average CPU usage was collected for each processor every 10'. The overall average CPU usage is 64%. Center: Profile of one job of a CFD application. Low: Profile of one job of a plasma physics application.

with additional data on the behaviour of the applications, coming for instance from accounting data collected by the resource management system, we believe to be able to automatically parameterise them. We can then define a cost function

based on these parameters through which it will be possible to decide on where to submit each application.

We have to mention that the zero CPU usage peak of the upper graph in Fig. 1 aggregates contributions from different sources: although I/O is the most frequent one, MPI message passing and idle processors in unbalanced jobs must be taken into account as well. In pathological cases, one task of a parallel job dies, and the other processors remain idle until the job is killed by the scheduling system.

These first results show that improvements must be made: the Γ model must include I/O, and being able to distinguish between the sources of inefficiencies would be most welcome. Monitoring already had a positive impact: badly behaving applications have already been detected and improved.

4 Conclusions

The goal of the ISS project is to make it possible to automatically detect the type of hardware that is well suited for a given application. We described a parameterisation of parallel machines and applications that allows to tailor a computational GRID to a set of applications, and checked the validity of the parameters against real executions. Although it is in its initial stage, the ISS project already has an interesting side-effect: badly behaving applications can already be detected and improved. Future work includes extending the current model to correct the weaknesses that were described, and using the data collected during job execution to improve the scheduling decision. This will enable us to constantly adjust a computational GRID to the needs of the applications.

Acknowledgements

ISS is part of the the SwissGrid programme managed by the CSCS. This paper is a co-operative work within the CoreGRID Network of Excellence.

References

1. Foster, I., Kesselman (Eds.), C.: The GRID Blueprint for a new Computing Infrastructure. Morgan Kaufman, San Francisco (1999)
2. Erwin, D.: UNICORE plus final report – uniform interface to computing resource. Forschungszentrum Juelich (2003)
3. Seidel, E., Allen, G., Merzky, A., Nabrzyski, J.: Gridlab–a grid application toolkit and testbed. Future Generation Computer Systems **18** (2002) 1143–1153
4. Gruber, R., Volgers, P., De Vita, A., Stengel, M., Tran, T.M.: Parameterisation to tailor commodity clusters to applications. Future Generation Computer Systems **19** (2003) 111–120
5. Gruber, R., Tran, T.M.: Parameterisation to tailor commodity clusters to applications. EPFL Supercomputing Review **14** (2004) 12–17

Running Interactive Jobs in the Grid Environment

Marcin Lawenda, Marcin Okoń, Ariel Oleksiak, Bogdan Ludwiczak, Tomasz Piontek, Juliusz Pukacki, Norbert Meyer, Jarosław Nabrzyski, and Maciej Stroiński

Poznań Supercomputing and Networking Center,
ul. Z.Noskowskiego 10, 61-704 Poznań, Poland
Marcin.Lawenda@man.poznan.pl
http://www.psnc.pl/

Abstract. In this work authors discuss the various aspects and problems connected with interactive jobs' management in the grid environment. The specific nature of scheduling the interactive experiments is closely analyzed, revealing the complex approach that has to be taken into account in order to consider many new, scientific-domain scheduling factors. Diagrams for submission, launching, prolonging and ending the interactive sessions are proposed. Presented idea is generic and can be used in many grid-based systems. The grid broker e.g. Grid Resource Management System (GRMS) is required in our conception and we assume its presence in the environment.

1 Introduction

There are two main kinds of jobs in the grid systems: regular (batch) jobs and interactive/visualization tasks (the ones performed in the real time, directly by the users - via the GUI). The main difference (and difficulty) between those types is that - in the interactive jobs - the time slot reserved for running a job on a computational machine must be synchronized with user preferences, considering specific work hours, daily schedule etc. Moreover, there are also maintenance periods, when a resource is unavailable.

Middleware systems for interactive jobs' invocation in the grid-based systems are in the initial state. It is very hard to find general solution for the wide spectrum of applications due to different users and software requirements [1].

In the paper general conception of interactive jobs invocation is presented. It is assumed that the Grid Resource Management System (GRMS) [4] is present in the computing environment. The VNC Manager and the VNC server are installed on every machine. Moreover, the *Interactive Applications Manager* responsible for managing information about running interactive jobs is also given. All services rely on Globus.

Presented solution can be used by many client applications. The Virtual Laboratory (VLab) [6] system can be an exemplary client of this solution. Majority of applications used in virtual labs are interactive ones. There are used for

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 758–765, 2006.

performing experiment, controlling lab devices, analyzing postprocessing data, etc.

The VLab is not a standalone system. It was designed to cooperate with many other grid systems, providing only the purpose-specific functionality and relaying on well known and reliable grid solutions. The most important system the VLab cooperates with is the Globus Toolkit - in the scope of scheduling computational tasks, software services and libraries for resource monitoring, discovery, and management. All computational tasks submitted in the VLab system are transferred to the Globus via the GRMS module. Among other external systems used by the Virtual Laboratory are: the VNC system, DMS (Data Management System) [7], Authentication module and GAS [5] authorization system.

2 Grid Resource Management System

2.1 Overview

GRMS is an open source meta-scheduling system for large scale distributed computing infrastructures [3]. Based on the dynamic resource selection, mapping and advanced grid scheduling methodologies, it has been tailored to deal with job and resource management challenges in Grid environments, i.e. load-balancing among clusters, setting up execution environments before and after job execution, remote job submission and control, file staging, and more. GRMS was developed entirely in Java and thus can be installed on various kinds of operating systems and resources. GRMS is infrastructure independent and can be easily integrated with various Globus versions. In the described installation GRMS works with Globus Toolkit 2.2.4 and uses its services deployed on resources. GRMS provides job and resource management mechanisms on the top of Globus Core Services. In particular, GRMS uses GRAM, GridFTP and GRIS/GIIS services. One of the main assumptions for GRMS is to perform remote jobs control and management in the way that satisfies Users (Job Owners) and their applications requirements [2].

The main GRMS functionality includes: queuing submitted job, finding the best resource, staging in/out files, submitting job to computational resource, job migration, job canceling, logging.

Fig. 1 shows a more detailed view of GridLab GRMS with all its main modules and the GridLab specific services, like *Replica Management*, *File Movement* and *Adaptive Components*.

As it is shown on fig. 1. GRMS consists of a set of modules. These include:

- **Broker Module** - responsible for job submission control,
- **Resource Discovery Module** - responsible for querying available information services,
- **Job Manager Module** - monitors the status of a running job,
- **Job Queue** - maintains the main GRMS queue,
- **Job Registry** is responsible for maintaining the database of all jobs submitted to GRMS and all information concerning those jobs.

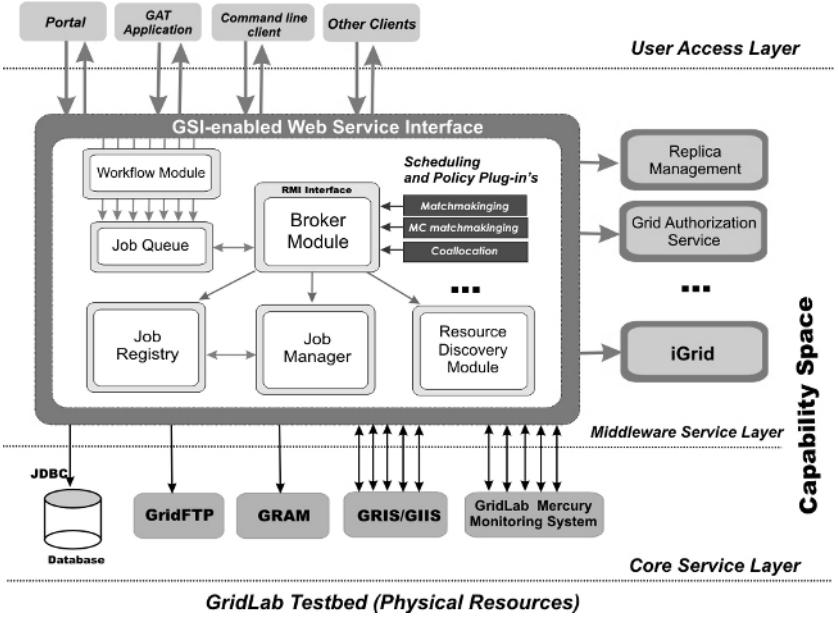


Fig. 1. Detailed view of GridLab GRMS

2.2 Time Slot Reservation

All information related to time requirements of interactive jobs is passed to the system during the job submission process as a part of job description. Every job to be submitted can have an optional section that defines in a formal way the time requirements for the job to be computed. This gives a user the possibility to build descriptions of advanced execution schedule in a simple and flexible way. The “execution time" section consists of three subsections defining following requirements: optional slot within the day when a job must be executed, mandatory execution time and optional time period when a job must be executed. The slot within the day is specified by start time of the slot and optionally end time of it or time duration. Specifying this time slot a user can determine that the job must be started after some time and not later then some other time of day, for example that its execution must be started between 10AM and 12AM. Mandatory information concerning the duration of the job execution determines length of the period when a resource reservation is needed for a job. It is the only time characteristic that can be changed by the user after the job was submitted. If it doesn't violate the schedule it is possible to extend the execution time of the previously submitted and running job. Planing the job execution a user can specify time period when a job must be executed. The presented job description (see fig. 2) illustrates usage of aforementioned functionality specifying liberal requirements that the job should be executed within the first ten days of May, but except Saturdays and Sundays.

```
<grmsjob appid = "interactive_example">
  <simplejob>
    <executable type="single" count="1">
      <file name="exec-file" type="in">
        <url>file:///${HOME}/interactive_test/interactive_exec</url>
      </file>
    </executable>
    <executionTime>
      <timeSlot>
        <slotStart>10:30:00</slotStart>
        <slotEnd>13:15:00</slotEnd>
      </timeSlot>
      <execDuration>P0Y0M0DT2H20M0S</execDuration>
      <timePeriod>
        <periodStart>2005-05-01T00:00:00-00:00</periodStart>
        <periodDuration>P0Y0M10DT0H0M0S</periodDuration>
           <excluding>
             <weekDay>Saturday</weekDay>
             <weekDay>Sunday</weekDay>
           </excluding>
      </timePeriod>
    </executionTime>
  </simplejob>
</grmsjob>
```

Fig. 2. Exemplary job description for interactive task

3 Managing Interactive Jobs

In this paragraph we will focus on both, batch and interactive computational jobs. Interactive jobs need a definitely more sophisticated approach. The time when the application GUI is performing the job (or in other words: job start and finish time) has to be synchronized with user work-time preferences and should be known by the user in advance. The mechanism responsible for displaying the interface has to be carefully designed, to address authorization and security issues.

A special module responsible for managing interactive-related data is needed to put into practice the idea of running interactive jobs in the grid. This module has to keep information about choosing application server for running the job, about port number where VNC server will listen, about password needed for connection and finally about current job status. In our conception this functionality will be met by the *Interactive Application Manager* (IAM).

Due to necessity of managing information about interactive connections, VNC server and target applications on the application server side - VNC Manager was developed. One instance of the VNC Manager is run for each application. First, it is looking for free port number (from the range specified by administrator), next it generates password, valid for this session only. All these data are sent to IAM and then VNC server with interactive application are launched. When application finishes VNC Manager kills previously run applications.

All operations connected with interactivity are transparent for GRMS. It manages interactive tasks in the same way like other ones. Reservation for a given session is stored in the Reservation Manager. Limitation for the number of interactive sessions in the same time per machine are necessary due to avoiding server overload. It can be changed dynamically by administrator.

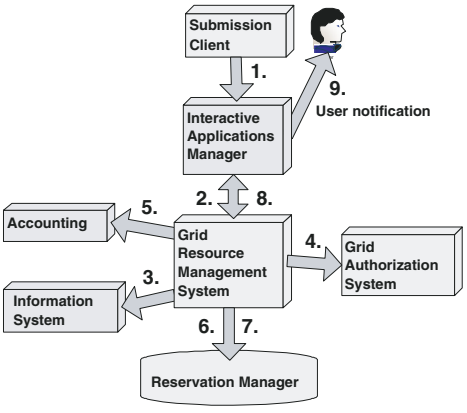


Fig. 3. Interactive job submission

All operations performed on interactive tasks are presented in the following paragraphs. Diagram for the batch job is not presented here. On diagrams data storing element is omitted. It is assumed that data is staged in to the application server before the interactive job starts and staged out after it from/to any file server (e.g. DMS). It is GRMS that is responsible for file staging.

3.1 Job Submission

Interactive jobs are submitted into the system in a same manner as the batch ones. Each consecutive step on the diagram was marked (see figure 3). The detailed description is given below.

1. Job is sent by a user to the Interactive Applications Manager.
2. Next, job is sent to the GRMS and the IAM registers for notifications.
3. GRMS asks the Information System about available computational machines at the moment, which fulfil requirements (e.g. desired application must be installed there).
4. GRMS checks where the user can run his jobs for the sake of the user's privileges (rights).
5. GRMS checks where the user has accounting limits for computation (for a given period of time).
6. GRMS checks available resources (e.g. port number) on machines for the interactive job.
7. GRMS makes a decision about the best machine for the user's job and reserves resources in the Reservation Manager.
8. Information about decision (machine and running time) is passed to the IAM.
9. The user is informed (e.g. via e-mail) about the job running time.

3.2 Establishing a Secure Connection

The interactive job is submitted to the system and the VNC session is scheduled. The next step is to prepare the proper environment for the given job, launch it and wait for connection establishment from the user (figure 4).

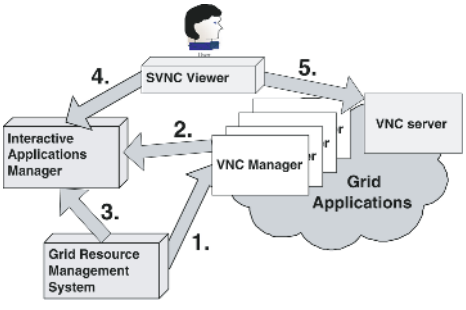


Fig. 4. Establishing a secure VNC connection

This operation describes following steps:

1. The GRMS launches scheduled job.
2. VNC Manager reports the port number in use and dynamically-generated password to the IAM.
3. The IAM is notified when everything is ready and when the session can be established.
4. The user starts the SVNC Viewer which takes all connection parameters from the IAM.
5. Secure connection is established to the VNC server.

3.3 Prolonging the Session

Any interactive job can be scheduled for the certain amount of time. The time period is specified by the user during task definition and submission. When the reservation period is about to expire, the client application (e.g. SVNC viewer with extended functionality) displays the appropriate warning and the user is given the possibility to request the session prolongation. After evaluation of the actual session state (Reservation Manager, etc.) the prolonged access is granted or the request is refused (figure 5).

Prolonging the VNC session step by step:

1. The user requests about session prolonging is sent to the IAM.
2. Further, it is forwarded to the GRMS.
3. GRMS asks the Information System about machine availability.
4. GRMS checks the user rights validity.
5. GRMS checks user's accounting limit.
6. GRMS checks available resources for the next time period.

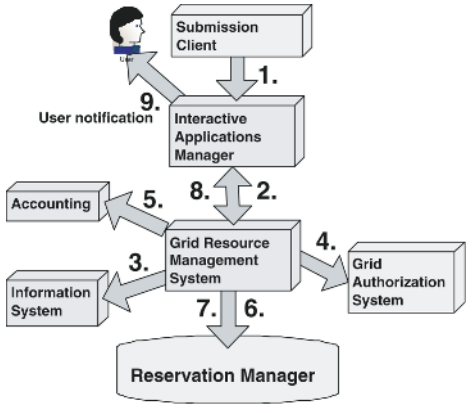


Fig. 5. Prolonging the VNC Session

7. GRMS makes a decision about the prolonging and change reservation in Reservation Manager.
8. Information is passed to the IAM.
9. User is informed about the decision (e.g. via e-mail).

3.4 Finishing the VNC Session

The VLab user has the ability to end an active VNC session at any time after the session has been started (figure 6a)).

The procedure is explained below:

1. The request concerning finishing the session is sent to IAM.
2. Next it is forwarded to GRMS.
3. The GRMS system sends the appropriate signal to the instance of VNC Manager responsible for a given application. Application is closed and resources are released.
4. GRMS updates the Resource Manager - registration is removed.

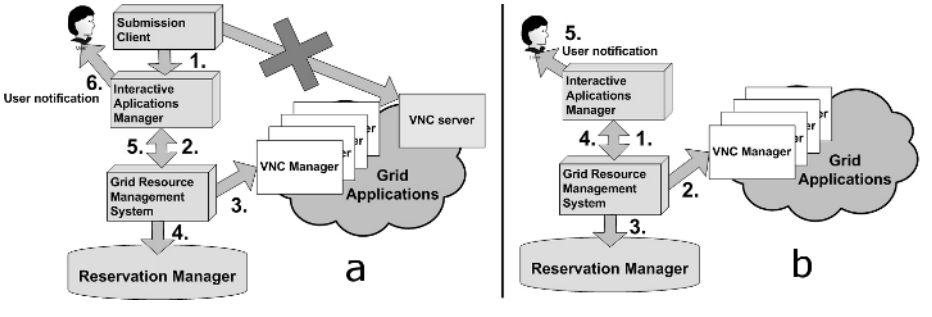


Fig. 6. Ending the VNC session: a) by the user b) by the system

5. The GRMS sends notification about job finishing to IAM.
6. The user is informed about job being finished.

Furthermore, an active VNC session can be terminated by the system (e.g. GRMS) when the reservation period expires, or for any other reason. The procedure is very similar to the one described above, with a difference in steps 1 and 2. In that case, GRMS sends a signal to stop the application (via VNC Manager) when the reservation time expires (figure 6b)).

4 Summary

Possibility of launching interactive jobs in the grid environment is undoubtedly very important for many users. In the paper a general concept from this area was presented. Thanks to its universality the idea can be implemented for many applications and adapted to many clients. The solution presented was implemented experimentally in the Virtual Laboratory system.

References

1. Okoń, M., Lawenda, M., Meyer, N., Stokłosa, D., Rajtar, T., Kaliszan, D., Stroiński, M.: Interactive task invocation in the Virtual Laboratory. The 4th Cracow Grid Workshop, ISBN 83-915141-4-5, pp. 293-300, Cracow, Poland, December 12-15, 2004
2. Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., Pukacki, J.: Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS. Scientific Programming. IOS Press. Amsterdam The Netherlands 12:4 (2004) 263-273
3. Adamski, M., Grabowski, P., Kurowski, K., Lewandowski, B., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., Piontek, T., Pukacki, J., Strugalski, R.: GridLab Middleware Services for Managing Dynamic Computational and Collaborative Infrastructures (VOs). Proceedings of 13th Annual Mardi Gras Conference, 2005
4. GRMS - WP9 Resource Management - GridLab Project `http://www.gridlab.org/grms`
5. GridLab Project `http://www.gridlab.org/`
6. Virtual Laboratory project `http://vlab.psnc.pl/`
7. Data Management System `http://dms.progress.psnc.pl/`

Comparison of Pricing Policies for a Computational Grid Market

Omer Ozan Sonmez and Attila Gursoy

Dept. of Computer Engineering,
Koç University, Istanbul, Turkey
osonmez05@alm.ku.edu.tr, agursoy@ku.edu.tr

Abstract. In this paper, we demonstrate and discuss the economical results of applying a fixed, a dynamic and a stochastic approximation based pricing policy in a free commodity market model designed for computational grids. We present the pricing policies and our economy-driven scheduling heuristic as a part of our market model in which we assume resource owners desire to profit or recover their costs, and users desire to execute their jobs within the limits of their budget and time. The comparison experiments revealed that our dynamic pricing policy is more successful as a means for achieving social welfare in the market.

1 Introduction

Computational grids have emerged to exploit geographically dispersed resources to solve large-scale computational and data demanding scientific problems [1]. To make use of a computational grid, an efficient scheduling mechanism is needed in order to assign each job to the most appropriate resource [2]. Real-world market models have been thought as a very natural way to manage large distributed resource allocation problems and consequently have been applied to solve the scheduling problems in the distributed computing framework including grid systems [3]. However, attaining the optimal resource allocation has not been considered as the only motivation behind applying market-based mechanisms in grid computing. Since there is no strong motivation to make people share their resources in the current grid systems, it has been considered that market models in which users pay for resources, would give resource owners incentive to share their resources making the computing power economically available that the society requires [4].

In this work, we consider three pricing policies (fixed, dynamic and stochastic approximation based [5]) that the resource owners may use in a free commodity market model. We performed simulation-based comparison experiments to evaluate the policies in terms of achieving social welfare, which is the utility of all market agents considered in aggregate.

The rest of the paper is organized as follows: Section 2 reviews the market-based distributed resource allocation studies. Section 3 describes the market model and the pricing policies. The experimental results and discussion is presented in Section 4. Finally, Section 5 makes some concluding remarks.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 766–773, 2006.

2 Related Work

In a real-world market, there exist a variety of economic models for determining the price of a commodity or a service, based on supply and demand [4]. Several approaches of these real market models have been applied to conventional distributed systems before grid systems have emerged. However, all these studies [6],[7],[8] adapted market mechanisms only to get optimal scheduling results. Setting up a real computational market seems to be more specific to grid computing.

Various real grid resource allocation systems that employ market structures have been developed and are being progressed such as Nimrod-G [9], Mariposa [10] and Compute Power Market [11]. Referring to the current interest on this subject we can assert that such systems will be widely used in the near future.

3 The Market Model

In this market model, we assume that self-interested resource owners desire to profit or recover their costs allocating their resources to the most profitable jobs, and self-interested users desire to solve their problems considering their budget and time constraints.

In the market model, the economy driven scheduling heuristic that we have proposed in [12], determines users' preferences; it can optimize time, cost or both of them taking account the budget and deadline constraint of a user. In this work, we only consider scheduling parameter sweep type of applications that involve the execution of various independent tasks over a choice of parameters [13]. Generally speaking, the heuristic assigns the biggest jobs to the cheapest resources if the application is cost limited, it assigns the biggest jobs to the fastest resources, if the application is time limited, and for other cases, it assigns the biggest jobs to the resources that offer the best value in terms of performance per money. Besides, we should note that we have made the following assumptions for the heuristic; each resource comprises identical processing elements, speed of a processing element is defined in Million Instructions per Second (MIPS) rating and size of a job is known a priori and represented in Million Instructions (MI) rating.

In a free commodity market, resource owners are free to determine a pricing policy and competitive users take prices as given. We consider three pricing policies that the resource owners may use; a fixed pricing policy that actually does nothing but preserve the initial prices, a dynamic pricing policy that adjusts prices according to the supply-demand dynamics, and a stochastic approximation based policy [5], which was actually proposed to be used in the e-commerce applications. For all pricing policies, we assume that each resource owner individually specifies a minimum price per unit time that she is not willing to execute jobs any price lower than this.

3.1 The Dynamic Pricing Policy

In this pricing policy, resource owners can enter the market with a price above than or equal to the minimum price determined in advance, and they adjust the

prices over time according to the demand on their resources. The demand on a resource, at a time instant, is assumed to be the utilization of the resource, *i.e.* the fraction of the resource's total capacity that is being used.

Algorithm 1. DynamicPrice($P_{init}, P_{min}, \mu, T$)

1. Set initial price $p = P_{init}$
2. Set previous utilization rate $ExUr = 0$
3. **for** i : 1 to ∞ **do**
4. Wait for a period of T
5. $Ur = getCurrentUtilization()$
6. **if** $Ur > \mu$ **and** $Ur > ExUr$ **then**
7. $\Delta p = Ur - \mu$
8. Update the current price $p = p * (1 + \Delta p)$
9. **else if** $Ur < \mu$ **then**
10. $\Delta p = \mu - Ur$
11. Update the current price $p = p * (1 - \Delta p)$
12. **if** $p < P_{min}$ **then**
13. $p = P_{min}$
14. **endif**
15. **endif**
16. $ExUr = Ur$
17. **endfor**

By Algorithm 1 each resource owner calculates a new price-per-unit-time; p. In the algorithm, Ur $(0 < Ur < 1)$ denotes the utilization and Δp is the price increment or decrement percentage. We assume that resource owners interpret a utilization below than a ratio that they consider (*i.e.* μ, $0 < \mu < 1$) as a decrease and a higher ratio as an increase on the demand.

If the utilization is over μ, owners increase the price only when the utilization changes (line 6-8), *i.e.* when new jobs are accepted or some running jobs are completed to prevent continuous price increase; however, they decrease the price at each predetermined time interval if the utilization is under μ (line 9-11), in order to attract the users. Naturally, price of a resource do not fall below the minimum price that the owner has specified (line 12-13). Moreover, once a job is accepted, it is charged with the agreed price. Further price changes do not affect the running or the queued jobs.

3.2 The Pricing Policy Based on Stochastic Approximation

Given a reasonable initial price, the policy (illustrated in Algorithm 2) simply tries to converge to a local optimal price, p, that is expected to maximize the revenues [5]. Using a step size, Δ, determined as a decreasing function of the number of trials so far, I, such as $I^{-1/3}$ (line 4), the policy conducts sales at both prices $p + \Delta$ (line 5) and $p - \Delta$ (line 7) for a certain period of time, T, and based on the number of received jobs during these periods (in its original form this is the number of commodities that has been sold), $S(p + \Delta)$ (line 6) and $S(p - \Delta)$ (line 8), it calculates the profits (revenues) obtained for the respective

prices (line 10-11). Then it updates the current price as shown in the line 13. Here, A is the update interval that is set as a decreasing function of the trial number such as $1/I$. Finally, the resulting price is prevented to be lower than the minimum price, P_{min} (line 14-15), and further price changes do not affect the running or the queued jobs, in this policy as well.

Algorithm 2. StochPrice($P_{init}, P_{min},\ T$)

1. Set initial price $p = P_{init}$
2. Set trial number $I = 1$
3. **for** $I : 1$ to ∞ **do**
4. Set step size $\Delta = I^{-1/3}$
5. For a period of T, set the price to $p + \Delta$
6. Let $S(p + \Delta)$ be the amount of jobs received during this time
7. For a period of T, set the price to $p - \Delta$
8. Let $S(p - \Delta)$ be the amount of jobs received during this time
9. Calculate the obtained profit as follows:
10. $P(p + \Delta) = S(p + \Delta) * (p + \Delta)$
11. $P(p - \Delta) = S(p - \Delta) * (p - \Delta)$
12. Set the update interval $A = 1/I$
13. Update the current price $p = p + \frac{A}{\Delta} * \frac{P(p+\Delta)-P(p-\Delta)}{2T}$
14. **if** $p < P_{min}$ **then**
15. $p = P_{min}$
16. **endif**
17. **endfor**

4 Experimental Results and Discussion

The attributes of the simulated resources are given in Table 1. In all of the experiments, users submit applications within a 10000 time unit interval, based on a Poisson arrival process. The experiments were repeated 10 times for each average arrival rate value, λ, that was varied from 5x10^{-4} to 5x10^{-3} with a

Table 1. Attributes of the simulated resources

Resource Name	Number of PEs	MIPS Rating of each PE	Resource Management	Min. Price per MI in G$
R1	32	10	Time-Shared	0.35
R2	36	12	Time-Shared	0.391
R3	48	13	Time-Shared	0.4
R4	24	15	Time-Shared	0.42
R5	36	20	Time-Shared	0.45
R6	16	12	Time-Shared	0.383
R7	24	11	Time-Shared	0.372
R8	40	10	Time-Shared	0.36
R9	24	10	Time-Shared	0.34

Table 2. Variation intervals for job parameters

Number of Jobs in an Application	Size of a Job	Input File Size	Output File Size
50-150	1000-10000 MI	250-750 B	2.5-7.5 KB

$5x10^{-4}$ step size. The variation of the parameters related to jobs are given in Table 2. For deadline and budget of each user, a relaxed and a tight (limited) value are computed considering the size of the application, and the minimum prices of the resources. Subsequently, from the tight-relaxed distributions a random budget and a random duration (time-constraint) are assigned to the user. Resource owners enter the market with the minimum prices. The sale period of the stochastic approximation based pricing policy, T, and the price update interval of the dynamic pricing policy is set to 5 time units, and the desired utilization rate parameter, μ, is set to 0.5.

Fig. 1 demonstrates the average income distribution of resource owners. Under the fixed pricing policy R5 holds the 74% of the total market gain, leaving only small shares to the other resources. This is because R5 is the most preferable resource in terms of time and time-cost optimization. Since prices are constant in the fixed pricing policy, resources are not able to adjust their prices to attract the users. However, all of the resources have been able to earn money under the dynamic and stochastic approximation (SA) based policies, breaking the monopoly of R5. The SA based policy can quickly converge to a local optimum price; however, after a certain amount of time (as A/Δ ratio increases) it behaves like the fixed price policy. Therefore, it is not versatile as the dynamic policy in adapting to the demand variations. This is clearly the reason for the income difference.

There cannot be an equilibrium point for the prices in this market since resources are not identical and preferences differ among users. As Fig. 2 illustrates

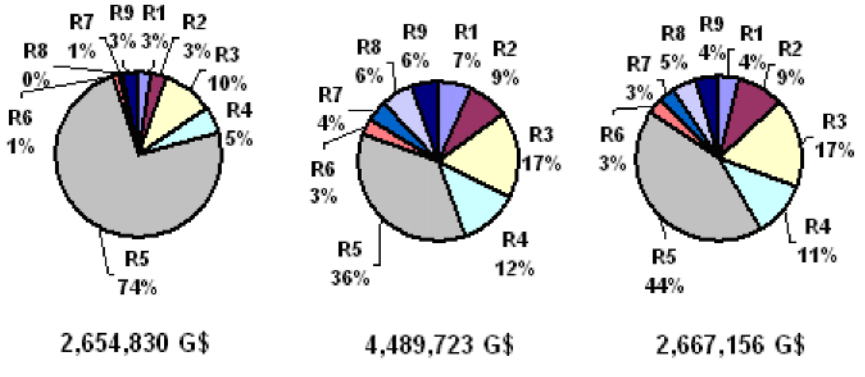


Fig. 1. Average income distribution under the fixed (the chart on the left), the dynamic (the chart on the middle), and stochastic approximation based (the chart on the right) pricing policies

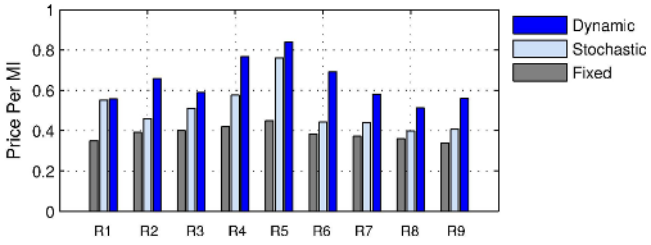


Fig. 2. Average price-per MI of the resources under the pricing policies

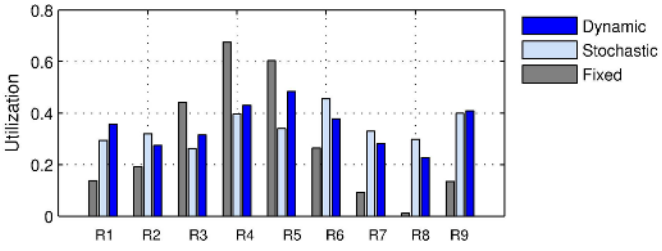


Fig. 3. Average utilization of the resources under the pricing policies

Price Per MI values varies among the resources. However, identical or comparable resources in terms of speed, have similar prices under the dynamic and SA based policies. Therefore, we can claim that these policies achieve a partial equilibrium in the market.

The average utilization of all resources under the fixed, dynamic, and SA based policies are 28%, 35%, and 34% respectively. As Fig. 3 shows, the average utilization of the resources are more balanced under the SA based and dynamic policies than that of the resources under the fixed policy.

Fig. 4 shows the average percentage of the applications completed without a budget shortage or a deadline violation. It can be seen that more applications are completed under the dynamic pricing policy. This is because it makes users,

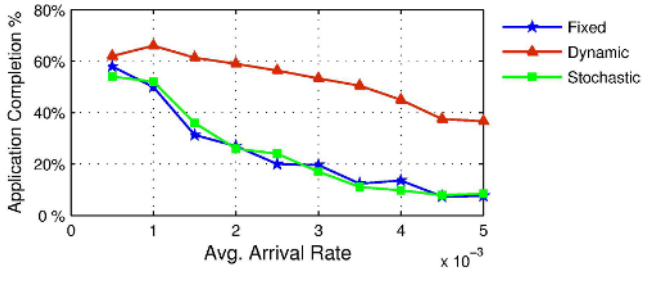


Fig. 4. Average application completion percentages under the pricing policies

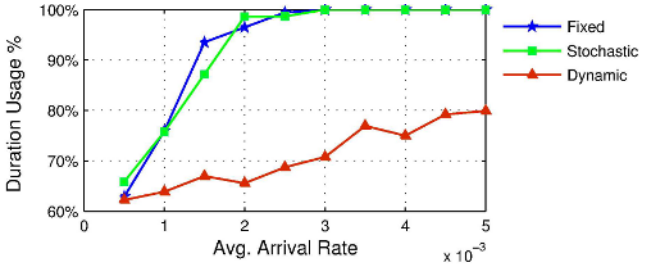


Fig. 5. Average duration usage percentages under the pricing policies

rapidly, tend to less loaded and cheaper resources when the formerly suitable resources became loaded and consequently their prices are increased. As Fig. 5 demonstrates, under the fixed and SA based policies, users' deadlines expire as the demand on the resources increases; hence, application completion percentages decreases.

To sum up, considering the experimental results we can assert that the dynamic pricing in which prices are adjusted with respect to the real-world market dynamics yields better social outcomes than the other pricing policies.

5 Conclusion

In real-world free markets, there is no and can be no any globally accepted method for price adjustment; each producer or vendor applies an individual pricing policy. Accordingly, we devised a rational dynamic pricing policy for computational grid markets such that resource owners can adjust the prices according to the demand on their resource. Then we have performed experiments that compares our policy with a fixed and a stochastic approximation based pricing policy and discussed the results both from users' and resource owners' point of view.

References

1. R. Buyya, R. Murshed, and D. Abramson, "A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global grids", Int. Conf. on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, 2002.
2. J. Nakai, "Pricing computing resources: Reading Between the Lines and Beyond", NASA Ames Research Center, Tech. Rep. NAS-01-010, 2002.
3. K. Krauter, R. Buyya and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", *International Journal of Software: Practice and Experience*, 2002, pp. 135-164.
4. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing", *The Journal of Concurrency and Computation: Practice and Experience*, 2002, pp. 1507-1542.

5. N.Abe and T.Kamba, "A Web marketing system with automatic pricing", *The International Journal of Computer and Telecommunications Networking*, 2000, vol. 33, pp. 775-788.
6. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and S. Stornetta. "Spawn: A distributed computational economy", *IEEE Transactions on Software Engineering*, 1992, vol. 18, pp. 103-117.
7. T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard, "Enterprise: A market-like task scheduler for distributed computing environments", *The Ecology of Computation*, B.A Huberman Ed., Amsterdam: Elsevier Science Publishers, 1988, pp. 177-255.
8. M. P. Wellman, "A market-oriented programming environment and its application to distributed multicommodity flow problems", *Journal of Artificial Intelligence Research*, 1993, pp. 1-23.
9. R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid", *Int. Conference on High-Performance Computing*, 2000, pp. 283-289.
10. M. Stonebraker et al., "Mariposa: a wide-area distributed database system", *VLDB Journal*, 1996, vol. 5, pp. 48-63.
11. R. Buyya and S. Vazhkudai, "Compute power market: Towards a market-oriented grid", *Int. Symposium on Cluster Computing and the Grid*, 2001, pp. 574.
12. O. O. Sonmez and A. Gursoy, "A novel economic-based scheduling heuristic for computational grids", Int. Computational Science and Engineering Conference, Istanbul, Turkey, 2005.
13. R.Buyya, J.Giddy, and D.Abramson, "An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications", *Proceedings of the 2nd International Workshop on Active Middleware*, Pittsburgh, Pennsylvania, USA, 2000.

Two Level Job-Scheduling Strategies for a Computational Grid*

Andrei Tchernykh[1], Juan Manuel Ramírez[2], Arutyun Avetisyan[3], Nikolai Kuzjurin[3], Dmitri Grushin[3], and Sergey Zhuk[3]

[1] CICESE Research Center, Ensenada, México
chernykh@cicese.mx
[2] University of Colima, México
jmramir@ucol.mx
[3] Institute of System Programming RAS, Moscow
{arut, nnkuz, grushin, zhuk}@ispras.ru

Abstract. We address parallel jobs scheduling problem for computational GRID systems. We concentrate on two-level hierarchy scheduling: at the first level broker allocates computational jobs to parallel computers. At the second level each computer generates schedules of the parallel jobs assigned to it by its own local scheduler. Selection, allocation strategies, and efficiency of proposed hierarchical scheduling algorithms are discussed.

1 Introduction

Recently, parallel computers and clusters have been deployed to support computation-intensive applications and become part of so called computational grids (C-GRIDs) or metacomputers [10, 8]. Such C-GRIDs are emerging as a new paradigm for solving large-scale problems in science, engineering, and commerce [15]. They comprise heterogeneous nodes (typically, clusters and parallel supercomputers) with a variety of computational resources. The efficiency of scheduling policies is crucial to C-GRID performance. The job scheduling solutions for a single parallel computer significantly differs from scheduling solution in such a grid. The scheduling problem becomes more complicated because many computers of different sizes are involved with different local scheduling policies [11, 12, 13, 14]. One possible solution is to consider two-level scheduling schemes: at the first level jobs are allocated to parallel computers by a GRID resource broker and then local schedulers are used at each computer. Typically, the broker is responsible for resource discovery, resource selection, and job assignment to ensure that the user requirements and resource owner objectives are met. The broker acts as a mediator between users and resources using middleware services. It is responsible for presenting the grid to the user as a single, unified resource. One of the broker's major responsibilities is to provide centralized access to distributed

* This work is partly supported by CONACYT (Consejo Nacional de Ciencia y Tecnología de México) under grant #32989-A, and by RFBR (Russian Foundation for Basic Research),grants 05-01-00798 and 03-07-00198.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 774–781, 2006.

resources. This simplifies the use of the computational GRID aggregating available computational resources, and collecting information on the current state of these resources.

In this paper, we discuss several scheduling policies for two level hierarchy: at the first level the broker allocates computational jobs to C-GRID node according to some selection criteria taking parameters of jobs and computers into consideration. At the second level, each node generates schedules by its own local scheduler. We present scheduling strategies based on combination of some selection strategies and scheduling algorithms. We limit our consideration to the scenario where jobs are submitted to the broker from a decentralized environment of other brokers, and can be processed into the same batch. The main objective of the paper is to compare different scheduling strategies and estimate their efficiency. In Section 2, we present a brief overview on two level hierarchy scheduling strategies, and compare their the worst case behavior in Section 3, followed by concluding remarks in Section 4.

2 Scheduling Strategies

2.1 Model

Let we have n jobs $J_1, J_2, ..., J_n$, and m uniform C-GRID nodes $N_1, N_2, ..., N_m$, characterized by $M = [m_1, m_2, ..., m_m]$, where m_i is the number of identical processors of the node N_i. We assume that there is no inter-communications between jobs, that they can be executed at any time, in any order, and on any node.

Each job is described by 2-tuple $(s_j, p^j_{s_j})$, where s_j is a job size that is referred to as the job's *degree of parallelism* or number of processors required for J_j, $p^j_{s_j}$ is the execution time of job J_j on s_j processors. The job work also called job area is $W_j = p^j_{s_j} \cdot s_j$. Each job can be executed at a single node, so the maximum size of a job is less than or equal to the maximum number of processors in a node. This means that system resources are not crossed, and co-allocation problem is not considered. All strategies are analyzed according to their approximation ratio. Let $C_{\text{opt}}(I)$ and $C_A(I)$ denote makespans of an optimal schedule and of a strategy A for a problem instance I, respectively. The approximation ratio of the strategy A is defined as $\rho_A = \sup_I C_A(I)/C_{\text{opt}}(I)$, and we call A an ρ approximation algorithm.

In this paper, we restrict our analysis to the scheduling systems where all jobs are given at time 0 and are processed into the same batch. This means that a set of available ready jobs is executed up to the completion of the last one. All jobs which arrive in the system during this time will be processed in the next batch. A relation between this scheme and the scheme where jobs arrived over time, either at their release time, according to the precedence constraints, or released by different users is known and studied for different scheduling strategies. Using results [6] the performance guarantee of strategies which allows release times is 2-competitive of the batch style algorithms.

2.2 Two Level Hierarchy Scheduling

The scheduling consists of two parts: selection of a parallel node for a job and then local scheduling at this node.

Selection Strategies. We consider the following scenario. On the first stage, to select a node for the job execution the broker analyzes the job request, and current C-GRID resources' characteristics, such as a load (number of jobs in each local queue), parallel load (sum of jobs' sizes or jobs' tasks), work (sum of jobs work), etc. The parameters of the jobs already assigned to nodes and known by the broker are used only. All nodes are considered in non decreasing order of their sizes m_i, $m_1 \leq m_2 \leq \cdots \leq m_m$. Let $first(J_j)$ be the minimum i such that $m_i \geq s_j$. Let $last(J_j)$ be the maximum i such that $m_i \geq s_j$. If $last(J_j) = m$ we denote the set of nodes $N_i,\ i = first(J_j), ..., last(J_j)$ as the set of available nodes *M-avail*. If $last(J_j)$ is the minimum r such that $\sum_{i=first(J_j)}^{r} m_i \geq \frac{1}{2} \sum_{i=first(J_j)}^{m} m_i$, we denote the set of nodes $N_i,\ i = first(J_j), ..., last(J_j)$ as the set of admissible nodes *M-admis*. The broker selects node for a job request using the following strategies:

- *Min-Load (ML)* strategy takes the node with the lowest load per processor (number of jobs over number of processors in the node).
- *Min-Parallel-Load (MPL)* strategy takes the node with the lowest parallel load per processor (the sum of job sizes over number of processors in the node).
- *Min-Lower-Bound (MLB)* strategy chooses the node with the least possible lower bound of completion time of previously assigned jobs, that is the node with the lowest work per processor. Instead of the actual execution time of a job that is an offline parameter, the value provided by the user at job submission, or estimated execution time is used.
- *Min-Completion-Time (MCT)*. In contrast to *MLB*, the earliest possible completion time is determined based on a partial schedule of already assigned jobs [9, 7]. For instance, Moab [3] can estimate the completion time of all jobs in the local queue because jobs and reservations possess a start time and a wallclock limit.

Local Scheduling Algorithms. We address the space sharing scheduling problem, hence scheduling can be viewed as a problem of jobs packing into strips of different width. In such geometric model each job corresponds to a rectangle of width s_j and height $p^j_{s_j}$. One known strategy for packing is the *Bottom-Left(BL)*. Each rectangle is slid as far as possible to the bottom and then as far as possible to the left [16]. It is known that for some problems *BL* can not find constant approximation to the optimal packing, but a successful approach is to apply *BL* to the rectangles ordered by decreasing sizes that is referred as *Bottom Left Decreasing(BLD)* or *Larger Size First(LSF)*. In this paper we use *LSF* for local scheduling.

3 Analysis

The *LSF* has been shown to be a 3-approximation [2]. Some results about asymptotic performance ratio of different strategies for this problem and improvements are presented in [1, 4, 5].

Below we will consider *LSF* for local scheduling and the following selection strategies: *Min-Load(ML)*, *Min-Load-admissible(ML-a)*, *Min-Parallel-Load(MPL)*, *Min-Parallel-Load-admissible(MPL-a)*, *Min-Lower-Bound(MLB)*, *Min-Lower-Bound-admissible(MLB-a)*, *Min-Completion-Time(MCT)*, and *Min-Completion-Time-admissible(MCT-a)*.

3.1 (ML, ML-a, MPL, MPL-a)-LSF

The simple example below shows that ML, ML-a, MPL, MPL-a selection strategies combined with *LSF* cannot guarantee constant approximation in the worst case. It is sufficient to consider m nodes of width 1 and the following list of jobs: $m-1$ jobs J_1, then J_2, then $m-1$ jobs J_1, etc., where $J_1 = (1, \varepsilon)$, $J_2 = (1, E)$. Suppose $n = rm$, where $r \in \mathbb{N}$. Note that $C_{(\mathrm{ML,MPL})-\mathrm{LSF}} = rE$ and $C_{\mathrm{opt}} \leq \lceil \frac{(m-1)r}{m} \rceil \varepsilon + \lceil \frac{r}{m} \rceil E$. If $E/\varepsilon \to \infty$, $m \to \infty$, and $r \to \infty$ then $\rho_{(\mathrm{ML,MPL})-\mathrm{LSF}} \to \infty$.

3.2 MCT-LSF

In the following two theorems we prove that constant approximation for *MCT-LSF* strategy is not guaranteed and that *MCT-a-LSF* is a 10 approximation algorithm.

Theorem 1. *For a set of grid nodes with identical processors and for a set of rigid jobs the constant approximation for MCT-LSF strategy is not guaranteed (in the worst case).*

Proof. Let us consider grid nodes and jobs that are divided into groups according to their sizes. Let there are $k+1$ groups of nodes and $k+1$ sets of jobs. The number of nodes in group i is equal $M_i = 2^i$ for $0 \leq i \leq k$. The number of jobs in a set i is equal $n_i = (i+1) \cdot 2^i$. The size of the nodes in group i is equal to the job size in the set i, $s_i = m_i = 2^{k-i}$. The execution time (height) of jobs in the set i is $p^i = \frac{1}{i+1}$. Since $p^i n_i s_i / M_i m_i = \frac{1}{i+1}(i+1)2^i/2^i = 1$, obviously $C_{\mathrm{opt}} = 1$. However, $n_i s_i = 2^{k-i}(i+1) \cdot 2^i = (i+1) \cdot 2^k = \sum_{j=0}^{i} M_j m_j = \sum_{j=0}^{i} 2^j 2^{k-j}$. Hence, any set of jobs may completely fill one layer of available nodes, and if jobs come in increasing order of their sizes $C_{\mathrm{MCT-LST}} = \sum_{j=0}^{k} \frac{1}{j+1} \sim lnk$ that means that the ratio $C_{\mathrm{MCT-LSF}}/C_{\mathrm{opt}}$ may be arbitrary large. □

3.3 MCT-a-LSF

Theorem 2. *For any list of rigid jobs and any set of grid nodes with identical processors the MCT-a-LSF is a 10 approximation algorithm.*

Proof. Let the maximum completion time is achieved at the kth node, J_a be the job that has been received last from the broker by this node, and $f = first(J_a)$, $l = last(J_a)$. Let $Y_f, \ldots, Y_l$ be the sets of jobs that were allocated on the nodes $f, \ldots, l$ (admissible for J_a), just before getting the job J_a. Because the job was sent to the kth node, then the completion time of the node $C_i + p^a \geq C_k + p^a \geq C_{\text{MCT-a-LSF}}$ $\forall i = f, ..., l$. Let in the node N_k the job with maximum completion time be J_c. Over all jobs with maximum completion time the job with largest processing time is chosen. Let t_c be the time when this job has started the execution, and let $r_c = t_c - p^a$. We have $t_c + p^c \geq C_{\text{MCT-a-LSF}}$, $r_c + p^a + p^c \geq C_{\text{MCT-a-LSF}}$

$$\sum_{i=f}^{l} m_i r_c + p^a \cdot \sum_{i=f}^{l} m_i + \sum_{i=f}^{l} m_i p^c \geq C_{\text{MCT-a-LSF}} \cdot \sum_{i=f}^{l} m_i \tag{1}$$

Before the time t_c the kth node is filled at least half (the property of the BLD algorithm) [2], hence $W_k \geq \frac{1}{2} \cdot m_k \cdot r_c$.

Let J_b be the job which requires minimal number of processors among the jobs allocated on nodes $f, ..., l$, and let $f_0 = first(J_b)$. Then all jobs allocated on nodes $f, ..., l$ cannot be allocated on nodes N_i with $i < f_0$. Since J_b is allocated on one of the nodes $f, ..., l$ then $last(J_b) \geq f \Rightarrow \sum_{i=f_0}^{f-1} m_i \leq \frac{1}{2} \sum_{i=f_0}^{m} m_i \Rightarrow \sum_{i=f}^{m} m_i \geq \frac{1}{2} \sum_{i=f_0}^{m} m_i$. Since $l = last(J_a)$, then $\sum_{i=f}^{l} m_i \geq \frac{1}{2} \sum_{i=f}^{m} m_i$ and

$$\sum_{i=f_0}^{m} m_i \leq 2 \sum_{i=f}^{m} m_i \leq 4 \sum_{i=f}^{l} m_i \tag{2}$$

Thus, $C_{\text{opt}} \cdot \sum_{i=f_0}^{m} m_i \geq S(\bigcup_{i=f}^{l} W_i) \geq \frac{1}{2} \sum_{i=f}^{l} m_i r_i$, where $S(\bigcup_{i=f}^{l} W_i)$ denote the sum of jobs' areas allocated at nodes $f, \ldots, l$. By (2)

$$\sum_{i=f}^{l} m_i r_i \leq 2 \cdot C_{\text{opt}} \cdot \sum_{i=f_0}^{m} m_i \leq 8 \cdot C_{\text{opt}} \cdot \sum_{i=f}^{l} m_i \tag{3}$$

The inequalities $C_{\text{opt}} \geq p^j, \forall j$, (1) and (3) imply

$$8 \cdot C_{\text{opt}} \cdot \sum_{i=f}^{l} m_i + p^a \cdot \sum_{i=f}^{l} m_i + \sum_{i=f}^{l} m_i p^i \geq C_{\text{MCT-a-LSF}} \cdot \sum_{i=f}^{l} m_i,$$

$$8C_{\text{opt}} \cdot \sum_{i=f}^{l} m_i + p^a \cdot \sum_{i=f}^{l} m_i + C_{\text{opt}} \cdot \sum_{i=f}^{l} m_i \geq C_{\text{MCT-a-LSF}} \cdot \sum_{i=f}^{l} m_i,$$

$$8C_{\text{opt}} + p^a + C_{\text{opt}} \geq C_{\text{MCT-a-LSF}},\ 8C_{\text{opt}} + C_{\text{opt}} + C_{\text{opt}} \geq C_{\text{MCT-a-LSF}},$$

and, finally $C_{\text{MCT-a-LSF}} \leq 10 \cdot C_{\text{opt}}$ □

3.4 MLB-LSF

Theorem 3. *For a set of grid nodes with identical processors, and for a set of rigid jobs the constant approximation for MLB-LSF strategy is not guarantied (in the worst case).*

The proof is similar to the proof of Theorem 1, so we omit it here.

3.5 MLB-a-LSF

The strategy is similar to *MLB-LSF* with only one difference: only admissible nodes are considered for the selection. Selecting admissible nodes prevents narrow jobs filling wide nodes causing wide jobs waiting for execution. It also allows us to find a constant approximation for the algorithm.

Theorem 4. *For any list of rigid jobs and any set of grid nodes with identical processors the MLB-a-LSF is a 10 approximation algorithm.*

Proof. Let the maximum completion time be at the kth node when algorithm terminates. Let the job J_a be the last job with the execution time p^a that was added to this node, and f=*first*(J_a), l=*last*(J_a). Let $Y_f, \ldots, Y_l$ be the sets of jobs that had been already allocated at nodes $N_f, \ldots, N_l$ admissible for J_a before adding J_a, and let W_i be the total area of all jobs of Y_i, $(i = f, \ldots, l)$. Since J_a was added to the kth node of width m_k, $\frac{W_k}{m_k} \leq \frac{W_i}{m_i}, \forall i = f, \ldots, l$. Therefore,

$$\sum_{i=f}^{l} W_i = \sum_{i=f}^{l} \frac{W_i}{m_i} m_i \geq \sum_{i=f}^{l} \frac{W_k}{m_k} m_i = \frac{W_k}{m_k} \sum_{i=f}^{l} m_i \tag{4}$$

Let in packing by the *LSF*(BLD) algorithm, the set of rectangles corresponding to jobs allocated at the kth strip be $Y_k \bigcup \{J_a\}$, J_T be a job with maximum completion time, and t_T be the time when this job has started the execution, hence $C_{\mathrm{MLB-a-LSF}} = t_T + p^T$, where p^T is the processing time of J_T, and $C_{\mathrm{MLB-a-LSF}}$ is the completion time of the *MLB-a-LSF* algorithm. Let $r_k = t_T - p^a$. Then

$$C_{\mathrm{MLB-a-LSF}} = r_k + p^T + p^a \tag{5}$$

By the property of the *LSF(BLD)* algorithm [2]

$$W_k \geq \frac{1}{2} m_k r_k \Rightarrow r_k \leq \frac{2W_k}{m_k}. \tag{6}$$

Let J_b be the job having the smallest size among rectangles that packed at strips and let $f_0 = first(J_b)$. Hence any of the rectangles packed at $N_f, \ldots, N_l$ cannot be packed at a strip with number $< f_0$. As far as J_b is packed at one of the strips $f, \ldots, l$, $last(J_b) \geq f \Rightarrow \sum_{i=f_0}^{f-1} m_i \leq \frac{1}{2} \sum_{i=f_0}^{m} m_i \Rightarrow \sum_{i=f}^{m} m_i \geq \frac{1}{2} \sum_{i=f_0}^{m} m_i$. Since $l = last(J_a)$ and $\sum_{i=f}^{l} m_i \geq \frac{1}{2} \sum_{i=f}^{m} m_i$, we obtain $\sum_{i=f_0}^{m} m_i \leq 2 \sum_{i=f}^{m} m_i \leq 4 \sum_{i=f}^{l} m_i$.

Then, clearly $C_{\text{opt}} \sum_{i=f_0}^{m} m_i \geq \sum_{i=f}^{l} W_i$. Substituting (4) in this formula, we have $C_{\text{opt}} \sum_{i=f_0}^{m} m_i \geq \frac{W_k}{m_k} \sum_{i=f}^{l} m_i \geq \frac{1}{4} \frac{W_k}{m_k} \sum_{i=f_0}^{m} m_i$. Taking into account (6) we obtain

$$r_k \leq \frac{2W_k}{m_k} \leq 8C_{\text{opt}} \tag{7}$$

As far as $C_{\text{opt}} \geq p^j, \forall j$, (5) and (7) imply $8C_{\text{opt}} + C_{\text{opt}} + C_{\text{opt}} \geq C_{\text{MLB-a-LSF}}$ and, $C_{\text{MLB-a-LSF}} \leq 10 \cdot C_{\text{opt}}$. □

4 Concluding Remarks

In this paper, we discuss approaches and present solutions to multiprocessor job scheduling in computational Grid hierarchical environment that includes a resource broker and a set of clusters or parallel computers. The selection and allocation strategies are discussed. We show that our strategies provide efficient job management with constant approximation guarantee despite they are based on relatively simple schemes. The comparison of *MLB-a-LSF* and *MCT-a-LSF* strategies shows that *MLB-a-LSF* has the same worst case bound as *MCT-a-LSF*, however the *MCT* selection strategy is based on a partial schedule of already scheduled jobs and requires more computational effort than *MLB* strategy that based only on the job parameters from the list of assigned job. With *MLB-a-LSF* the broker can select appropriate node without feedback about the schedule from the node. The results are not meant to be complete, but give an overview on the methodology and some interesting relations. These results motivate finding approximation bounds of other two level hierarchy scheduling strategies. Another interesting question is how fuzzy execution time affects the efficiency. It seems important also to study moldable (or malleable) jobs hierarchical scheduling when the number of processors for a job is not given explicitly by a user but can be chosen by a broker or a local scheduler. Simulations are planned to evaluate proposed strategies considering real and synthetic workload models.

References

1. B. Baker, D. Brown, H. Katseff, A 5/4 algorithm for two-dimensional packing, J. of Algorithms, 1981, v. 2, pp. 348-368.
2. B. Baker, E. Coffman, R. Rivest, Orthogonal packings in two dimensions, SIAM J. Computing, 1980, v. 9, 4, pp. 846-855.
3. www.clusterresources.com
4. K. Jansen, Scheduling malleable parallel jobs: an asymptotic fully polynomial-time approximation scheme, Euro. Symp. on Algorithms, 2002.
5. C. Kenyon, E. Remila, A near optimal solution to a two dimensional cutting stock problem, Math. of Operations Res., 25 (2000), 645-656.

6. D. Shmoys, J.Wein, D.Williamson. Scheduling parallel machines on-line. SIAM J. Comput., 24:1313-1331, 1995.
7. S.Zhuk, A.Chernykh, N.Kuzjurin, A.Pospelov, A.Shokurov, A.Avetisyan, S.Gaissaryan, D.Grushin. Comparison of Scheduling Heuristics for Grid Resource Broker. PCS2004 Third International Conference on Parallel Computing Systems (in conjunction with ENC'04), IEEE, p. 388-392. 2004
8. Foster, C. Kesselman, editors. The Grid: Blueprint for a future computing infrastructure, Morgan Kaufmann, San Fransisco, 1999.
9. G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan, Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment, in the Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2003.
10. L. Smarr and C. Catlett. Metacomputing. Communications of the ACM, 35(6): 44-52, June 1992.
11. S. S. Vadhiyar and J. J. Dongarra, "A Metascheduler for the Grid," Proc. of 11-th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002.
12. J. Gehring and A. Streit, "Robust Resource Management for Metacomputers," In Proc. HPDC '00, pages 105-111, 2000.
13. V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. R. Buyya and M. Baker (Eds.) In Proc. Grid 2000, LNCS 1971, pp. 191-202, 2000.
14. A. James, K. A. Hawick, and P. D. Coddington, "Scheduling Independent Tasks on Metacomputing Systems," In Proc. Conf. on Parallel and Distributed Systems, 1999.
15. The Grid Forum, http://www.gridforum.org/
16. E. Hopper, B. C. H. Turton, An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem, Europian Journal of Operational Research, 2001.

A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources

Oliver Wäldrich[1], Philipp Wieder[2], and Wolfgang Ziegler[1]

[1] Fraunhofer Institute SCAI, Department of Bioinformatics,
53754 Sankt Augustin, Germany
{oliver.waeldrich, wolfgang.ziegler}@scai.fraunhofer.de
[2] Central Institute for Applied Mathematics,
Research Centre Jülich, 52425 Jülich, Germany
ph.wieder@fz-juelich.de

Abstract. The Grid paradigm implies the sharing of a variety of resources across multiple administrative domains. In order to execute a work-flow using these distributed resources an instrument is needed to co-allocate resources by reaching agreements with the different local scheduling systems involved. Apart from compute resources to execute the work-flow the co-ordinated usage of other resource types must be also guaranteed, as there are for example a network connectivity with dedicated QoS parameters or a visualisation device. We present a Web Service-based MetaScheduling Service which allows to negotiate a common time slot with local resource management systems to enable the execution of a distributed work-flow. The successful negotiation process results in a formal agreement based on the WS-Agreement recommendation that is currently specified by the GRAAP working group of the Global Grid Forum. As a use case we demonstrate the integration of this MetaScheduling Service into the UNICORE middleware.

1 Introduction

To successfully execute distributed applications or work-flows, usually different resources like compute nodes, visualisation devices, storage devices, or network connectivity with a defined QoS are required at the same time or in a certain sequence. Orchestrating such resources locally within one organisation represents only a minor task, whereas the orchestration of resources on a Grid level requires a service that is able to solve the same problems in an environment that may stretch across several administrative domains. Additional conditions have to be taken into account, like the compliance with site-specific policies or the protection of a site's autonomy.

In this paper we first describe in Section 2 an environment where co-allocation of resources is of vital importance, the requirements for the MetaScheduling Service that provides the required co-allocation means, and related work. In a next step we characterise the functionality of the MetaScheduling Service (Section 3), followed by a description of the current implementation. Then, in Section 5, we

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 782–791, 2006.

present the integration of the scheduling system into the UNICORE Grid middleware [1]. The performance of the the whole system is evaluated in Section 6, and the last section contains conclusions and an outlook to future work.

2 Scope, Requirements, and Related Work

The subject we are dealing with is scheduling and resource management in Grids and our primary focus is the co-allocation of different resources needed for the execution of a distributed application or a work-flow. Today's Grid-aware applications benefit more and more from using heterogeneous hardware that allows to optimise the performance of the applications. However, to make use of such distributed resources a tool or service is required that is capable of dealing with multiple policies for the operation and usage of resources. Furthermore, the scheduling systems that manage these resources will usually not be altered when the resources, in addition to local utilisation, are made available for usage in a Grid environment. Taking into account heterogeneity, site autonomy, and different site policies, a common approach for co-scheduling [2] of resources is not available so far. However, there have been several approaches over the last years ranging from commercial products like those from the Load Sharing Facility (LSF) family [3], to project-specific implementations like MARS [4] (a meta-scheduler for Campus Grids), GridFlow [5] (supporting work-flows in the DataGrid), or developments of the GrADS or the GriPhyN project. Other approaches like the one in Condor-G [6] or Legion's JobQueue [7] are limited to the underlying middleware environment. The Community Scheduler Framework (CSF) [8] is a global scheduler that interfaces with local schedulers such as OpenPBS [9], LSF or SGE [10], but it requires the obsolete Globus Toolkit 3.0 [11]. Finally there is GARA [12], an API supporting co-scheduling of, inter alia, network resources in a Globus middleware environment.

All the approaches mentioned above are to some extent limited and not suitable to be adopted to the needs of a common meta-scheduling service, because of

- their commercial nature,
- their limited support for different types of resources,
- their restriction to the needs of a specific project,
- their limitation to a particular middleware environment, and
- the use of proprietary protocols.

The work presented in the following sections of the paper tries to overcome the limitations by providing an extensible service that is not restricted to the needs of a particular project, that implements evolving standards, and that is able to support arbitrary types of resources.

3 Required Functionality of the MetaScheduling Service

To achieve co-allocation of resources managed by multiple, usually different scheduling systems, the minimal requirement these systems have to fulfil is to provide functions to

1. schedule a single reservation some time in the future (e.g. "from 5:00 pm to 8:00 pm tomorrow") and to
2. give an aggregated overview of the usage of the managed resources between now and a defined time in future.

Once a reservation is scheduled the starting time for it is fixed, i.e. it may not change except for the reservation being cancelled. This feature is called advance reservation. There are at least two possibilities to realise such advance reservation. The first possibility is to schedule a reservation for a requested time, called fixed time scheduling. The second possibility is to schedule a reservation not before a given time, which means a scheduling system tries to place the reservation at the requested time, otherwise it will be scheduled for the earliest possible time after the one requested. This results in a first fit reservation. The implementation of the MetaScheduling Service described in this paper interacts with, but is not limited to, scheduling systems that implement the first fit behaviour. The main function of the MetaScheduling Service is to negotiate the reservation of network-accessible resources that are managed by their respective local scheduling systems. The goal of the negotiation is to determine a common time slot where all required resources are available for the requested starting time of the job. The major challenges for a meta-scheduler are

- to find Grid resources suitable for the user's request,
- to take security issues like user authentication and authorisation into account,
- to respect the autonomy of the sites offering the resources, and
- to cope with the heterogeneity of the local scheduling systems.

We do not address the problem of finding suitable resources here, this task is usually delegated to a Grid information system or a resource broker. Security issues are only considered here with respect to user authentication and authorisation as the MetaScheduling Service has to reserve on behalf of the user's respective identity at the sites that he wants to use resources from. The implementation described in the next section addresses both site autonomy and heterogeneity.

4 Implementation of the MetaScheduling Service

To interact with different types of scheduling systems we decided to use the adapter pattern approach. The role of such adapters is to provide a single interface to the MetaScheduling Service by encapsulating the specific interfaces of the different local scheduling systems. Thus the MetaScheduling Service can negotiate resource usage using a single interface. The adapters are connected to both the MetaScheduling service and the scheduling systems and may therefore be installed either at the site hosting the MetaScheduling Service, the (remote) sites where the scheduling systems are operated, or at any other system accessible through a network. Currently adapters are available for two scheduling systems:

EASY and a proprietary scheduling system. Two more adapters will be be available late summer 2005: one for the Portable Batch System Professional (PBS Pro) and another one for ARGON, the resource management system for network resources that is currently under development in the VIOLA project [16].

4.1 Negotiation Protocol

In this section we describe the protocol the MetaScheduling Service uses to negotiate the allocation of resources with the local scheduling systems. This protocol is illustrated in Fig. 1. The user specifies the duration of the meta-job and additionally - for each subsystem - reservation characteristics like the number of nodes of a cluster or the bandwidth of the connections between nodes. In the UNICORE based VIOLA testbed the UNICORE client is used to describe the request of the user. The client sends the job description to the MetaScheduling Service using the Web Services Agreement (WS-Agreement) protocol [14]. Based on the information in the agreement the MetaScheduling Service starts the resource negotiation process:

1. The MetaScheduling Service queries the adapters of the selected local systems to get the earliest time the requested resources will be available. This time possibly has to be after an offset specified by the user.
2. The adapters acquire previews of the resource availability from the individual scheduling systems. Such a preview comprises a list of time frames during which the requested QoS (e.g. a fixed number of nodes) can be provided. It is possible that the preview contains only one entry or even zero entries if the resource is fully booked within the preview's time frame. Based on the preview the adapter calculates the possible start-time.

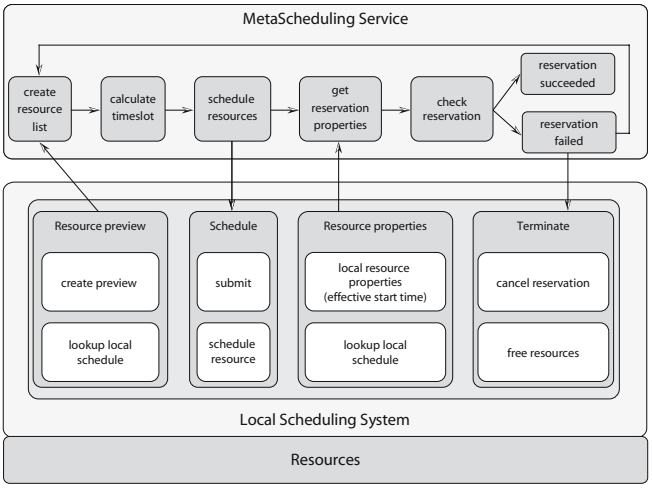


Fig. 1. The negotiation process

3. The possible start times are sent back to the MetaScheduling Service.
4. If the individual start times do not allow the co-allocation of the resources, the MetaScheduling Service uses the latest possible start time as the earliest start time for the next scheduling iteration. The process is repeated from step 1. until a common time frame is found or the end of the preview period for all the local systems is reached. The latter case generates an error condition.
5. In case the individual start times match, the MetaScheduling Service checks the scheduled start times for each reservation by asking the local schedulers for the properties of the reservation. This step is necessary because in the meantime new reservations may have been submitted by other users or processes to the local schedulers, preventing the scheduling of the reservation at the requested time.
6. If the MetaScheduling Service detects one or more reservations that are not scheduled at the requested time, all reservations will be cancelled. The latest effective start time of all reservations will be used as the earliest start time to repeat the process beginning with step 1.
7. If all reservations are scheduled for the appropriate time the co-allocation of the resources has been completed.
8. The IDs of the MetaScheduling Service and the local reservations are added to the agreement and a reference to the agreement is sent back to the UNICORE client.

4.2 Hosting Environment and Command Line Interface

For the hosting environment of the MetaScheduling Service we use Sun Java 1.3.1 JRE with Apache Tomcat 3.3.2 as a servlet engine. In addition we make use of the Apache SOAP Framework 2.3 for sending and receiving SOAP [13] messages. The interface between the adapter and the local scheduling systems is implemented as a simple CGI module, which is hosted at the Apache web server and wraps the local scheduling system's commands. The communication between Apache and the adapters is secured by using HTTPS.

A simple Command Line Interface (CLI) is also available to access the MetaScheduling Service. It enables users to submit a meta-scheduling request, to query job details related to a certain reservation, and to cancel a given reservation. To submit a request a user may specify the duration of the reservation, the resources needed per reservation (number of nodes, network connectivity, ...), and the executable for each site.

The current CLI implementation does not contain integrated security means to provide single sign-on, as e.g. the UNICORE client does. Instead the CLI user has to enter a valid login and password for every requested resource. The credentials are stored by the CLI, passed to the MetaScheduling Service, and used for authentication at the local systems. Please note that this is an interim solution and that the meta-scheduling framework will provide a concise security solution at a later implementation stage.

4.3 Interface Between MetaScheduling Service and Adapter

The interface/protocol between the MetaScheduling Service and the adapters is implemented using Web Services and it basically provides the four functions `couldRunAt()`, `submit()`, `state()`, and `cancel()`.

These functions cover the whole negotiation process described in Section 4.1. The Web Services interface/protocol as implemented is not standardised, but the GRAAP [15] working group at the Global Grid Forum (GGF) is currently discussing to address the standardisation of a protocol for the negotiation process. Once such a recommendation is available we plan to implement it instead of the current one.

4.4 Implementation of the Agreement

The development of the MetaScheduling Service includes, as already mentioned in Section 4.1, an implementation of the WS-Agreement specification [14]. The client submits a request using an agreement offer (a template provided by the MetaScheduling Service and filled in with the user's requirements by the client) to the MetaScheduling Service. It then negotiates the reservation of the resources in compliance with the QoS defined in the agreement template. As a result of the successfully completed negotiation process a valid agreement is returned to the client, containing the scheduled end times of the reservation, the reservation identifier, and further information.

5 Integration into the UNICORE Middleware

The first generation meta-scheduler architecture developed in the VIOLA project [16] focuses on the scheduling functionality requiring only minimal changes to

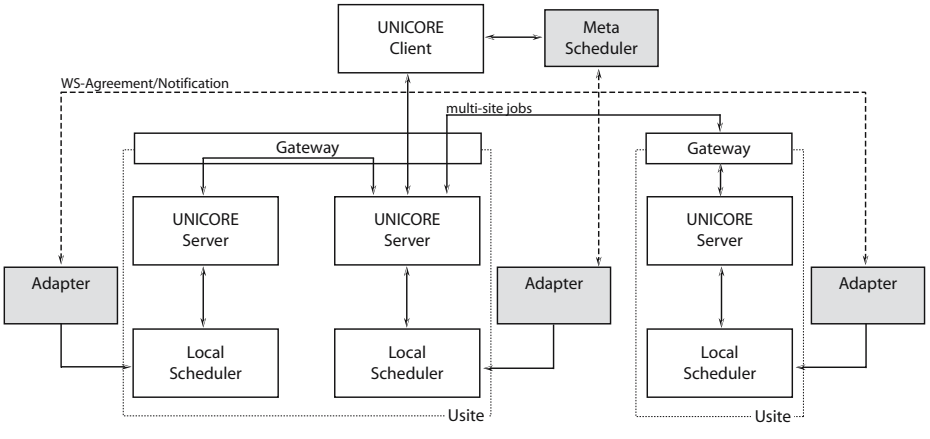


Fig. 2. The meta-scheduling architecture

the UNICORE system (please refer to [1] for an in-depth description of the UNICORE models and components). As depicted in Fig. 2, the system comprises the adapter, the MetaScheduling Service itself [17], and a MetaScheduling plug-in (which is part of the client and not pictured separately). Before submitting a job to a UNICORE Site (Usite), the MetaScheduling plug-in and the MetaScheduling Service exchange the data necessary to schedule the resources needed. The MetaScheduling Service is then (acting as an Agreement Consumer in WS-Agreement terms [14]) contacting the adapter (acing as an Agreement Manager) to request a certain level of service, a request which is translated by the Manager into the appropriate local scheduling system commands. In case of VIOLA's computing resources the targeted system is the EASY scheduler. Once all resources are reserved at the requested time the MetaScheduling Service notifies the UNICORE Client via the MetaScheduling plug-in to submit the job. This framework will also be used to schedule the interconnecting network, but potentially any resource can be scheduled if a respective adapter/Agreement Manager is implemented and the MetaScheduling plug-in generates the necessary scheduling information. The follow-on generation of the meta-scheduling framework will then be tightly integrated into UNICORE.

6 Experiences and Performance Evaluation

We set up different tests to evaluate the performance of the MetaScheduling Service. All tests were performed on a PC Cluster where multiple local scheduling systems and MetaScheduling services were deployed across the compute nodes. We executed the following two types of performance tests:

1. Reservations with multiple adapter instances (Fig. 3).
2. Reservations with multiple MetaScheduling Service instances (Fig. 4).

In the first test series we used a single MetaScheduling Service instance to co-allocate resources across six different local schedulers. This implied that six partial reservations were generated for six different local scheduling instances that were randomly chosen from a pool of 160 instances. The queue of each local scheduler was randomly initialised to simulate a certain load per system. In the second test series we realised a layered approach: The MetaScheduling Service that received the reservation request from the user (the topMSS) delegated the reservations to other MetaScheduling Services which then negotiated with the local schedulers. As in the first test a reservation consisted of six partial reservations. Apart from the topMSS three MetaScheduling Services were used, each being responsible for the co-allocation of resources managed by two local schedulers. The topMSS instance negotiated the complete reservation with these three instances which for the topMSS take the role of adapters as in the first test using the same protocol for the inter-MetaScheduling Service communication.

Both tests indicate that the co-allocation approach selected is reasonable: a complete schedule is delivered in less than 20 seconds (30 seconds in the case

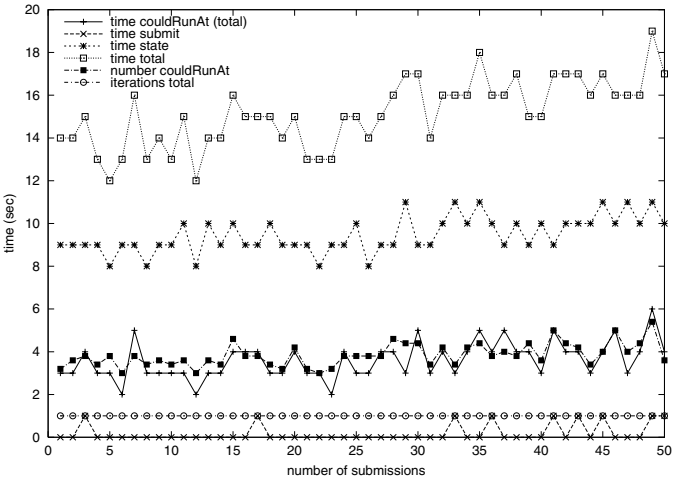


Fig. 3. Performance of a MetaScheduling Service negotiating with multiple adapters

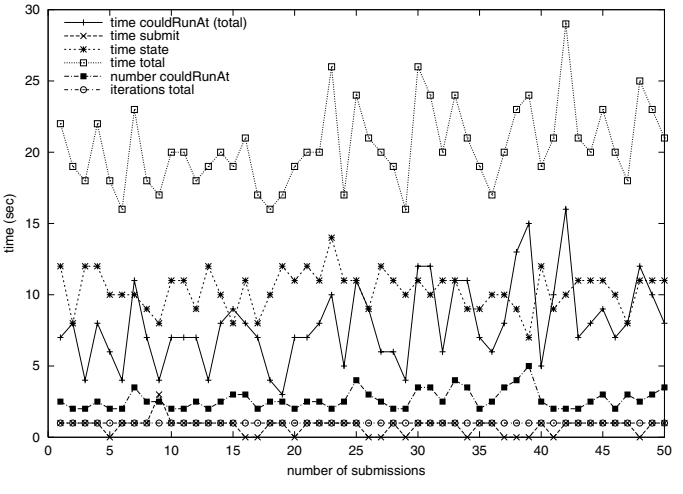


Fig. 4. Performance of a MetaScheduling Service negotiating with multiple MetaScheduling Services

of layered MetaScheduling Services). Fig. 3 and 4 indicate a slight increase of the total reservation time in case the local systems' load increases and thus the number of lookups needed for identifying free slots and for scheduled reservations also increases. The total number of iterations indicates how often the co-allocation process had to be restarted due to reservation inconsistencies at the local schedulers (see Section 4.1, step 7.). In the two tests series this value was constant since there were no additional scheduling activities (apart from those initiated by the MetaScheduling Service) carried out that could interfere.

7 Conclusions and Future Work

In this paper a meta-scheduling framework has been described which is capable of co-allocating arbitrary types of resources by means of adapters that abstract the different interfaces of the local scheduling systems. An implementation has been presented that may be used via a command line interface or integrated into a Grid middleware. As a use case a first approach to an WS-Agreement-based integration into the UNICORE Grid system has been shown. Future work will concentrate on several improvements of the service and focus, e.g. on:

- implementing adapters for additional scheduling systems,
- implementing the WS-Negotiation protocol as it is evolving at the GGF,
- integrating the MetaScheduling Service into OGSA/WSRF-based UNICORE [18] and Globus [19] systems,
- providing resource broker capabilities,
- integrating the Intelligent Scheduling System (ISS) for selection of resources best fitted to execute an specific application [21], and
- integrating a Grid Scheduling Ontology [20].

Acknowledgements

Some of the work reported in this paper is funded by the German Federal Ministry of Education and Research through the VIOLA project under grant #123456. This paper also includes work carried out jointly within the CoreGRID Network of Excellence funded by the European Commission's IST programme under grant #004265.

References

1. D. Erwin, ed. *UNICORE Plus Final Report – Uniform Interface to Computing Resources*. UNICORE Forum e.V., ISBN 3-00-011592-7, 2003.
2. J. Schopf. Ten Actions when Grid Scheduling. In *Grid Resource Management* (J. Nabrzyski, J. Schopf and J. Weglarz, eds.), pages 15-23, Kluwer Academic Publishers, 2004.
3. Load Sharing Facility, Resource Management and Job Scheduling System. Web site, 2005. Online: http://www.platform.com/Products/Platform.LSF.Family/.
4. A. Bose, B. Wickman, and C. Wood. MARS: A Metascheduler for Distributed Resources in Campus Grids. In *5th International Workshop on Grid Computing (GRID 2004)*. IEEE Computer Society, 2004.
5. J. Weinberg, A. Jagatheesan, A. Ding, M. Faerman, and Y. Hu. Gridflow: Description, Query, and Execution at SCEC using the SDSC Matrix In *13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)* IEEE Computer Society, 2004.
6. D. Thain and M. Livny. Building Reliable Clients and Servers. In *The Grid: Blueprint for a New Computing Infrastructure* (I. Foster and C. Kesselman, eds.), Morgan Kaufmann, 2003.

7. D. Katramatos, M. Humphrey, A. S. Grimshaw, and S. J. Chapin. JobQueue: AComputational Grid-Wide Queueing System. In *Grid Computing - GRID 2001, Second International Workshop*, Volume 2242 of Lecture Notes in Computer Science, Springer, 2001.
8. Community Scheduler Framework. Web site, 2005. Online: http://sourceforge.net/projects/gscf.
9. OpenPBS Batch Processing and Resource Management System. Web site, 2005. Online: http://www.openpbs.org/.
10. Sun Grid Engine.Web site, 2005. Online: http://www.sun.com/software/gridware.
11. The Globus Toolkit 3.0. Web site, 2005. Online: http://www.globus.org/toolkit/downloads/3.0/.
12. I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. In *8th International Workshop on Quality of Service (IWQOS 2000)*, pages 181-188, June, 2000.
13. Simple Object Access Protocol Specification, SOAP Specification version 1.2. Web site, 2005. Online: http://www.w3.org/TR/soap12/.
14. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web Services Agreement Specification.* Grid Forum Draft, Version 2005/09, Global Grid Forum, September, 2005.
15. Grid Resource Allocation Agreement Protocol Working Group. Web site, 2005. Online: https://forge.gridforum.org/projects/graap-wg/.
16. VIOLA – Vertically Integrated Optical Testbed for Large Application in DFN. Web site, 2005. Online: http://www.viola-testbed.de/.
17. G. Quecke and W. Ziegler. MeSch – An Approach to Resource Management in a Distributed Environment. In *Proc. of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000)*, Volume 1971 of Lecture Notes in Computer Science, pages 47-54, Springer, 2000.
18. R. Menday and Ph. Wieder. GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In *Proc. of the 3rd Cracow Grid Workshop (CGW'03)*, Cracow, PL, Oct. 27-29, 2003.
19. The Globus Toolkit 4.0. Project documentation web site, 2005. Online: http://www.globus.org/toolkit/docs/4.0/.
20. Ph. Wieder and W. Ziegler. Bringing Knowledge to Middleware - The Grid Scheduling Ontology. In *First CoreGRID Workshop*, Springer, 2005, to appear.
21. V. Keller, K. Cristiano, R. Gruber, P. Kuonen, S. Maffioletti, N. Nellari, M.-C. Sawley, T.-M. Tran, Ph. Wieder, and W. Ziegler. Integration of ISS into the VIOLA Meta-scheduling Environment. In *Proc. of the Integrated Research in Grid Computing Workshop*, Pisa, IT, November 28-20, 2005, to appear.

Comparison of Workflow Scheduling Strategies on the Grid*

Marek Wieczorek, Radu Prodan, and Thomas Fahringer

University of Innsbruck,
Institute of Computer Science,
Technikerstraße 21a, A-6020 Innsbruck, Austria
marek@dps.uibk.ac.at

Abstract. Scheduling is a key concern for the execution of performance-driven Grid applications. In this paper we comparatively examine different existing approaches for scheduling of scientific workflow applications in a Grid environment. We evaluate three algorithms namely genetic, HEFT, and simple "myopic" and compare incremental workflow partitioning against the full-graph scheduling strategy. We demonstrate experiments using real-world scientific applications covering both balanced (symmetric) and unbalanced (asymmetric) workflows. Our results demonstrate that full-graph scheduling with the HEFT algorithm performs best compared to the other strategies examined in this paper.

1 Introduction

Scheduling of scientific workflow applications on the Grid is a challenging problem, which is an ongoing research effort followed by many groups. Deelman [7] distinguishes several workflow processing strategies covering trade-offs between dynamicity and look-ahead range in workflow processing. In [3] Deelman proposed a scheduling strategy based on initial partitioning of the workflow into sequential subworkflows, that are scheduled sequentially one after another. Prodan [8] applied genetic algorithms to schedule the whole workflow at once, and rescheduling it many times during the execution. These approaches were not compared against each other.

In this paper we examine three scheduling algorithms to evaluate their performance for scheduling scientific workflows in Grid environments. The scheduling algorithms comprise a genetic algorithm similar to the one presented in [8], the well-known HEFT algorithm [12], and a "myopic" algorithm. HEFT is a simple and computationally inexpensive algorithm, which schedules workflows by creating an ordered list of tasks out of the workflow, and mapping the tasks to the resources in the most appropriate way. The last algorithm we applied is a simple "myopic" algorithm, similar to the Condor DAGMan [10] resource broker, which schedules the next task onto the best machine available without any long-term

* The work described in this paper is supported by the European Union through the IST-2002-511385 project K-WfGrid.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 792–800, 2006.

optimization strategy. The Grid model applied by us in the experiments assumes high availability rate and good control over the resources by the scheduler.

Additionally, we compared different scheduling strategies including full graph scheduling and incremental workflow partitioning strategy [3]. The Myopic algorithm can be considered as a just-in-time scheduling strategy, as the scheduling decisions made by the algorithm are optimized for the current time instance.

In the remainder of this paper, we evaluate the scheduling approaches through a series of experiments. We show that the HEFT algorithm applied with the full-ahead scheduling strategy performs best compared to other approaches evaluated by us. We also prove the efficacy of this approach for a specific class of strongly unbalanced (asymmetric) workflows.

2 Experimental Environment

The scheduling experiments have been performed in ASKALON environment [5], which a Grid environment for scientific workflow applications. The environment consists of many components responsible for composition and execution of the applications. The enactment engine supervises the reliable and fault tolerant execution of the tasks and transfer of the files. The resource broker and the performance predictor are auxiliary services which provide information about the resources available on the Grid, and predictions about expected execution times and data transfer times.

Scientific workflows executed in the ASKALON environment are based on the model described in the AGWL specification language [11]. AGWL documents can express simple DAGs as well as more sophisticated workflow graphs containing loops and conditional branches which impose control flow decisions that can only be decided at runtime. Special workflow structures (conditional branches *if-then* or *switch*, loops, or parallel loops) can be evaluated in various ways for different executions. To cope with such uncertainties the scheduler makes assumptions about the actual evaluation of the workflow, and makes the schedule for the graph predicted as result of such assumption. If an assumption fails, the scheduler transforms the workflow once again in the proper way and reschedules it. This approach may bring about considerable benefit if the structure of the workflow is predicted correctly (especially, when a strong unbalance in the workflow is detected). If the conditions are predicted incorrectly, the workflow execution time is the same as in the case of a just-in-time strategy which schedules only those parts of the workflow that are resolved at the moment of scheduling. Fig. 2(a)- 2(d) show two such workflow transformations applied to real Grid workflow applications (see Section 5).

3 Scheduling Algorithms

The scheduling algorithms under consideration map tasks as part of workflows onto Grid sites (clusters). Each Grid site consists of a set of CPUs, each of which is considered as a single computational resource. If a task is executed on a CPU,

no other tasks can use the same CPU at the same time. Execution times of tasks and data transfers generated by the performance predictor are given as input data to the scheduler.

3.1 HEFT Algorithm

The HEFT algorithm that we applied consists of 3 phases:

1. *Weighting* assigns the weights to the nodes and edges in the workflow;
2. *Ranking* creates a sorted list of tasks, organized in the order how they should be executed;
3. *Mapping* assigns the tasks to the resources.

The weights assigned to the nodes are calculated based on the predicted execution times of the tasks. The weights assigned to the edges are calculated based on predicted times of the data transferred between the resources. In heterogeneous environments, the weights must be approximated considering different predictions for execution times on different resources, and for different data transfer times on different data links. Several approximation methods were proposed and compared [9]. Each of them provides different accuracy for different cases. We chose the arithmetic average.

The ranking phase is performed traversing the workflow graph upwards, and assigning a rank value to each of the tasks. Rank value is equal to the weight of the node plus the execution time of the successors. The successor execution time is estimated, for every edge being immediate successor of the node, adding its weight to the rank value of the successive node, and choosing the maximum of the summations. A list of resources is arranged, according to the decreasing rank values. In the mapping phase, consecutive tasks from the ranking list are mapped to the resources. For each task, the resource which provides the earliest expected time to finish execution is chosen.

3.2 Genetic Algorithms

The idea of the Genetic Algorithms is to encode possible solutions of the problem into a *population* of *chromosomes*, and subsequently to transform the population using standard operations of *selection*, *crossover* and *mutation*, producing successive *generations*. The selection is driven by an established *fitness function*, which evaluates the chromosomes in terms of accuracy of the represented solutions. Crossover and mutation respond to standard biological operations of mutual exchange of a part of body within a pair of chromosomes, and of change of some elements (so-called *genes*) in the chromosomes randomly selected from the population. The end condition of a genetic algorithm is usually the *convergence criterion* which checks how much the best individual found changes between subsequent generations. A maximum number of generations can also be established.

Genetic Algorithms are a good general purpose heuristic, which is able to find the optimal solution even for complicated multi-dimensional problems. The

challenge might be to correctly encode solutions of the problem into the chromosomes, which is not always feasible. Prodan in [8] encoded the actual mapping of tasks to the resources without specifying the order of execution of independent tasks (not linked through control and data flow dependencies) that are scheduled on the same CPU. Therefore this execution order cannot be a subject to optimization. Moreover, Genetic Algorithms tend to be computationally extensive.

3.3 Myopic Algorithm

To compare the scheduling algorithms described so far, we developed a simple and inexpensive scheduling algorithm, which makes the planning based on locally optimal decisions. In analyses every task separately, traversing the workflow in the top-down direction, and assigns it to the resource which seems to be the most optimal in the given moment. The algorithm represents a class of schedulers covering for instance the Condor DAGMan resource broker which employs the matchmaking mechanism [10]. The Myopic algorithm can produce reasonably accurate results for rather simple workflows given accurate performance predictions. But it does not provide any full-graph analysis and does not consider the order of task execution.

4 Scheduling Strategies

Different scheduling strategies [7] can be applied, considering the trade-off between dynamicity and look-ahead range in workflow processing. At one extreme we have *just-in-time* strategy, in [7] referred to as *in-time local scheduling*, consisting in mapping the tasks to the resources, always choosing the most appropriate solution for the current step. At the other extreme, we have *full-ahead planning* where the full-graph scheduling is performed at the beginning of execution. Those two strategies represent the most dynamic and the most static approach, respectively. Intermediate solutions try to reconcile workflow planning with Grid dynamism, and to find an approach which considers both the work flow structure and the Grid behavior. One of the possible solutions is the workflow partitioning applied in the Pegasus system [3]. In this approach the workflow is initially partitioned into a sequence of subworkflows, which are subsequently scheduled and executed. Each partitioning can be characterized by the width of a slice. The width of a slice is expressed as maximal number of node layers within each slice. Any element of the sequence can be scheduled and executed only if the immediate predecessor has already finished its execution.

Full-graph scheduling, however, can also be applied in a dynamic way, if we do not consider the initial scheduling as the ultimate decision but we allow rescheduling of the workflow during its execution. One of the workflows we applied in our experiments belongs to a specific class of strongly unbalanced workflows, which seems to require full-graph analysis for proper scheduling. The workflow contains a parallel section and some of the branches take longer to execute than the others (see Fig. 2(d)). The tasks that belong to the longer

branch should, therefore, execute with higher priority than the ones that belong to the shorter branches. One important goal of our experiments was to investigate how the scheduling results depend on the workflow strategy applied for strongly unbalanced workflows.

In our experiments we assumed high availability rate and good control over the resources. In particular, we assumed that the scheduler always has precise information about the resources, so that the execution of the workflow is performed in the same way as it was planned by the scheduler.

5 Experimental Results

In our experiments we compare the HEFT algorithm with a genetic algorithm similar to the one proposed in [8], and with the Myopic algorithm described in Section 3.3. We also compare the full-graph scheduling with the workflow partitioning strategy. As results, we show execution times of the scheduled workflow applications (*execution times*), and the times spent in preparing the schedules (*scheduling times*). The execution times were measured for two scenarios of workflow execution. In the first scenario, we do not provide to the scheduler any performance predictions, so the scheduler has to assume that all the execution times are equal for all tasks on all the resources (scheduling *without performance guidance*). In the second scenario, the scheduler is provided with experience-based performance predictions derived from historical executions (scheduling *with performance guidance*). The predictions were provided to the scheduler in a two-dimensional array, containing the execution time of each task on each computer architecture available in our Grid. The assumption was that each task takes the same execution time on every machine that belongs to the same type (i.e, has the same CPU model, CPU speed and total RAM size).

Experiments were performed incorporating seven Grid sites (clusters) of the Austrian Grid [2] infrastructure with 116 CPUs in total (not all Grid sites were used in all the experiments). In Fig. 1 we present the performance of the individual clusters, where each cluster shows the average execution time of all the workflow tasks executed on a single CPU. As we can see, the fastest cluster is more than three times faster than the slowest one.

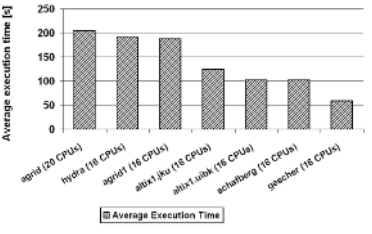
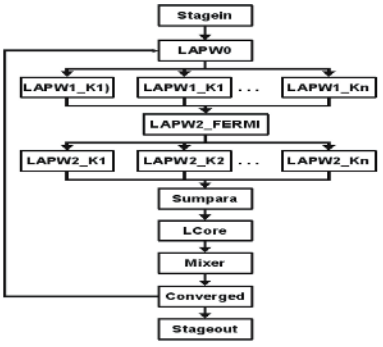
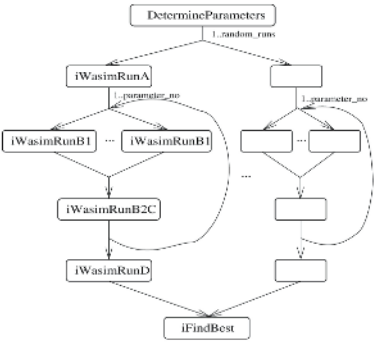


Fig. 1. Performance of Grid clusters used in the experiments

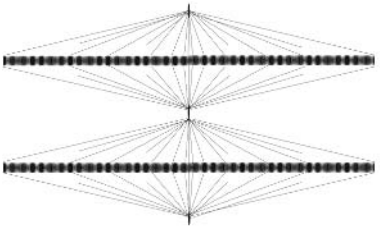
Similarly to execution time predictions, the execution times of the tasks on the sites were measured on the Austrian Grid during a test phase. Time consumed by data transfers between two tasks connected with a data link was considered as constant. We also fixed the middleware overhead introduced by the Globus GSI security [6] and the queuing system to 30 seconds.



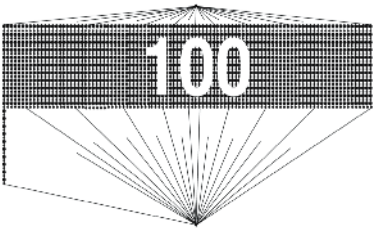
(a) WIEN2k, original workflow



(b) Invmod, original workflow



(c) WIEN2k, transformed workflow

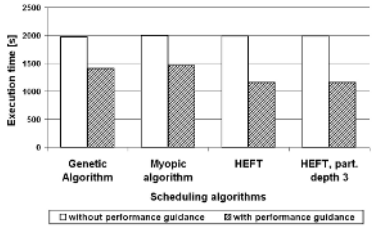


(d) Invmod, transformed workflow

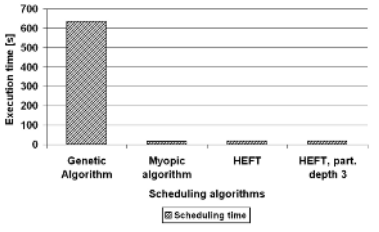
Fig. 2. Workflows of the applications

In our experiments we used two real-world applications, WIEN2k [1] and Invmod [4]. WIEN2k workflow (Fig. 2(a)) is a fully-balanced workflow which contains two parallel sections with possibly many parallel tasks, and an external loop. For our tests we considered the workflow with one iteration of the loop, and 250 parallel tasks in each section (Fig. 2(c)). We used the Invmod workflow (Fig. 2(b)) to simulate the common case of strongly unbalanced workflow. If the loops in individual workflow threads have different numbers of iterations (and the iteration numbers are predicted correctly), then the threads may differ significantly with regard to their expected execution times. The workflow used for these experiments (Fig. 2(d)) contains 100 parallel iterations one of which contains 20 iterations of the optimization loop. The remaining 99 iterations contain 10 optimization iterations each. It means, that one of the threads takes approximately twice as much execution time as all others.

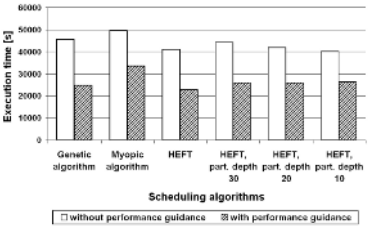
The genetic algorithm that we applied is based on the population of 100 chromosomes transformed in 20 generations. Probability of crossover was fixed by us to 0.25, and mutation rate to 0.01. We performed workflow partitioning by dividing the workflow into slices with well-defined width (see Section 4). For the WIEN2k workflow (consisting of five layers) we applied a three-layer partioning, and for Invmod workflow (which consists of 44 layers) we applied three different partitionings, with 10, 20 and 30 layers.



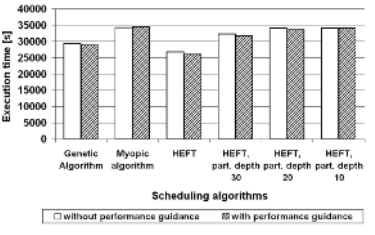
(a) WIEN2k executed in heterogeneous environment, execution time



(b) WIEN2k executed in heterogeneous environment, scheduling time



(c) Invmod executed in heterogeneous environment, execution time



(d) Invmod executed in homogeneous environment, execution time

Fig. 3. Experimental results

The first conclusion we draw from the results (Fig. 3(a)-3(c)) is that performance prediction is very important in heterogeneous Grid environments. For both workflows, the results achieved with performance guidance are in the best case nearly two times better that the results achieved without performance guidance. Performance estimates are clearly important even if they are not highly accurate.

Comparing the results measured for the WIEN2k workflow we can notice that HEFT produces much better results than the other algorithms. Execution time of the workflow is 17% shorter than for the genetic algorithm, and even 21% than for the Myopic. The simple solution applied in Myopic appears to be insufficient for large and complicated workflows, and the algorithm produces the worst results. Also the genetic algorithm appears to be not a good method to deal with our problem. Comparing the scheduling times of individual algorithms we can see that the genetic algorithm executes two to three orders of magnitude longer than the others.

The results measured for the Invmod workflow present how individual algorithms deal with strongly unbalanced workflows. As expected, the Myopic algorithm provides the worst results of all, approximately 32% worse than HEFT. The genetic algorithm produces quite good results, however it was not able to find the best possible solution, since the order of execution (of independent tasks scheduled to the same CPU) was not considered for optimization. In the workflows scheduled without an established task order, the tasks are executed in an

arbitrary order chosen by the runtime system. For a strongly unbalanced workflow, however, the runtime system has to be guided to assign higher priority to the tasks that execute in iterations of the parallel loop with longer execution times. Scheduling strategies based on the workflow partitioning were also not able to find the optimal solution, although their results are still better than the one found by the Myopic algorithm. Since all of the algorithms (except for the genetic algorithm) execute really fast (less than 20 seconds for large and complicated workflows), there is no reason to apply the partitioning strategy in place of the full-graph analysis.

Fig. 3(d) presents the execution results of the Invmod workflow on a homogeneous environment (three nearly identical Grid sites). As expected, there is now almost no difference between the scheduling with and without performance guidance, as the execution on each cluster takes the same time. Again, HEFT produces the best results, 24% better than Myopic.

6 Conclusions

Scheduling applications on the Grid is of paramount importance to optimize non-functional parameters such as execution time. In this paper we compared three different algorithms examining aspects such as incremental versus full-graph scheduling for balanced versus unbalanced workflows.

Based on two real world Grid workflows we observed that the HEFT algorithm appears to be a good and computationally inexpensive scheduling algorithm that performs better than the other 2 candidates discussed in this paper.

We also investigated a specific class of strongly unbalanced workflows. We demonstrated that any just-in-time scheduling strategy is likely to produce poor results for workflows of this class. Also the workflow partitioning strategy used in Pegasus system [3] appears to have no advantage over the full-graph scheduling, and may produce less efficient results for unbalanced workflows.

We implemented the HEFT algorithm in the ASKALON environment for scheduling scientific workflow applications on the Grid. In the future we will try to make our strategy more efficient for heterogeneous environments. We will also examine more carefully different network issues and their influence on the performance-oriented scheduling.

References

1. P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. *WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties.* Institute of Physical and Theoretical Chemistry, Vienna University of Technology, 2001.
2. The Austrian Grid Consortium. http://www.austriangrid.at.
3. Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20, 2004.

4. Peter Rutschmann Dieter Theiner. An inverse modelling approach for the estimation of hydrological model parameters. In *Journal of Hydroinformatics*, 2005.
5. Rubing Duan, Thomas Fahringer, Radu Prodan, Jun Qin, Alex Villazon, and Marek Wieczorek. Real World Workflow Applications in the Askalon Grid Environment. In *European Grid Conference (EGC 2005)*, Lecture Notes in Computer Science. Springer Verlag, February 2005.
6. GT 4.0 Security website. http://www.globus.org/toolkit/docs/4.0/security/.
7. Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz. *Grid Resource Management, State of the Art and Future Trends*. Kluwer, 2003.
8. Radu Prodan and Thomas Fahringer. Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study. In *20th Symposion of Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005. ACM Press.
9. Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *IPDPS*, 2004.
10. The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.
11. Jun Qin Thomas Fahringer and Stefan Hainzer. Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*, Cardiff, UK, May 9-12 2005. IEEE Computer Society Press.
12. Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par*, pages 189–194, 2003.

A Web Computing Environment for Parallel Algorithms in Java*

Olaf Bonorden, Joachim Gehweiler, and Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute, Computer Science Departement,
Paderborn University, 33095 Paderborn, Germany,
{bono, joge, fmadh}@uni-paderborn.de

Abstract. We present a web computing library (PUBWCL) in Java that allows to execute strongly coupled, massively parallel algorithms in the bulk-synchronous (BSP) style on PCs distributed over the internet whose owners are willing to donate their unused computation power.

PUBWCL is realized as a peer-to-peer system and features migration and restoration of BSP processes executed on it.

The use of Java guarantees a high level of security and makes PUBWCL platform independent. In order to estimate the loss of efficiency inherent in such a Java-based system, we have compared it to our C-based PUB-Library.

1 Introduction

Motivation. Bearing in mind how many PCs do exist distributed all over the world, one can easily imagine that all their idle times together represent a huge amount of unused computation power. There are already several approaches geared to utilize this unused computation power (e. g. *distributed.net* [6], *Great Internet Mersenne Prime Search (GIMPS)* [8], *Search for Extraterrestrial Intelligence (SETI@home)* [15]). A common characteristic of most of these approaches is that the computational problem to be solved has to be divided into many small subproblems by a central server; clients on all the participating PCs download a subproblem, solve it, send the results back to the server, and continue with the next subproblem. Since there is no direct communication between the clients, only independent subproblems can be solved by the clients in parallel.

Our contribution. We have developed a web computing library (PUBWCL) that removes this restriction; in particular, it allows to execute strongly coupled, massively parallel algorithms in the bulk-synchronous (BSP) style on PCs distributed over the internet. PUBWCL is written in Java to guarantee a high level of security and to be platform independent. PUBWCL furthermore features migration and restoration of BSP processes executed on it in order to provide for load balancing and to increase fault tolerance. A systematic evaluation of load balancing strategies for PUBWCL is described in [10].

* Partially supported by DFG-SFB 376 "Massively Parallel Computation".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 801–808, 2006.

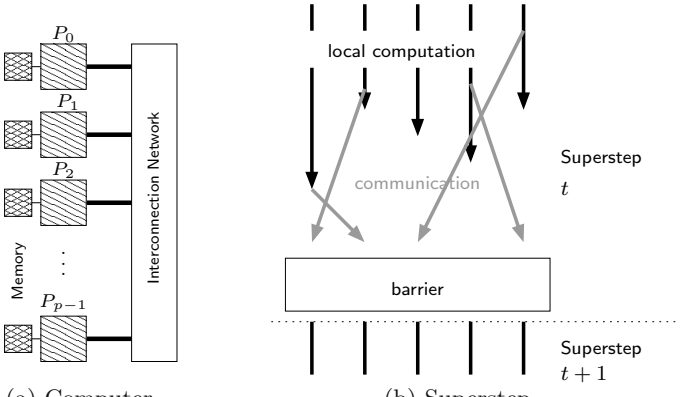
Related work. Like PUBWCL, *Oxford BSPlib* [3] and *Paderborn University BSP Library (PUB)* [12, 13] are systems to execute strongly coupled, massively parallel algorithms according to the BSP model (see Section 2). They are written in C and are available for several platforms. These BSP libraries are optimized for application on monolithic parallel computers and clusters of workstations. These systems have to be centrally administrated, whereas PUBWCL runs on the internet, taking advantage of Java's security model and portability.

The *Bayanihan* BSP implementation [2] follows the master-worker-paradigm: The master depacketizes the BSP program to be executed into pieces of work, each consisting of one superstep in one BSP process. The workers download a packet consisting of the process state and the incoming messages, execute the superstep, and send the resulting state together with the outgoing messages back to the master. When the master has received the results of the current superstep for all BSP processes, it moves the messages to their destination packets. Then the workers continue with the next superstep. With this approach all communication between the BSP processes passes though the server, whereas the BSP processes communicate directly in PUBWCL.

Organization of paper. The rest of the paper is organized as follows: In Sections 2 and 3, we give an overview of the used parallel computing model and the Java thread migration mechanism. In Sections 4 and 5, we describe our web computing library and evaluate its performance. Section 6 concludes this paper.

2 The BSP Model

In order to simplify the development of parallel algorithms, Leslie G. Valiant has introduced the *Bulk-Synchronous Parallel (BSP)* model [18] which forms a bridge between the hardware to use and the software to develop. It gives the developer an abstract view of the technical structure and the communication features of



(a) Computer (b) Superstep

Fig. 1. BSP model

the hardware to use (e. g. a parallel computer, a cluster of workstations or a set of PCs interconnected by the internet).

A *BSP computer* is defined as a set of processors with local memory, interconnected by a communication mechanism (e. g. a network or shared memory) capable of point-to-point communication, and a barrier synchronization mechanism (cf. Fig. 1 (a)).

A *BSP program* consists of a set of *BSP processes* and a sequence of *supersteps* – time intervals bounded by the barrier synchronization. Within a superstep each process performs local computations and sends messages to other processes; afterwards it indicates by calling the sync method that it is ready for the barrier synchronization. When all processes have invoked the sync method and all messages are delivered, the next superstep begins. Then the messages sent during the previous superstep can be accessed by its recipients. Fig. 1 (b) illustrates this.

3 Thread Migration in Java

As the available computation power of the computers to be used by our library permanently fluctuates, the execution time of a parallel program can be significantly improved if it is possible to migrate its processes at run-time to other hosts with currently more available computation power.

There are three ways to migrate threads in Java: modification of the Java Virtual Machine (VM) [5], bytecode transformations [4, 11], and sourcecode transformations [16, 17]. Modifying the Java VM is not advisable because everybody would have to replace his installation of the original Java VM with one from a third party, just to run a migratable Java program.

Inside PUBWCL, we use *JavaGo RMI* [16, 9] which is an implementation of the sourcecode transformation approach. It extends the Java programming language with three keywords: migrations are performed using the keyword `go` (passing a filename instead of a hostname as parameter creates a backup copy of the execution state). All methods, inside which a migration may take place, have to be declared `migratory`. The depth, up to which the stack will be migrated, can be bounded using the `undock` statement.

The JavaGo compiler *jgoc* translates this extended language into Java sourcecode, using the unfolding technique described in [16]. Migratable programs are executed by dint of the wrapper `javago.Run`. In order to continue the execution of a migratable program, an instance of `javago.BasicServer` has to run on the destination host.

Since the original implementation of JavaGo is not fully compatible with the Java RMI standard, we use our own adapted version JavaGo RMI.

4 The Web Computing Library

People willing to join the *Paderborn University BSP-based Web Computing Library (PUBWCL)* system, have to install a PUBWCL client. With this client,

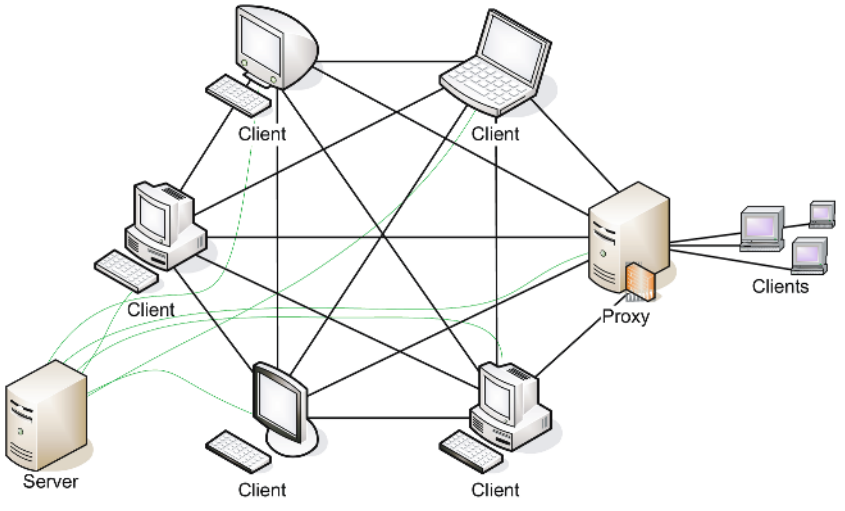


Fig. 2. The architecture of PUBWCL

they can donate their unused computation power and also run their own parallel programs.

Architecture of the system. PUBWCL is a hybrid peer-to-peer system: The execution of parallel programs is carried out on peer-to-peer basis, i. e., among the clients assigned to a task. Administrative tasks (e. g. user management) and the scheduling (i. e. assignment of clients and selection of appropriate migration targets), however, are performed on client-server-basis. Clients in private subnets connect to the PUBWCL system via the *proxy* component. The interaction of the components is illustrated in Fig. 2.

Since the clients may join or leave the PUBWCL system at any time, the login mechanism is lease-based, i. e., a login session expires after some timeout if the client does not regularly report back at the server.

Though a permanent internet connection is required, changes of dynamically assigned IP addresses can be handled. This is accomplished by using *Global Unique Identifiers (GUIDs)* to unambiguously identify the clients: when logging in, each client is assigned a GUID by the server. This GUID can be resolved into the client's current IP address and port.

Executing a parallel program. If users want to execute their own parallel programs, they must be registered PUBWCL users (otherwise, if they only want to donate their unused computation power, it is sufficient to use the guest login). To run a BSP program, it simply has to be copied into a special directory specified in the configuration file. Then one just needs to enter the name of the program and the requested number of parallel processes into a dialog form. Optionally, one may pass command line arguments to the program or choose a certain load balancing algorithm.

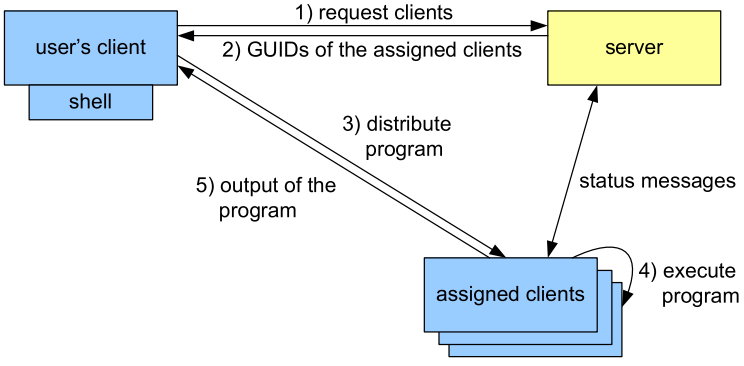


Fig. 3. Executing a BSP program in PUBWCL

The server then assigns the parallel program to a set of clients and sends a list of these clients to the user's client (cf. Fig. 3). From now on, the execution of the parallel program is supervised by the user's client. On each of the assigned clients a PUBWCL runtime environment is started and the user's parallel program is obtained via dynamic code downloading. The output of the parallel program and, possibly, error messages including stack traces are forwarded to the user's client.

All processes of parallel programs are executed in an own PUBWCL runtime environment in a separate process, so that it is possible to cleanly abort single parallel processes (e. g. in case of an error in a user program).

The programming interface. User programs intended to run on PUBWCL have to be BSP programs. Thereto the interface `BSPProgram` resp. `BSPMigratableProgram` must be implemented, which contains the BSP main method. The following BSP library functions can be accessed via the `BSP` resp. `BSPMigratable` interface which is implemented by the PUBWCL runtime environment:

- `sync()` enters the barrier synchronization; the migratable version `syncMig()` additionally creates backup copies of the execution state and performs migrations if suggested by the load balancing strategy.
- With the `send()` method family messages can be sent or broadcasted to an arbitrary subset of the BSP processes. A message can be any serializable Java object.
- Messages sent in the previous superstep can be accessed with the `getMessage()` method family.
- Calling `abort()` terminates all processes of the BSP program.
- Any file in the BSP program folder of the user's client can be read using the `getResourceAsStream()` method.
- Furthermore, there are methods to obtain the number of processes of the BSP program, the own process ID, and so on.
- In migratable programs, there is also a method `mayMigrate()` available which may be called to mark additional points inside a superstep where a migration is safe (i. e. no open files etc.).

Security aspects. The *Java Sandbox* allows to grant code specific permissions depending on its origin. For example, access to (part of) the file system or network can be denied. In order to guarantee a high level of security, we grant user programs only read access to a few Java properties which are needed to write completely platform independent code (e. g. `line.separator` etc.).

5 Performance Evaluation

In order to determine the performance drawback of PUBWCL in comparison to a BSP implementation in C, we have conducted benchmark tests with both PUBWCL and PUB under the same circumstances: We used a cluster of 48 dual Intel Pentium III Xeon 850 MHz machines, that were exclusively reserved for our experiments, to avoid influences by external work load. The computers were interconnected by a switched Fast Ethernet. The used benchmark program was a sequence of 10 equal supersteps. Per superstep, each BSP process did a number of integer operations and sent a number of messages. We performed tests using every possible combination of these parameters:

- 8, 16, 24, 32, 48 BSP processes
- 10, 20, 30 messages per BSP process and superstep
- 10 kB, 50 kB, 100 kB message size
- 0, 10^8, $2 \cdot 10^8$, $3 \cdot 10^8$, ..., 10^9 integer operations per BSP process and superstep

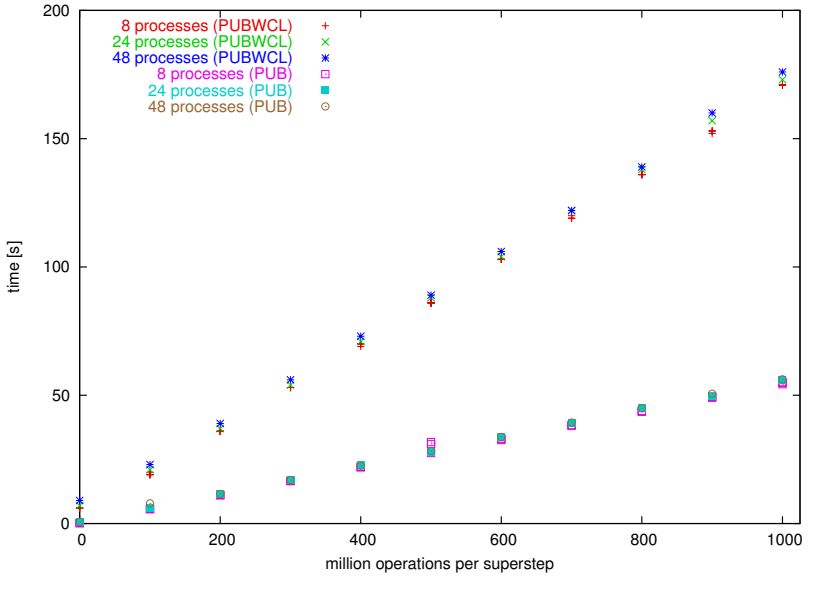


Fig. 4. Results of the benchmark tests

Selected results of the benchmark tests are shown in Fig. 4. As you can see, both BSP libraries scale well, and there is a performance drawback of a factor 3.3. Note that the running time of this benchmark program is dominated by the sequential work. Communication tests with no sequential work showed a performance impact of a factor up to 8.7.

When porting existing BSP programs to PUBWCL, you cannot directly compare the running times due to the overhead of the Java memory management. For example, we ported our C-based solver for the 3-Satisfiability-Problem (3-SAT), which is a simple parallelized version of the sequential algorithm in [1], to PUBWCL. As in the case of the benchmark program, the algorithm is dominated by the sequential work. But in contrast to the benchmark program, it continuously creates, clones, and disposes complex Java objects. This is much slower than allocating, copying, and freeing structures in the corresponding C-program and has led to a performance drawback of a factor 5.4.

6 Conclusion

We have developed a web computing library that allows to execute BSP programs in a peer-to-peer network, utilizing only the computation power left over on the participating PCs. It features migration and restoration of the BSP processes in order to rebalance the load and increase fault tolerance because the available computation power fluctuates and computing nodes may join or leave the peer-to-peer system at any time.

Due to security and portability reasons one has to use a virtual machine like Java's one, so a performance drawback cannot be avoided. The slowdown depends on the type of the BSP algorithm.

We have already implemented and analyzed different load balancing strategies for PUBWCL. A detailed description and comparison of them can be found in [10].

In order to further improve PUBWCL, work is in progress to realize PUBWCL as a pure peer-to-peer system in order to dispose of the bottleneck at the server, and to replace Java RMI in PUBWCL with a more efficient, customized protocol.

To protect PUBWCL against attacks by malicious users, securing the system with a public key infrastructure, encrypted communication tunnels, and access control lists are conceivable.

References

1. Hromkovic, J.: Algorithmics for hard problems. Springer 2003.
2. Sarmenta, L.: An Adaptive, Fault-tolerant Implementation of BSP for Java-based Volunteer Computing Systems. Lecture Notes in Computer Science, vol. 1586, pp. 763–780, 1999.
3. Hill, J., McColl, B., Stefanescu, D., Goudreau, M., Lang, K., Rao, S., Suel, T., Tsantilas, T., Bisseling, R.: The BSP Programming Library. Technical Report, University of Oxford, 1997.

4. Sakamoto, T., Sekiguchi, T., Yonezawa, A.: Bytecode Transformation for Portable Thread Migration in Java. Technical Report, University of Tokyo, 2000.
5. Ma, M., Wang, C., Lau, F.: Delta Execution: A preemptive Java thread migration mechanism. Cluster Computing, 3:83ff, 2000.
6. distributed.net. http://www.distributed.net/
7. Gehweiler, J.: Entwurf und Implementierung einer Laufzeitumgebung für parallele Algorithmen in Java. Studienarbeit, Universität Paderborn, 2003.
8. Great Internet Mersenne Prime Search (GIMPS). http://www.mersenne.org/
9. JavaGo RMI. http://www.joachim-gehweiler.de/software/javago.php
10. Bonorden, O., Gehweiler, J., Meyer auf der Heide, F.: Load Balancing Strategies in a Web Computing Environment. Proceedings of the PPAM 2005, to appear.
11. Truyen, E., Robben, B., Vanhaute, B., Coninx, T., Joosen, W., Verbaeten, P.: Portable Support for Transparent Thread Migration in Java. Technical Report, K.U.Leuven, Heverlee, Belgium, 2000.
12. Bonorden, O., Juurlink, B., von Otte, I., Rieping, I.: The Paderborn University BSP (PUB) Library. Parallel Computing, 29(2):187–207, February 2003.
13. The Paderborn University BSP (PUB) Library. http://wwwcs.unipaderborn.de/∼pub/
14. The Paderborn University BSP-based Web Computing Library (PUBWCL). http://wwwcs.uni-paderborn.de/∼pubwcl/
15. Search for Extraterrestrial Intelligence (SETI@home). http://setiathome.berkeley.edu/
16. Sekiguchi, T., Masuhara, H., Yonezawa, A.: A Simple Extension of Java Language for Controllable Transparent Migration and its Portable Implementation. Technical Report, University of Tokyo, 1999.
17. Fünfrocken, S.: Transparent Migration of Java-base Mobile Agents. Technical Report, Universität Darmstadt, 1998.
18. Valiant, L.: A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.

Matchmaking of Mathematical Web Services

Simone A. Ludwig[1], William Naylor[2], Omer F. Rana[1], and Julian Padget[2]

[1] Department of Computer Science, Cardiff University, UK
[2] Department of Computer Science, University of Bath, UK

Abstract. Service discovery and matchmaking in a distributed environment has been an active research issue since at least the mid 1990s. Previous work on matchmaking has typically presented the problem and service descriptions as free or structured (marked-up) text, so that keyword searches, tree-matching or simple constraint solving are sufficient to identify matches. We discuss the problem of matchmaking for mathematical services, where the semantics play a critical role in determining the applicability or otherwise of a service and for which we use OpenMath descriptions of pre and post-conditions. We describe a matchmaking architecture supporting the use of match plug-ins and describe four kinds of plug-in that we have developed to date.

1 Introduction

There has been an increase in the number of services being made available by many Grid projects – generally through the use of Web Services technologies (examples include BioInformatics services, Mathematical services, and recently data mining services). Identifying suitability of such services in the context of a particular application remains a challenging task. Humans typically use Google, but they can filter out the irrelevant and spot the useful. Although UDDI (the Web Services registry) with keyword searching essentially offers something similar, it is a long way from being very helpful. Consequently, there has been much research on intelligent brokerage, such as Infosleuth [3] and LARKS [8]. This approach is generally referred to as "Matchmaking", and involves potential producers and consumers of information sending messages describing their information capabilities and needs. These descriptions are unified by the *matchmaker* to identify potential matches. Existing literature in this area focuses on architectures for brokerage, which are as such domain-independent, rather than concrete or domain-specific techniques for identifying matches between a *task* or problem description and a *capability* or service description. Approaches to matching in the literature fall into two broad categories: (1) *syntactic matching*, such as textual comparison or the presence of keywords in free text. (2) *semantic matching*, which typically seems to mean finding elements in structured (marked-up) data and perhaps additionally the satisfaction of constraints specifying ranges of values or relationships between one element and another. In the particular domain of mathematical services the actual mathematical semantics are critical to determining the suitability (or otherwise) of the capability for the task. The requirements are neatly captured in [2] by the following condition:

$$T_{in} \geq C_{in} \wedge T_{out} \leq C_{out} \wedge T_{pre} \Rightarrow C_{pre} \wedge C_{post} \Rightarrow T_{post}$$

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 809–816, 2006.

where T refers to the task, C to the capability, in are inputs, out are outputs, pre are pre-conditions and $post$ are post-conditions. What the condition expresses is that the signature constrains the task inputs to be at least as great as the capability inputs (i.e. enough information), that the inverse relationship holds for the outputs and there is a pairwise implication between the respective pre- and post-conditions. This however leaves unaddressed the matter of establishing the validity of that implication. Agents play key rôles within the matchmaking framework: (1) Provider agents: offer (a) specialist service. (2) Requester agents: consume services offered by provider agents in the system and create tasks. (3) Matchmaker agents: mediate between both for some mutually beneficial cooperation. Each provider must register their capabilities with the matchmaker. Every request received will be matched with the actual set of advertisements. If matches are found a ranked set of appropriate provider agents is returned to the requester agent. In the MONET [4] and GENSS [9] projects the objective is mathematical problem solving through service discovery and composition by means of intelligent brokerage. *Mathematical* capability descriptions turn out to be both a blessing and a curse: precise service description are possible thanks to the use of the OpenMath [5] mathematical semantic mark-up, but service matching can rapidly turn into intractable (symbolic) mathematical calculations unless care is taken.

2 Description of Mathematical Services

2.1 OpenMath

In order to allow mathematical objects to be exchanged between computer programs, stored in databases, or published on the Internet, an emerging standard called OpenMath has been introduced. OpenMath is a mark up language for representing the semantics (as opposed to the presentation) of mathematical objects in an unambiguous way. It may be expressed using an XML syntax. OpenMath expressions are composed of a small number of primitives. The definition of these may be found in [10], for instance: `OMA` (OpenMath Application), `OMI` (OpenMath Integer), `OMS` (OpenMath Symbol) and `OMV` (OpenMath Variable). Symbols are used to represent objects defined in the Content Dictionaries (CDs), applications specify that the first child is a function or operator to be applied to the following children whilst the variables and integers speak for themselves. As an example, the expression $x + 1$ might look like:[1]

```
<om:OMA>
  <om:OMS cd="arith1" name="plus"/>
  <om:OMV name="x"/>
  <om:OMI> 1 </om:OMI>
</om:OMA>
```

where the symbol `plus` is defined in the CD `arith1`. CDs are therfore definition repositories in the form of files defining a collection of related symbols and their meanings, together with various *Commented Mathematical Properties* (for human consumption) and *Formal Mathematical Properties* (for machine consumption). The symbols

[1] Throughout the paper, the prefix `om` is used to denote the namespace: `http://www.openmath.org/OpenMath`

may be uniquely referenced by the CD name and symbol name via the attributes `cd` and `name` respectively, as in the above example. Another way of thinking of a CD is as a small, specialised ontology.

2.2 Matchmaking Requirements

To achieve matchmaking we want sufficient input information in the task to satisfy the capability, while the outputs of the matched service should contain at least as much information as the task is seeking. This is achieved when the capability pre-conditions are a subset of the task pre-conditions, while the task post-conditions are a subset of the capability post-conditions. These constraints reflect work in component-based software engineering and are, in fact, derived from [11]. We find that in certain situations information required by a capability may be inferred from the task, or from widely excepted conventions. Examples include assuming the lower limit of integration is zero for a numerical integration, or the dependant variable of a symbolic integration of a univariate function. Conversely, a numerical integration routine might only work from 0 to the upper limit, while the lower limit of the problem is non-zero. A capability that matches the task can be synthesized from the subtraction of two invocations of the capability with a fixed lower limit of 0 in order to achieve the required task. Clearly the nature of the second solution is quite different from the first, but both serve to illustrate the complexity of this domain. Furthermore, we believe that given the nature of the problem, it is only very rarely that a task description will match exactly a capability description and so a range of reasoning mechanisms must be applied to identify candidate matches. This results in two requirement: (1) A plug-in architecture supporting the incorporation of an arbitrary number of matchers. (2) A ranking mechanism is required that takes into account technical (as discussed above in terms of signatures and pre- and post-condition) and quantitative and qualitative aspects (if there are multiple matching service found).

2.3 Matchmaking Architecture

Our matchmaking architecture is shown in Figure 1 and comprises the following:

1. Client interface: employed by the user to specify their service request.
2. Matchmaker: which contains a reasoning engine and the matching module.
3. Matching Algorithms: where the logic of the matching is defined.
4. Mathematical ontologies: such as OpenMath CDs, etc.
5. Registry service: where the mathematical service descriptions are stored.
6. Mathematical Web Services: available on third party sites, accessible over the Web.

Both a graphical interface and an Application Programming Interface are supported. The interactions of a search request are as follows: (i) the user contacts the matchmaker, then (ii) the matchmaker loads the matching algorithms specified by the user; in the case of an ontological match, further steps are necessary (iii) the matchmaker contacts the reasoner which in turn loads the corresponding ontology (iv) having additional match values results in the registry being queried, to see whether it contains services which match the request, and finally (v) service details are returned to the user via the matchmaker. The parameters stored in the registry are service name, URL, taxonomy, input, output, pre and post-conditions.

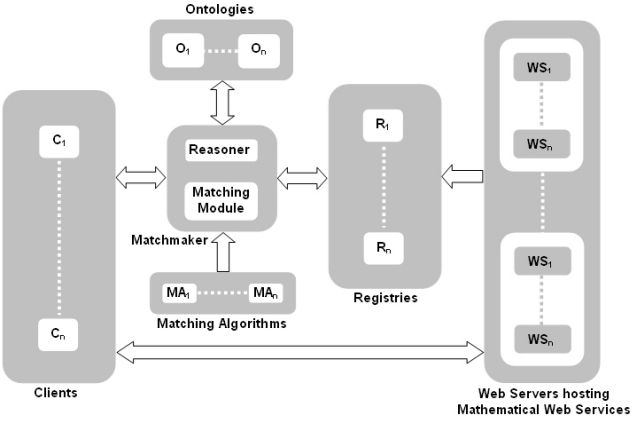


Fig. 1. Matchmaking Architecture

The Client GUI[2] allows the user to specify the service request via entry fields for pre and post-conditions. The matchmaker returns the matches in the table at the bottom of the GUI listing the matched services ranked by similarity. Subsequently the user can invoke the service by clicking on the URL.

Currently four complementary matching algorithms have been implemented within the matchmaker: structural match, syntax and ontological match, algebraic equivalence match and value substitution match. The structural match only compares the OpenMath symbol element structures (e.g. `OMA`, `OMS`, `OMV` etc.). The syntax and ontological match algorithm goes a step further and compares the OpenMath symbol elements (`OMS` elements) by comparing the values of the `cd` and `name` attributes of the `OMS` elements. If a syntax match is found, which means that the `cd` and `name` values are identical, then no ontology match is necessary. If an ontology match is required, then inclusion relationships are considered to attempt to form a match. The algebraic equivalence match and value substitution match use mathematical reasoning. In the ontological match OpenMath elements are compared with an ontology [7] representing the OpenMath elements. The matchmaking mechanism allows a more effective matchmaking process by using mathematical ontologies such as the one for sets shown in Figure 2. OWL-JessKB [1] was used to implement the ontological match. It involves reading Ontology Web Language (OWL) files, interpreting the information as per the OWL and RDF languages, and allowing the user to query on that information. To give an example the user query contains the OpenMath element `<om:OMS cd='setname1' name='Z'/>` (the integers) and the service description contains the OpenMath element `<om:OMS cd='setname1' name='P'>` (the prime numbers). One way of determining the distance between these (set valued) concepts is to count the number of arcs between their representatives in the ontology (which in this case represents an inclusion graph) Figure 2. The query finds the nodes labelled `Z` and `P`, and determines the similarity

[2] `http://agentcities.cs.bath.ac.uk:8080/genss_axis/GENSSMatchmaker.jsp`

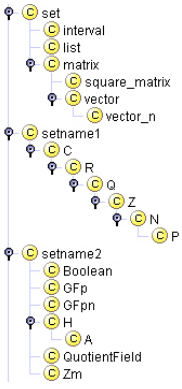


Fig. 2. Set Ontology Fragment

```
<om:OMOBJ><om:OMA>
  <om:OMS cd="arith1" name="minus"/>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="x"/>
    <om:OMI>2</om:OMI>
  </om:OMA>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="y"/>
    <om:OMI>2</om:OMI>
  </om:OMA></om:OMA>
</om:OMOBJ>
```

```
<om:OMOBJ><om:OMA>
  <om:OMS cd="arith1" name="times"/>
  <om:OMA>
    <om:OMS cd="arith1" name="plus"/>
    <om:OMV name="x"/>
    <om:OMV name="y"/>
  </om:OMA>
  <om:OMA>
    <om:OMS cd="arith1" name="minus"/>
    <om:OMV name="x"/>
    <om:OMV name="y"/>
  </om:OMA></om:OMA>
</om:OMOBJ>
```

Fig. 3. $x^2 - y^2$ (left) and $(x + y)(x - y)$ (right) in OpenMath

value depending on the distance between the two entities (inclusive, on one side) which in this case is $SV = \frac{1}{n} = 0.5$, where n is the number of arcs separating the two nodes, which also gives a good measure of the separation of the concepts defined by the corresponding ontology.

For both the ontological and structural match, it is necessary that the pre- and post-conditions are in some standard form. For instance, consider the algebraic expression $x^2 - y^2$, this could be represented in OpenMath as on the left of Figure 3, however, $x^2 - y^2 = (x+y)(x-y)$, leading to the ontologically and structurally different markup on the right of Figure 3. In order to address the above observation, we must look deeper into the mathematical structure of the expressions which make up the post-conditions. Most of the conditions examined may be expressed in the form: $\mathrm{Q}(\mathrm{L}(\mathrm{R}))$ where:

- Q is a quantifier block, e.g. $\forall x \exists y$ s.t. $\cdots$
- L is a block of logical connectives, e.g. $\wedge, \vee, \Rightarrow, \cdots$
- R is a block of relations, e.g. $=, \leq, \geq, \neq, \cdots$

Processing the Quantifier Block: In most cases, the quantifier block will just be a range restriction. Sometimes it may be possible to use *quantifier elimination* to replace the

quantifier block by an augmented logical block and many computer algebra systems, e.g. *RedLog* in Reduce, provide quantifier elimination facilities.

Processing the Logical Block: Once the quantifier elimination has been performed on the query descriptions and the service descriptions, the resulting logical blocks must be converted into normal forms. The query logical block will be transformed into disjunctive normal form: $\bigvee_{i,\tilde{i}}(\bigwedge_j QT_{i_j} \wedge \bigwedge_{\tilde{j}} \overline{QT_{\tilde{i}_{\tilde{j}}}})$. The service description block will be transformed into conjunctive normal form: $\bigwedge_{i,\tilde{i}}(\bigvee_j ST_{i_j} \vee \bigvee_{\tilde{j}} \overline{ST_{\tilde{i}_{\tilde{j}}}})$.
Hence we may perform a match by taking any one of the conjunctions $\bigwedge_j QT_{i_j}$ (or $\bigwedge_{\tilde{j}} \overline{QT_{\tilde{i}_{\tilde{j}}}}$) and looking for the case that each one of the QT_{i_j} (or $\overline{QT_{\tilde{i}_{\tilde{j}}}}$ appears in separate ST_{i_j} (or $\overline{ST_{\tilde{i}_{\tilde{j}}}}$). Non matches may be detected if one of the QT_{i_j} matches one of the $\overline{ST_{\tilde{i}_{\tilde{j}}}}$ or if one of the $\overline{QT_{\tilde{i}_{\tilde{j}}}}$ matches one of the ST_{i_j}.

Processing the Relations Block: We now have a disjunction of terms which we are matching against a set of conjunction of terms. It is useful to note that a term is of the general form: $T_L \succ T_R$ where $\succ$ is some relation i.e. a predicate on two arguments. If T_L and T_R are real valued, we have two terms we wish to compare $Q_L \succ Q_R$ and $S_L \succ S_R$, we first isolate an output variable r, this will give us terms $r \succ Q$ and $r \succ S$. There are two approaches which we now try in order to prove equivalence of $r \succ Q$ and $r \succ S$:

- Algebraic equivalence: we attempt to show that the expression $(Q - S = 0)$ using algebraic means. There are many cases were this approach will work, however it has been proved [6] that in general this problem is undecidable. Another approach involves substitution of r determined from the condition $r \succ S$ into $r \succ Q$, and subsequently proving their equivalence.
- Value substitution: we attempt to show that $(Q - S = 0)$ by substituting random values for each variable in the expression, then evaluating and checking to see if the valuation we get is zero. This is evidence that $(Q - S = 0)$, but is not conclusive, since we may have been unlucky in the case that the random values coincide with a zero of the expression.

3 Case Study

For the case study we consider the four matching modes. The Factorisor service we shall look at is a service which finds all prime factors of an Integer. The Factorisor has the following post-condition:

```
<om:OMOBJ>
  <om:OMA>
    <om:OMS cd ='relation1' name ='eq'/>
    <om:OMV name ='n'/>
    <om:OMA>
      <om:OMS cd ='fns2' name ='apply_to_list'/>
      <om:OMS cd ='arith1' name ='times'/>
      <om:OMV name ='lst_fcts'/>
    </om:OMA>
  </om:OMA>
</om:OMOBJ>
```

where `n` is the number we wish to factorise and `lst_fcts` is the output list of factors.

As the structural and ontological modes compare the OpenMath structure of queries and services, and the algebraic equivalence and substitution modes perform mathematical reasoning, the case study needs to reflect this by providing two different types of queries.

For the structural and ontological mode let us assume that the user specifies the following query:

```
<om:OMOBJ>
  <om:OMA>
    <om:OMS cd ='fns2' name ='apply_to_list'/>
    <om:OMS cd ='arith1' name ='plus'/>
    <om:OMV name ='lst_fcts'/>
  </om:OMA>
</om:OMOBJ>
```

For the structural match, the query would be split into the following `OM` collection: `OMA`, `OMS`, `OMS`, `OMV` and `/OMA` in order to search the database with this given pattern. The match score of the post-condition results in a value of 0.27778 using the equations described earlier.

The syntax and ontology match works slightly different as it also considers the *values* of the `OM` symbols. In our example we have three `OM` symbol structures. There are two instances of `OMS` and one of `OMV`. First the query and the service description are compared syntactically. If there is no match, then the ontology match is called for the `OMS` structure. The value of the content dictionary (CD) and the value of the name are compared using the ontology of that particular CD. In this case the result is a match score of 0.22222. If the `OM` structure of the service description is exactly the same as the query then the structural match score is the same as for the syntax and ontology match. The post-condition for the Factorisor service represents:

$$\mathtt{n} = \prod_{i=1}^{l} \mathtt{lst\_fcts}_i \qquad \text{where } \mathtt{l} = |\mathtt{lst\_fcts}| \tag{1}$$

Considering the algebraic equivalence and the value substitution, a user asking for a service with post-condition:

$$\forall i | 1 \leq i \leq |\mathtt{lst\_fcts}| \Rightarrow n \bmod \mathtt{lst\_fcts}_i = 0 \tag{2}$$

should get a match to this Factorisor service.
To carry out the algebraic equivalence match we use a proof checker to show that:

1. equation (1) $\Rightarrow$ equation (2): This is clear since the value of n may be substituted into equation (1) and the resulting equality will be true for each value in `lst_fcts`.
2. equation (2) $\Rightarrow$ equation (1): A slightly stronger version of equation (2) which says that there are no other numbers which divide n.

To compute the value substitution match we must gather evidence for the truth of equations 1 and 2 by considering a number of random examples, we proceed as follows. We first need to decide on the length of the list for our random example. A good basis would be to take $|\mathtt{lst\_fcts}| = \lceil \log_2(\mathtt{n}) \rceil$, this represents a bound on the number of factors in the input number. We then collect that number of random numbers, each of size bounded by $\sqrt{\mathtt{n}}$. Then we calculate their product, from equation (1), this gives a new value for `n`. We may now check equation (2). We see that it is true for every value

in lst_fcts. If we try this for a few random selections, we obtain evidence for the equivalence of equations (1) and (2).

4 Conclusion and Further Work

We have presented an approach to matchmaking in the context of mathematical semantics. The additional semantic information greatly assists in identifying suitable services in some cases, but also significantly complicates matters in others, due to the inherent richness of mathematical semantics. Consequently, we have put forward an extensible matchmaker architecture supporting plug-in matchers that may employ a variety of reasoning techniques, including theorem provers, as well as information retrieval from textual documentation of mathematical routines. Although our set of test cases is as yet quite small, the results are promising and we foresee the outputs of the project being of widespread utility in both the e-Science and Grid communities, as well as more generally advancing semantic matchmaking technology. Although the focus here is on matchmaking mathematical capabilities, the descriptive power, deriving from quantification and logic combined with the extensibility of OpenMath creates the possibility for an extremely powerful general purpose mechanism for the description of both tasks and capabilities.

References

1. OWLJessKB. http://edge.cs.drexel.edu/assemblies/software/owljesskb/.
2. M. Gomez and E. Plaza. Extended matchmaking to maximize capability reuse. In Nicholas R. Jennings et al., editor, *Proceedings of The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 144–151. ACM Press, 2004.
3. W. Bohrer M. Nodine and A.H. Ngu. Semantic brokering over dynamic heterogenous data sources in infosleuth. In *Proceedings of the 15th International Conference on Data Engineering, pp. 358-365*, 1999.
4. MONET Consortium. MONET Home Page, www. Available from http://monet.nag.co.uk.
5. OpenMath Society. OpenMath website. http://www.openmath.org, February www.
6. D. Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Computational Logic*, 33:514–520, 1968.
7. J.F. Sowa. Ontology, metadata, and semiotics, conceptual structures: Logical, linguistic, and computational issues. *Lecture Notes in AI #1867, Springer-Verlag, Berlin, pp. 55-81*, 2000.
8. K. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Journal of Autonomous Agents and Multi Agent Systems*, 5(2):173–203, June 2002.
9. The GENSS Project. GENSS Home Page, www. Available from http://genss.cs.bath.ac.uk.
10. The OpenMath Society. The OpenMath Standard, October 2002. Available from http://www.openmath.org/standard/om11/omstd11.xml.
11. A.M. Zaremski and J.M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, October 1997.

Porting CFD Codes Towards Grids: A Case Study

Dana Petcu[1,2], Daniel Vizman[3], and Marcin Paprzycki[4]

[1] Computer Science Department, Western University
[2] Institute e-Austria, Timişoara
[3] Physics Department, Western University of Timişoara, Romania
[4] Computer Sciecne Institute, SWPS, Warsaw, Poland
petcu@info.uvt.ro, vizman@physics.uvt.ro, marcin.paprzycki@swps.edu.pl

Abstract. In this paper we discuss an application of a modified version of a graph partitioning-based heuristic load-balancing algorithm known as the *Largest Task First with Minimum Finish Time and Available Communication Costs*, which is a part of the *EVAH* package. The proposed modification takes into account the dynamic nature and heterogeneity of grid environments. The new algorithm is applied to facilitate load balance of a known CFD code used to model crystal growth.

1 Introduction

Computational fluid dynamics codes are computationally demanding, both in terms of memory usage and also in the total number of arithmetical operations. Since the most natural methods of improving accuracy of a solution are (1) refining a mesh or/and (2) shortening the time step, either of these approaches result in substantial further increase of both computational cost and total memory usage. Therefore, a tendency can be observed, to use whatever computational resources are available to the user and, particularly among researchers working in the CFD area, there exist an almost insatiable demand for more powerful computers with ever increasing size of available memory. One of possible ways to satisfy this demand is to divide up the program to run on multiple processors. In the last twenty years several codes have been introduced to facilitate parallel computational fluid dynamics.

Parallel CFD codes have been typically developed assuming a homogeneous set of processors and a fast network. Recent ascent of grid technologies requires re-evaluation of these assumptions as the very idea of computational grids is based on combining heterogeneous processors and using substantially slower networks connections (typically the Internet or a corporate LAN). This makes the environment much different than parallel computers or clusters of workstations connected using a fast switch. Furthermore, the migration process of CFD codes designed for parallel computing architectures towards grids must take into account not only the heterogeneity of the new environment but also the dynamic evolution of the pool of available computational resources. At the same time we

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 817–824, 2006.

have to acknowledge that to be able to fully rewrite the existing codes is usually not a viable option (e.g. because of the cost involved in such an endeavor).

In this context let us note that large body of research has been already devoted to dynamic load balancing in heterogeneous environments, and more recently in grid environments (e.g. in [2] dynamic load-balancing in a grid environment is used for a geophysical application).

Approaches to load-balancing in distributed systems can be classified into three categories: (1) graph-theoretic, (2) mathematical programming based, and (3) heuristic. Graph-theoretic algorithms consider graphs representing the inter-task dependencies and apply graph partitioning methodologies to obtain approximately equal partitions of the graph such that the inter-node communication is minimized. A CFD simulation using such an approach is described in [5]. The mathematical programming method, views the load-balancing as an optimization problem and solves it using techniques originating from that domain. Finally, heuristic methods provide fast solutions (even though usually sub-optimal ones) when the time to obtain the exact optimal solution is prohibitive. For example in [1] a genetic algorithm is used as an iterative heuristic method to obtain near optimal solutions to a combinatorial optimization problem that is then applied to job scheduling on a grid.

When approached from a different perspective, we can divide load management approaches into (1) system level, and (2) user-level. A system-level centralized management strategy, which works over all applications uses schedulers to manage loads in the system. It is typically based on rules associated with job types or load classes. An example of the the user-level individual management of loads in a parallel computing environment is the *Dynamic Load Balancing* (*DLB* [8]) tool that lets the system balance loads without going through centralized load management and, furthermore, provides application level load balancing for individual parallel jobs. Here, a CFD test case was used as an example. System load measurement of the *DLB* is modified using average load history provided by computing systems rather than by tracking processing of tasks.

The *EVAH* package [3] was developed to predict the performance scalability of overset grid applications executing on large numbers of processors. In particular, it consists of a set of allocation heuristics that consider the constraints inherent in multi-block CFD problems.

In this paper we analyze and modify a graph partitioning-based heuristic algorithm available within the *EVAH* package, the *Largest Task First with Minimum Finish Time and Available Communication Costs* (*LTF_MFT_ACC*). The main drawback of this algorithm is that it assumes that grid-available resources are homogeneous. For instance, to show its efficiency tests performed on an Origin2000 system and were reported in [3]. We propose a modification of the *LTF_MFT_ACC* algorithm that can be applied in the case of a heterogeneous computing environment (such as a typical grid is supposed to be). Inspired by the *DLB* tool, our algorithm takes into account (1) the history of the computation time on different nodes, (2) the communication requirements, and (3) the current network speeds. To study the robustness of the proposed improvements,

the modified *LTF_MFT_ACC* algorithm is used to port an existing CFD code into a heterogeneous computing environment.

The paper is organized as follows. Section 2 describes the CFD code and its parallel implementation. Section 3 presents the modified *LTF_MFT_ACC* algorithm. Section 4 discusses the results of our initial test.

2 Crystal Growth Simulation

Materials processing systems are often characterized by the presence of a number of distinct materials and phases with significantly different thermo-physical and transport properties. Understanding of the complex transport phenomena in these systems is of vital importance for the design and fabrication of various desired products as well as optimization and control of the manufacturing process. It is well known that numerical simulation prove to be an effective tool for the understanding of the transport mechanisms. However, in computational practice, three-dimensional simulations are necessary to yield a reliable description of the flow behavior. Usual computational methods applied to these problems include finite difference, finite volume, finite element, and spectral methods.

In particular, let us consider the Czochralski process [11] of bulk crystal growth that features a rod holding an oriented seed crystal which is lowered through the top surface of the molten liquid contained in a crucible. With thermal control to maintain the upper surface of the fluid at the melt temperature, growth begins on the seed and when the crystal reaches a specified diameter, the rod is slowly withdraw to continue growth (Figure 1.a). The flow in the melt, from which the crystal is pulled, is transient and, depending on the size of the crucible, mostly turbulent.

The silicon melt flow into a rotating crucible is governed by the three-dimensional partial differential equations describing mass, momentum, and heat transport. Solution methods that employ finite volume (see e.g [4]) require generation of the solution grid that conforms to the geometry of the flow region (a grid of small volume elements for which the average values of flow quantities are stored). An important issue for the quality of the numerical simulations is the choice of the (discretizing) grid. Here, both the numerical resolution and the internal structure of the grid are very important. The second item can be seen, is the refining of the grid towards the walls of the melt container, which is necessary to properly resolve the boundary layers of the flow.

As far as the solution was concerned, a matched multiblock method was used in our simulations (here, the grid lines match each other at the block conjunction). Here, a multiblock structured grid system [13] uses advanced linear solvers, for the inner iteration, and a multigrid technique for the outer iterations. Furthermore, the computational domain is divided into blocks consisting of control volumes (from hundreds to millions; see Figure 1.b).

More specifically, the finite volume code *STHAMAS 3D* (developed partially by the second author at the Institute of Materials Science in Erlangen) allows three-dimensional time-dependent simulations on a block-structured numerical

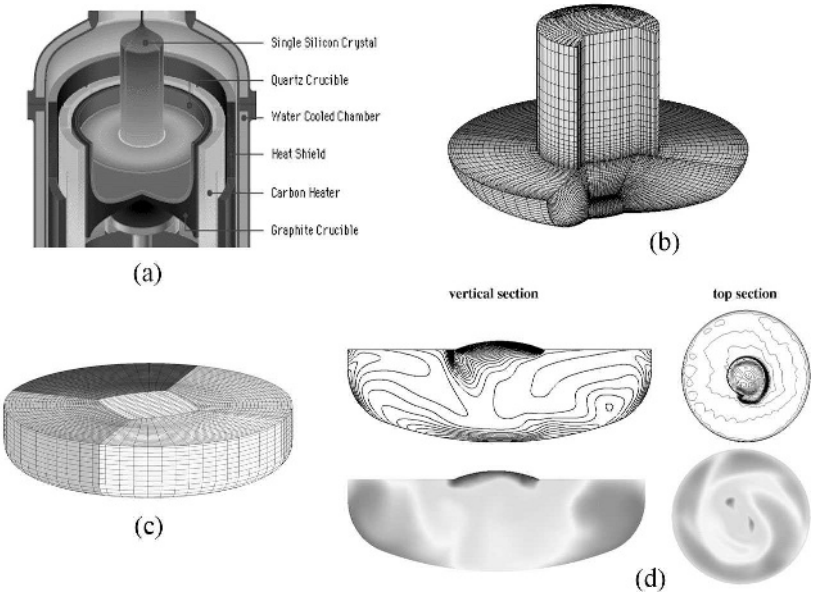


Fig. 1. Crystal growth: (a) device; (b) control volumes; (c) blocks of control volumes; (d) code outputs – isoterms and animation frames

grid. *SIP* (*Stone's strongly implicit procedure* [12]) is used to solve the system of linear equation resulting from the discretization of PDEs for three-dimensional problems (it is applicable to seven-diagonal coefficient matrices that are obtained when central-difference approximation is used to discretize the problem). *SIMPLE* algorithm (*Semi-Implicit Method for Pressure-Linked Equations*, [9]) is used for the pressure correction and the implicit Euler method is applied for time integration. The *SIP* and the *SIMPLE* were studied and compared (in [6]) with other solvers and shown to be very robust. A simple example of the graphical output of the code is presented in Figure 1.d.

Time-dependence and three-dimensionality coupled with extensive parameter variations require a very large amount of computational resources and result in very long solution times. The most time-consuming part of the sequential code *STHAMAS 3D* is the numerical solution obtained, using the SIP, on different blocks of CVs. In order to decrease the response time of *STHAMAS 3D*, a parallel version was recently developed by the first two authors and compared with other similar CFD codes (see [10]). It is based on a parallel version of the SIP solver, where simultaneous computations are performed on different blocks mapped to different processors (different colors in Figure 1.c). After each inner iteration, information exchanges are performed at the level of block surfaces (using the MPI library). Thus far, the new parallel *STHAMAS 3D* was tested only utilizing homogeneous computing environments, in particular, a clusters of workstations and a parallel computer.

For completeness it should be noted that a different parallel version of a crystal growth simulation has been reported in [7]. It utilizes a parallel version of the *SSOR* preconditioner and the *BiCGSTAB* iterative solver.

3 Load Balancing Strategy

In the *Largest Task First with Minimum Finish Time and Available Communication Costs* (*LTF_MFT_ACC*) algorithm from the *EVAH* package [3] a task is associated with a block in the CFD grid system. The size of a task is defined as the computation time for the corresponding block. According the *Largest Task First* (*LTF*) policy, the algorithms first sorts the tasks in a descending order by their size. Then it systematically allocates tasks to processors respecting the rule of *Minimum Finish Time* (*LTF_MFT*). The overhead involved in this process, due to data exchanges between neighboring blocks, is also taken into account. The *LTF_MFT_ACC* utilizes approximations of communication costs, which are estimated on the basis of the inter-block data volume exchange and the inter-processor communication rate.

Input:
Current distribution of the N blocks on P processors: $p(i) \in \{1, \ldots, P\},\ i = 1, \ldots, N$
Output:
New distribution of the N blocks on P processors: $p'(i) \in \{1, \ldots, P\},\ i = 1, \ldots, N$

Preliminaries, using the current distribution:
Record the computation time for each block: $T_i,\ i = 1, \ldots, N$
Record the quantity of data to be send/receive between blocks: $V_{j,k},\ j, k = 1, \ldots, N$
Estimate communication time between each pair of processors depending on the quantity of transmitted data: $Send(p, q, dim),\ Recv(p, q, dim),\ p, q = 1, \ldots, P,\ p \neq q$
Record the time spent to perform a standard test: $c_p,\ p = 1, \ldots, P$
Compute the relative speeds of computers: $w_p \leftarrow c_p / \min_{p=1,\ldots,P} c_p,\ p = 1, \ldots, P$
Normalize computation time for each block: $t_i \leftarrow T_i / w_{p(i)}, i = 1, \ldots, N$

To do:
Sort $t_i, i = 1, N$ in descending order
Set costs $C_p = 0, p = 1, \ldots, P$
For each $t_i, i = 1, \ldots, N$ do
  Find the processor q with minimum load: $?q,\ C_q = \min_{p=1,\ldots,P} C_p$
  Associate block i with processor q, $p'(i) \leftarrow q$
  Modify the costs: $C_q \leftarrow C_q + w_q t_i$
  For each processor $o \neq q$ having assigned a task j sending a message to task i,
    $C_o \leftarrow C_o + Send(o, q, V(j, i))$
    $C_q \leftarrow C_q + Recv(q, o, V(j, i))$
  For each processor $o \neq q$ having assigned a task j receiving a message from task i,
    $C_o \leftarrow C_o + Recv(o, q, V(i, j))$
    $C_q \leftarrow C_q + Send(q, o, V(i, j))$

Fig. 2. Modified *LTF_MFT_ACC* algorithm

In the *LTF_MFT_ACC* algorithm described in [3] for the homogeneous case, the estimated computational time for block i, t_i does not vary with the processor power. To take into account the variation of the computational power of the heterogeneous resources, we have modified the *LTF_MFT_ACC* (Figure 2) as follows. When the load balancing procedure is activated before a specific inner iteration of the simulation, several counters are started and they stop only at the end of the inner iteration. Those counters are measuring:

- the computer power, conceptualized as the time of performing a single cycle involving floating point operations; this information is further used to rank the resources;
- the computation time spent working on each block at an inner iteration; this information and the relative computer power are used to assume what will be the time spent working on a given block by other processor(s);
- the required volume of data to be exchanged between neighboring blocks (number of elementary data items);
- samples of communication times between each pair of processors collected for several volumes data (further time values are estimated by a linear interpolation).

The *DLB* tool [8] uses as inputs for the load balance strategy also the timing of computation for parallel blocks and the size of the interface of each block with its neighbors. Those times are not referring to a specific inner iteration of the CFD simulation, but to an average load history provided by computing systems that are part of the grid that is used to solve the problem.

4 Tests

The initial *LTF_MFT_ACC* algorithm was applied in [3] to a selected CFD problem, a Navier-Stokes simulation of vortex dynamics in the complex wake of a region around hovering rotors. The overset grid system consisted of 857 blocks and approximately 69 million grid points. The experiments running on the 512-processor SGI Origin2000 distributed-shared-memory system showed that the *EVAH* algorithm performs better than other heuristic algorithms.

The CFD test case from the [8] used a three-dimensional grid for a heat transfer problem. The grid consisted of 27-block partitions with 40x40x40 grid points on each block (1.7 millions of grid points). For a range of relative speeds of computers from $1 \div 1.55$ a 21% improvement in the elapsed time was registered.

In our tests we have considered a three-dimensional grid applied to the crystal growth simulation using STHAMAS 3D with 38-block-partitions with a variable number of grid points: the largest one had 25x25x40 points, while the smallest one had 6x25x13 points (total of 0.3 millions of grid points in the simulation).

We considered two computing environments:

- a homogeneous one a Linux cluster of 8 dual PIV Xeon 2GHz Processors with 2Gb RAM and a Myrinet2000 interconnection (http://www.oscer.edu);

– a heterogeneous one – a Linux network of 16 PCs with variable computational power, from an Intel Celeron running at 0.6GHz, with 128Mb RAM to an Intel PIV running at 3GHz and with 1Gb RAM; these machines were connected through an Ethernet 10 Mbs interconnection (http://www.risc.unilinz.ac.at).

Thus, the interval of the relative speeds of computers used in the second case reached $1 \div 2.9$.

Initially blocks of the discretization were distributed uniformly between the processors (e.g. in the case of two processors, first 19 blocks were send to the first processor, and the last 19 blocks were send to the second processor).

Due to the different number of grid points in individual blocks, the initial *LTF_MFT_ACC* algorithm running in the homogeneous environment recommended a new distribution of the nodes. Also the modified *LTF_MFT_ACC* algorithm made such a recommendation. For example, a reduction of 6% of the computation time required by an inner iteration was registered by applying both algorithms (the original one and the modified one) in the case of using 4 processors (Figure 3.a).

In the case of the heterogeneous environment, the *LTF_MFT_ACC* algorithm performs better: we observe a reduction ranging from 14% to 20% of the time spent by the CFD code in the inner iteration. A further reduction of the time was obtained when applying the modified *LTF_MFT_ACC* algorithm — varying from 20% to 31% (Figure 3.b). Comparing the time results with the ones for the initial distribution, a total reduction time obtained in our experiments varies from 33% to 45%.

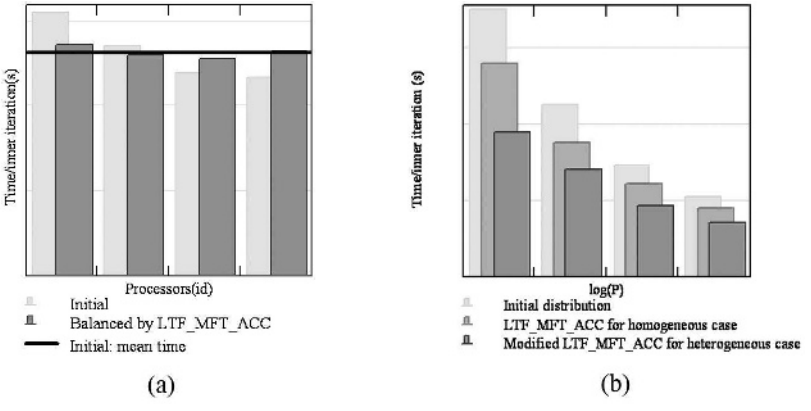


Fig. 3. Load balancing results : (a) in the case of 4 processors of the homogeneous environment, the *LTF_MFT_ACC* algorithm reduces the differences between the computation time spent by each processor in the inner iteration; (b) in the case of 2, 4, 8 and 16 processors of the heterogeneous environment, the *LTF_MFT_ACC* reduces the time per inner iteration, but a further significant reduction is possible using the modified *LTF_MFT_ACC* algorithm.

5 Further Improvements

The proposed load balancing technique shows to be useful in the considered case, a version of a CFD code running within heterogeneous or grid environments. Tests must be further performed to compare several dynamic load balancing techniques with the proposed one, not only in what concerns the influence of the computer power variations as in this paper, but also of the network speed variation. A particular grid testbed running MPICH-G2 applications will be used in the near future to perform such tests.

References

1. J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd, Agent-based grid load balancing using performance-driven task scheduling. In Procs. of IPDPS03, IEEE Computer Press (2003).
2. R. David, S. Genaud, A. Giersch, B. Schwarz, and E. Violard, Source code transformations strategies to load-balance grid applications. In Procs. GRID 2002, M. Parashar (ed.), *LNCS* **2536**, Springer (2002), 82–87.
3. M.J. Djomehri, R. Biswas, N. Lopez-Benitez, Load balancing strategies for multi-block overset grid applications, NAS-03-007, Available at www.nas.nasa.gov/News/Techreports/2003/PDF/nas-03-007.pdf.
4. J.H.Ferziger, M.Perić, *Computational Methods for Fluid Dynamics*, Springer (1996).
5. H. Gao, A. Schmidt, A. Gupta, P. Luksch, Load balancing for spatial-grid based parallel numerical simulations on clusters of SMPs, In *Procs. Euro PDP03*, IEEE Computer Press (2003), 75–82.
6. O. Iliev, M. Scäfer, A numerical study of the efficiency of SIMPLE-type algorithms in computing incompressible flows on streched grids. In Procs. LSSC99, M. Griebel, S. Margenov, P. Yalamov (eds.), *Notes on Numerical Fluid Mechanics* **73**, Vieweg (2000), 207–214
7. D. Lukanin, V. Kalaev, A. Zhmakin, Parallel simulation of Czochralski crystal growth. In Procs. PPAM 2003, R. Wyrzykowski et al, *LNCS* **3019** (2004), 469–474
8. R.U. Payli, E. Yilmaz, A. Ecer, H.U. Akay, and S. Chien, DLB A dynamic load balancing tool for grid computing. In *Procs. Parallel CFD04*, G. Winter, A. Ecer, F.N. Satofuka, P. Fox (eds.), Elsevier (2005), 391–399
9. M. Perić, A finite volume method for the prediction of three-dimensional fluid flow in complex ducts, Ph.D. Thesis, University of London (1985).
10. D.Petcu, D.Vizman, J.Friedrich, M.Popescu, Crystal growth simulation on clusters. In *Procs. of HPC2003*, I. Banicescu (ed.), Simulation Councils Inc. San Diego (2003), 41–46
11. P.A. Sackinger, R.A. Brown, J.J. Brown, A finite element method for analysis of fluid flow, heat transfer and free interfaces in Czochralski crystal growth, *Internat.J. Numer. Methods in Fluids* **9** (1989), 453–492.
12. H. L. Stone, Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Num. Anal.* **5** (1968), 530–558.
13. D. Sun, A Multiblock and Multigrid Technique for Simulations of Material Processing, Ph.D. Thesis, State University of New York (2001).

Parallelization of Numerical CFD Codes in Cluster and Grid Environments*

Jacek Rokicki, Marian Krause, and Michał Wichulski

Warsaw University of Technology,
Institute of Aeronautics and Applied Mechanics,
Nowowiejska 24, 00-665 Warsaw, Poland

Abstract. The paper discusses parallelization of codes and algorithms used in Computational Fluid Dynamics. A theoretical model is presented for parallelization based on domain decomposition. This model allows for quantitative prediction of acceleration and efficiency. Finally, certain unresolved problems related to grid environment are reported.

1 Introduction

Finite-volume or finite-element methods are routinely used in Computational Fluid Dynamics for discretization of Euler/Navier-Stokes equations which govern the fluid flow.

As a result, simulations consist of a long sequence of consecutive iterations which aim to solve a very large system of nonlinear algebraic equations. Typical computational meshes contain millions of cells, but even such detailed spatial resolution is still not sufficient to effectively predict certain phenomena.

Reduction of computational time is possible (at the present level of algorithm development) only through application of parallel processing.

The present paper deals with parallelization based on domain decomposition [3]. This means, that prior to the actual calculations, the computational mesh is divided into parts each of which is served by a different processor. The exchange of boundary information (on fictitious interfaces) occurs at every nonlinear iteration step of the main solution algorithm. This exchange is limited to the information available in the immediate neighborhood of the fictitious boundary.

In the present paper parallel efficiency is discussed. The model allows for quantitative predictions of acceleration and efficiency (as a function of both the problem size and the number of processors used).

The results and conclusions are formulated for the particular CFD problem, it is evident however that they remain valid for many physical problems described by nonlinear partial differential equations.

* This work has been supported in part by the Polish Ministry of Science and Information Society Technologies under grant 6T11 2003C/06098.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 825–830, 2006.

2 Parallel Algorithm

The parallel algorithm considered here, consist of the following steps:

1. Each node initializes calculations reading its own grid and restart files (the latter only in the restart mode) as well as the configuration file.
2. Communication is initialized by preparing and exchanging information about the size of the future data packets (for each pair of neighboring nodes).
3. Each node performs separately one (or more) iteration(s) of the local nonlinear solver (on its own part of the mesh).
4. The interfacial boundary information is exchanged between all neighboring nodes. The boundary conditions at each node are updated.
5. The convergence criterion is checked at each node, subsequently reduced (using logical AND) and the result is scattered to all nodes. If FALSE is returned, the control goes back to step 3.
6. Execution is terminated after each node stores the corresponding restart and solution output files.

In the above, the neighborhood is understood in the sense of mesh partition topology. The nodes are considered as neighbors if their meshes have a common interface (or if the meshes overlap as it was the case in [3]).

The main computational effort corresponds to the step 3 of the algorithm. This effort can be assumed proportional to the number of mesh cells, perhaps with the exception of the vicinity of the physical boundary, where the computational cost can be higher (this however will be disregarded in the analysis below).

The main communication effort is located in step 4 and is proportional to the length of the data packet (the latency is not significant in this case). The communication present in step 5 can be neglected as its volume is not significant.

The workload of each processor is proportional to the number of cells present in its local mesh. Thus with N denoting the total number of mesh cells and assuming ideal load balancing we can estimate the computation time on the L-processor system by:

$$\tau_{\mathrm{COMP}}[L] \;=\; \frac{A \cdot N}{L} \cdot \tag{1}$$

where A stands for the proportionality factor (which can depend both on the algorithm as well as on the processor speed).

3 Communication Models

Two possible communication models are considered here.

The first bases on the *master-worker* concept. In this approach one of the nodes, *master* is responsible for gathering, re-calculating and scattering data that is exchanged between nodes (*workers*) (see Fig. 1).

In contrast *neighbor to neighbor* approach assumes that communication exists between these nodes which have common interface, without the need for

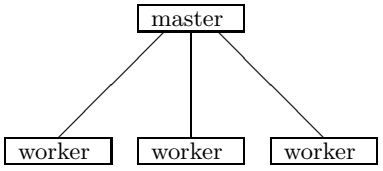


Fig. 1. *Master-worker* communication

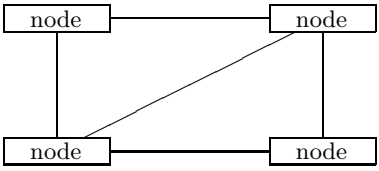


Fig. 2. *Neighbor to neighbor* communication

a centralized control (see Fig. 2). Here simultaneous exchange of data between various nodes allows for significant reduction of communication overhead. Both approaches can be rooted in the solution algorithm. Yet the success of implementation heavily depends on the communication hardware.

In particular in popular clusters, the computing nodes are in fact connected via a single switching device. Therefore efficiency of *neighbor to neighbor* communication will be limited by the switch ability to simultaneously transmit data between separate pairs of nodes. This hardware arrangement typical for clusters is present also in some shared memory architectures where fast switching devices connect computing nodes with separate memory banks (e.g., Compaq ES40).

For grids [1], which are built out of separate clusters, the analysis is more complicated due to the heterogeneous nature of the connection topology (in particular when computations are distributed over a number of local clusters).

4 Performance Analysis

The performance of algorithm described in Section 2 can be evaluated by considering average time $\tau[L]$ necessary to perform single iteration (steps 3, 4, 5) (L denotes number of processors). In the present analysis, it is tacitly assumed that the overall computational effort and total number of iterations do not significantly depend on the number of processors.

4.1 Communication Time

Communication time depends on the range of factors and in particular on:

- Numerical Algorithm
- Dimension of the computational problem (2D or 3D)
- Quality of partition into subdomains (is the communication volume equally distributed and minimized)
- Communication model *master-worker* or *neighbor to neighbor*
- Hardware properties.

All of these issues cannot be reasonably included into the present analysis. Instead we aim at obtaining some optimal estimation, e.g., when partition into subdomains minimizes and evenly distributes the communication volume.

We assume that the communication volume assigned to each processor is proportional to the number of interfacial (boundary) cells in each local mesh. In particular in 2D, the checker-board partition of a rectangular domain results in a number of interfacial cells (per computing node) proportional to $\sqrt{N/L}$. In contrast, partition into stripes increases this result to $\sqrt{N}$.

In 3D the partition of the cubic domain into smaller cubes gives the number of interfacial cells (per computing node) proportional to $(N/L)^{3/2}$. Again this partition can be regarded as optimal.

The communication time can thus be estimated as:

$$\tau_{\mathrm{COMM}}[L] = B_{d,\mu} \left(\frac{N}{L}\right)^{\frac{d-1}{d}} L^{\mu} + \tau_{\mathrm{LAT}}$$

where in the above: (i) τ_{LAT} stands for latency, (ii) $d = 2, 3$ denotes the dimension of the problem, (iii) μ is equal 0 in case of the successful *neighbor to neighbor* model and is equal 1 for the *master-worker* model, (iv) $B_{d,\mu}$ is a proportionality constant depending both on the space dimension as well as on the communication model.

It is worth stressing that for *master-worker* approach the communication time depends on the total communication volume (hence $\mu = 1$) whereas for the *neighbor to neighbor* model only local communication has to be taken into account (hence $\mu = 0$).

4.2 Parallel Efficiency

Taking into account all considered subcases parallel efficiency can be estimated as:

$$\eta[L] = \frac{1}{L} \frac{\tau_{\mathrm{COMP}}[1]}{\tau_{\mathrm{COMP}}[L] + \tau_{\mathrm{COMM}}[L]} = \frac{1}{\beta[L] + \frac{L}{AN}\left[BN^{1-1/d}L^{\mu-1/d} + \tau_{\mathrm{LAT}}\right]} \quad (2)$$

where for clarity reasons subscripts of the proportionality constant B are dropped.

For large problems when latency is negligible and by assuming ideal load balancing one obtains a simple formula for efficiency:

$$\eta[L] = \frac{1}{1 + L^{1+\mu-1/d}N^{-1/d} \cdot B/A} \quad (3)$$

Formula 3 allows in turn to evaluate the size of the problem (represented here by N) required to preserve a prescribed efficiency on an L-processor system

$$N[\eta, L] = \left(\frac{B}{A}\right)^{d} L^{d(1+\mu)-1} \left(\frac{\eta}{\eta - 1}\right)^{d} \quad (4)$$

In particular, for the *master-worker* model keeping fixed efficiency is possible by increasing problem size N proportionally to L^3 in 2D and to L^5 in 3D. For the *neighbor to neighbor* model the same effect is obtained if problem size grows as L^1 in 2D and to L^2 in 3D. This is why the latter seems so attractive from the point of view of the parallel program design.

The current formulas (3) and (4) form a generalization of the one presented and verified in [3] for $d = 2$, $\mu = 1$ and for $L = 2, \ldots, 21$.

5 Parallelization in Grid Environment

Traditional parallelization is characterized by the requirement to distribute computations over separate processors belonging to a single system. In such system the RAM memory is often divided and accessible only locally. MPI library [2] facilitates the necessary exchange of data.

In contrast in grid environment not only RAM but also a disk memory is distributed over two or more local systems. This itself is not a major drawback since Broker is designed to replicate and gather all data for the user.

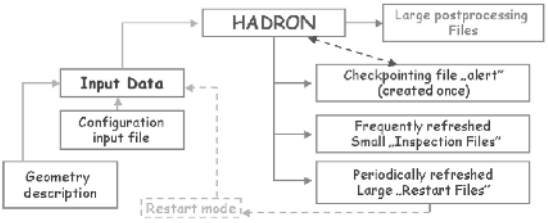


Fig. 3. File system used by the HADRON code both for checkpointing as well as to facilitate restart mechanism

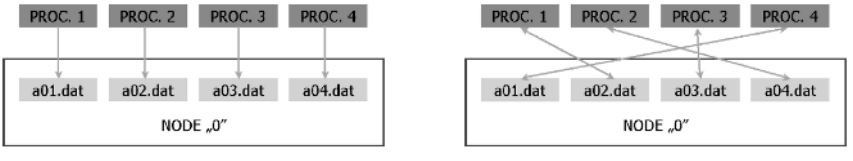


Fig. 4. Regular and restart modes on system with common disk memory

When computations, however, take a very long time to complete, the application is usually equipped with some restart mechanism (see Fig. 3 presenting the file system used by the CFD code HADRON). This means (as described in Section 2) that each processor stores to its own file, in regular intervals, a huge amount of data – while the file name is often parameterized by the process number (See Fig. 4). When the computations are terminated (either by accident, or because of encountering error or by the action of the user) these files are gathered by the Broker and transferred to a single location.

When the user decides to restart the computations the Broker should perform the inverse operation. In order to operate correctly each new process should have its own restart file available locally (the name of the file should match the process number). This number however is neither available in advance to the Broker, nor can be guessed by the application itself. As a consequence the application will not work correctly in the restart mode (See Fig. 5).

It is of course possible to add a mechanism to the application, which recognizes existing files and re-numbers computing nodes. This however is not very practical and should be solved in a different way.

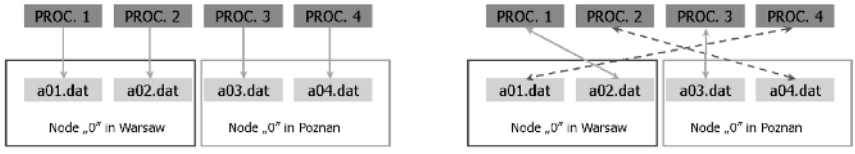


Fig. 5. Regular and restart modes on system with distributed disk memory (grid) – dashed lines describe incorrectly positioned files

Still other issue that remains open is related to the control of the running application. This usually requires checking the inspection files, while the program is running. In the grid environment the temporary inspection files, are as rule not accessible for the user - so the only possibility seems to be to redirect the data flow to the standard output.

On the other hand the code itself has its own checkpointing mechanism (see Fig. 3). Namely the code creates a tiny file *alert* at the beginning of computations. During execution the program checks regularly whether this file still exists. If it does not (the user may have just deleted the *alert* file to stop further execution) the code terminates its activity by writing down all restart and output files, enabling subsequent restarting without any data loss.

Again in the grid environment this action is not always supported, since this particular *alert* file can neither be accessed nor deleted by the user.

References

1. CLUSTERIX Project Home page, http://www.clusterix.pl
2. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press, Cambridge MA, 1995
3. J. Rokicki, J., Żółtak, J., Drikakis, D., and Majewski, J., Parallel Performance of Overlapping Mesh Techniques for Compressible Flows, Future Generation Computer Systems **19** (2001) 3–15

Execution of a Bioinformatics Application in a Joint IRISGrid/EGEE Testbed*

José Luis Vázquez-Poletti[1], E. Huedo[2],
Rubén S. Montero[1], and Ignacio M. Llorente[1,2]

[1] Departamento de Arquitectura de Computadores y Automática. Facultad de Informática, Universidad Complutense de Madrid. 28040 Madrid, Spain
[2] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain

Abstract. This paper describes the execution of a Bioinformatics application over a highly distributed and heterogeneous testbed. This testbed is composed of resources devoted to EGEE and IRISGrid projects and has been integrated by taking advantage of the modular, decentralized and "end-to-end" architecture of the Grid *W*ay framework. Results show the feasibility of building loosely-coupled Grid environments based only on Globus services, while obtaining non trivial levels of quality of service. Such approach allows a straightforward resource sharing as the resources are accessed by using *de facto* standard protocols and interfaces.

1 Introduction

Different Grid infrastructures are being deployed within growing national and transnational research projects. The final goal of these projects is to provide the end user with much higher performance than that achievable on any single site. However, from our point of view, it is arguable that some of these projects embrace the Grid philosophy, and to what extent. This philosophy, proposed by Foster [1], defines *a Grid* as a system (i) not subject to a centralized control and (ii) based on standard, open and general-purpose interfaces and protocols, (iii) while providing some level of quality of service (QoS), in terms of security, throughput, response time or the coordinated use of different resource types. In current projects, there is a tendency to ignore the first two requirements in order to get higher levels of QoS. However, these requirements are even more important because they are the key to the success of *the Grid*.

The Grid philosophy leads to computational environments, which we call *loosely-coupled Grids*, mainly characterized by [2]: autonomy (of the multiple administration domains), heterogeneity, scalability and dynamism. In a loosely-coupled Grid, the different layers of the infrastructure should be separated from

* This research was supported by Ministerio de Ciencia y Tecnología, through the research grants TIC 2003-01321 and 2002-12422-E, and by Instituto Nacional de Técnica Aeroespacial "Esteban Terradas" (INTA) – Centro de Astrobiología. The authors participate in the EGEE project, funded by the European Union under contract IST-2003-508833.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 831–838, 2006.

each other, being only communicated with a limited and well defined set of interfaces and protocols. This layers are [2]: Grid fabric, core Grid middleware, user-level Grid middleware, and Grid applications.

The coexistence of several projects, each with its own middleware developments, adaptations or extensions, arise the idea of using them simultaneously (from an user's viewpoint) or contribute the same resources to more than one project (from an administrator's viewpoint). One approach could be the development of gateways between different middleware implementations [3]. Other approach, more in line with the Grid philosophy, is the development of client tools that can adapt to different middleware implementations. We hope this could lead to a shift of functionality from resources to brokers or clients, allowing the resources to be accessed in a standard way and easing the task of sharing resources between organizations and projects. We should consider that the Grid not only involves the technical challenge of constructing and deploying this vast infrastructure, it also brings up other issues related to security and resource sharing policies [4] as well as other socio-political difficulties [5].

Practically, the majority of the Grid infrastructures are being built on protocols and services provided by the Globus Toolkit[1], becoming a *de facto* standard in Grid computing. Globus architecture follows an hourglass approach, which is indeed an "end-to-end" principle [6]. Therefore, instead of succumbing to the temptation of tailoring the core Grid middleware to our needs (since in such case the resulting infrastructure would be application specific), or homogenizing the underlying resources (since in such case the resulting infrastructure would be a highly distributed cluster), we propose to strictly follow the "end-to-end" principle. Clients should have access to a wide range of resources provided through a limited and standardized set of protocols and interfaces. In the Grid, these are provided by the core Grid middleware, Globus, just as, in the Internet, they are provided through the TCP/IP protocols. Moreover, the "end-to-end" principle reduces the firewall configuration to a minimum, which is also welcome by the security administrators.

One of the most ambitious projects to date is EGEE[2] (Enabling Grids for E-sciencE), which is creating a production-level Grid infrastructure providing a level of performance and reliability never achieved before. EGEE currently uses the LCG[3] (LHC Computing Grid) middleware, which is based on Globus. Other much more modest project is IRISGrid[4] (the Spanish Grid Initiative), whose main objective is the creation of a stable national Grid infrastructure. The first version of the IRISGrid testbed is based only on Globus services, and it has been widely used through the Grid*W*ay framework[5].

For the purposes of this paper we have used a Globus-based testbed to run a Bioinformatics application through the Grid*W*ay framework. This testbed was

[1] http://www.globus.org
[2] http://www.eu-egee.org
[3] http://lcg.web.cern.ch
[4] http://www.irisgrid.es
[5] http://www.gridway.org

built up from resources inside IRISGrid and EGEE projects. The aim of this paper is to demonstrate the application of an "end-to-end" principle in a Grid infrastructure, and the feasibility of building loosely-coupled Grid environments based only on Globus services, while obtaining non trivial levels of quality of service through an appropriate user-level Grid middleware.

The structure of the paper follows the layered structure of Grid systems, from bottom-up. The Grid fabric is described Section 2. Section 3 describes the core Grid middleware. Section 4 introduces the functionality and characteristics of the Grid*Way* framework, used as user-level Grid middleware. Section 5 describes the target application. Finally, Section 6 presents the experimental results and Section 7 ends up with some conclusions.

2 Grid Fabric: IRISGrid and EGEE Resources

This work has been possible thanks to the collaboration of those research centers and universities that temporarily shared some of their resources in order to set up a geographically distributed testbed. The testbed results in a very

Table 1. IRISGrid and EGEE resources contributed to the experiment

Testbed	Site	Resource	Processor	Speed	Nodes	RM
IRISGrid	RedIRIS	heraclito	Intel Celeron	700MHz	1	Fork
		platon	2×Intel PIII	1.4GHz	1	Fork
		descartes	Intel P4	2.6GHz	1	Fork
		socrates	Intel P4	2.6GHz	1	Fork
	DACYA-UCM	aquila	Intel PIII	700MHz	1	Fork
		cepheus	Intel PIII	600MHz	1	Fork
		cygnus	Intel P4	2.5GHz	1	Fork
		hydrus	Intel P4	2.5GHz	1	Fork
	LCASAT-CAB	babieca	Alpha EV67	450MHz	30	PBS
	CESGA	bw	Intel P4	3.2GHz	80	PBS
	IMEDEA	llucalcari	AMD Athlon	800MHz	4	PBS
	DIF-UM	augusto	4×Intel Xeon*	2.4GHz	1	Fork
		caligula	4×Intel Xeon*	2.4GHz	1	Fork
		claudio	4×Intel Xeon*	2.4GHz	1	Fork
	BIFI-UNIZAR	lxsrv1	Intel P4	3.2GHz	50	SGE
EGEE	LCASAT-CAB	ce00	Intel P4	2.8GHz	8	PBS
	CNB	mallarme	2×Intel Xeon	2.0GHz	8	PBS
	CIEMAT	lcg02	Intel P4	2.8GHz	6	PBS
	FT-UAM	grid003	Intel P4	2.6GHz	49	PBS
	IFCA	gtbcg12	2×Intel PIII	1.3GHz	34	PBS
	IFIC	lcg2ce	AMD Athlon	1.2GHz	117	PBS
	PIC	lcgce02	Intel P4	2.8GHz	69	PBS

* These resources actually present two physical CPUs but they appear as four logical CPUs due to hyper-threading.

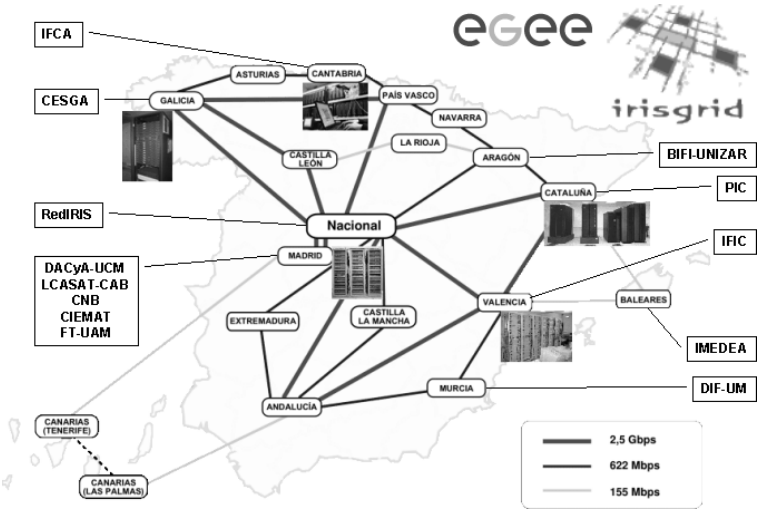


Fig. 1. Geographical distribution and interconnection network of sites

heterogeneous infrastructure, since it presents several middlewares, architectures, processor speeds, resource managers (RM), network links, etc. A brief description of the participating resources is shown in Table 1.

Some centers are inside IRISGrid, which is composed of around 40 research groups from different spanish institutions. Seven sites participated in the experiment by donating a total number of 195 CPUs. Other centers participate in the EGEE project, which is composed of more than 100 contracting and non-contracting partners. Seven spanish centers participated by donating a total number of 333 CPUs.

Together, the testbed is composed of 13 sites (note that LCASAT-CAB is both in IRISGrid and EGEE) and 528 CPUs. In the experiments below, we limited to four the number of jobs simultaneously submitted to the same resource, with the aim of not saturating the whole testbed, so only 64 CPUs were used at the same time. All sites are connected by RedIRIS, the Spanish Research and Academic Network. The geographical location and interconnection links of the different sites are shown in Figure 1.

3 Core Grid Middleware: Globus

Globus services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [7]. Table 2 summarizes the core Grid middleware components existing in both IRISGrid and EGEE resources used in the experiments. In the case of EGEE, we only used Globus basic services, ignoring any higher-level services, like the resource broker or the replica location service.

Table 2. Core Grid middleware

Component	IRISGrid	EGEE
Security Infrastructure	IRISGrid CA and manually generated `grid-mapfile`	DATAGRID-ES CA and automatically generated `grid-mapfile`
Resource Management	GRAM with shared home directory in clusters	GRAM without shared home directory in clusters
Information Services	IRISGrid GIIS and local GRIS, using the MDS schema	CERN BDII and local GRIS, using the GLUE schema
Data Management	GASS and GridFTP	GASS and GridFTP

We had to introduce some changes in the security infrastructure in order to perform the experiments. For authentication, we used a user certificate issued by DATAGRID-ES CA, so we had to give trust to this CA on IRISGrid resources. Regarding authorization, we had to add an entry for the user in the `grid-mapfile` in both IRISGrid and EGEE resources.

4 User-Level Grid Middleware: Grid *W*ay

User-level middleware is required in the client side to make it easier and more efficient the execution of applications. Such client middleware should provide the end user with portable programming paradigms and common interfaces.

In a Globus-based environment, the user is responsible for manually performing all the submission steps [7] in order to achieve any functionality. To overcome this limitation, Grid *W*ay [8] was designed with a *submit & forget* philosophy in mind. The core of the Grid *W*ay framework is a personal *submission agent* that performs all scheduling stages and watches over the correct and efficient execution of jobs on Globus-based Grids. The Grid *W*ay framework provides adaptive scheduling and execution, as well as fault tolerance capabilities to handle the dynamic Grid characteristics.

A key aspect in order to follow the "end-to-end" principle is how job execution is performed. In EGEE, file transfers are initiated by a job wrapper running in the compute nodes, therefore they act as client machines, so needing network connectivity and client tools to interact with the middleware. In Grid *W*ay, however, job execution is performed in three steps by the following modules:

1. *prolog*: It prepares the remote system by creating a experiment directory and transferring the input files from the client.
2. *wrapper*: It executes the actual job and obtains its exit status code.
3. *epilog*: It finalizes the remote system by transferring the output files back to the client and cleaning up the experiment directory.

This way, GridWay doesn't rely on the underlying middleware to perform preparation and finalization tasks. Moreover, since both *prolog* and *epilog* are submitted to the front-end node of a cluster and *wrapper* is submitted to a compute node, GridWay doesn't require any middleware installation nor network connectivity in the compute nodes.

Other projects [9, 10, 11, 12] have also addressed resource selection, data management, and execution adaptation. We do not claim innovation in these areas, but remark the advantages of our modular, decentralized and "end-to-end" architecture for job adaptation to a dynamic environment.

In this case, we have taken full advantage of the modular architecture of GridWay, as we didn't have to directly modify the source code of the *submission agent*. We extended the *resource selector* in order to understand the GLUE schema used in EGEE. The *wrapper* module also had to be modified in order to perform an explicit file staging between the front-end and the compute nodes in EGEE clusters.

5 Grid Application: Computational Proteomics

One of the main challenges in Computational Biology concerns the analysis of the huge amount of protein sequences provided by genomic projects at an ever increasing pace. In the following experiments, we will consider a Bioinformatics application aimed at predicting the structure and thermodynamic properties of a target protein from its amino acid sequence [13].

The algorithm, tested in the 5th round of Critical Assessment of techniques for protein Structure Prediction (CASP5)[6], aligns with gaps the target sequence with all the 6150 non-redundant structures in the Protein Data Bank (PDB)[7], and evaluates the match between sequence and structure based on a simplified free energy function plus a gap penalty item. The lowest scoring alignment found is regarded as the prediction if it satisfies some quality requirements. In such cases, the algorithm can be used to estimate thermodynamic parameters of the target sequence, such as the folding free energy and the normalized energy gap.

We have applied the algorithm to the prediction of thermodynamic properties of families of orthologous proteins, i.e. proteins performing the same function in different organisms. The biological results of the comparative study of several families of orthologous proteins are presented elsewhere [14].

6 Experiences and Results

The experiments presented here consist in the analysis of a family of 80 orthologous proteins of the *Triose Phosphate Isomerase* enzyme (an enzyme is a special case of protein). Five experiments were conducted in different days during a week. The average turnaround time for the five experiments was 43.37 minutes.

[6] http://PredictionCenter.llnl.gov/casp5/
[7] http://www.pdb.org

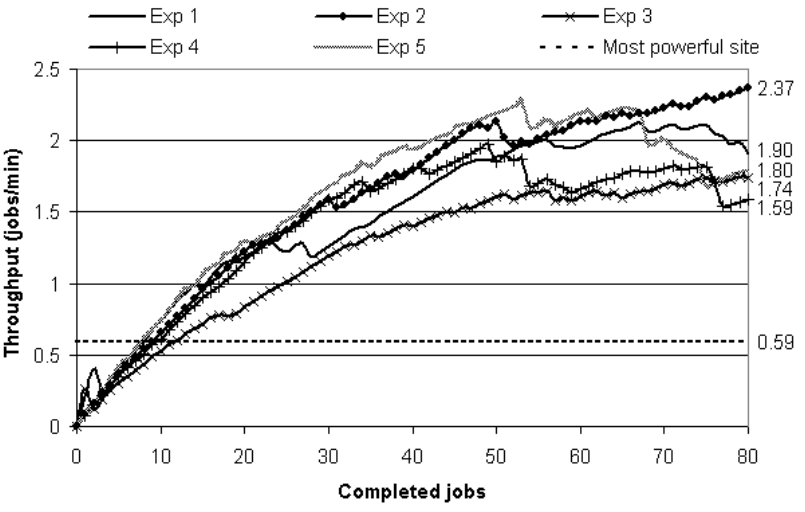


Fig. 2. Testbed dynamic throughput during the five experiments and theoretical throughput of the most powerful site

Figure 2 shows the dynamic throughput achieved during the five experiments alongside the theoretical throughput of the most powerful site, where the problem could be solved in the lowest time, in this case DIF-UM (taking into account that the number of active jobs per resource was limited to four). The throughput achieved on each experiment varies considerably, due to the dynamic availability and load of the testbed. For example, resource ce00 at site LCASAT-CAB was not available during the execution of the first experiment. Moreover, fluctuations in the load of network links and computational resources induced by non-Grid users affected to a lesser extent in the second experiment, as it was performed at midnight.

7 Conclusions

We have shown that the "end-to-end" principle works at the client side (i.e. the user-level Grid middleware) of a Grid infrastructure. Our proposed user-level Grid middleware, Grid*W*ay, can work with Globus, as a standard core Grid middleware, over any Grid fabric in a *loosely-coupled* way. The smooth process of integration of two so different testbeds, although both are based on Globus, demonstrates that the Grid*W*ay approach (i.e. the Grid way), based on a modular, decentralized and "end-to-end" architecture, is appropriate for the Grid.

Moreover, loosely-coupled Grids allow a straightforward resource sharing since resources are accessed and exploited through *de facto* standard protocols and interfaces, similar to the early stages of the Internet. This way, the loosely-coupled model allows an easier, scalable and compatible deployment.

Acknowledgments

We would like to thank all the institutions involved in the IRISGrid initiative and the EGEE project, in particular those who collaborated in the experiments. We would like to also thank Ugo Bastolla, staff scientist in the Bioinformatics Unit at Centro de Astrobiología (CAB) and developer of the Bioinformatics application used in the experiments.

References

1. Foster, I.: What Is the Grid? A Three Point Checklist. GRIDtoday **1**(6) (2002) Available at http://www.gridtoday.com/02/0722/100136.html.
2. Baker, M., Buyya, R., Laforenza, D.: Grids and Grid Technologies for Wide-Area Distributed Computing. Software – Practice and Experience **32**(15) (2002) 1437–1466
3. Allan, R.J., Gordon, J., McNab, A., Newhouse, S., Parker, M.: Building Overlapping Grids. Technical report, University of Cambridge (2003)
4. San José, O., Suárez, L.M., Huedo, E., Montero, R.S., Llorente, I.M.: Resource Performance Management on Computational Grids. In: Proc. 2nd Intl. Symp. Parallel and Distributed Computing (ISPDC 2003), IEEE CS (2003) 215–221
5. Schopf, J.M., Nitzberg, B.: Grids: The Top Ten Questions. Scientific Programming, special issue on Grid Computing **10**(2) (2002) 103–111
6. B. Carpenter, E.: RFC 1958: Architectural Principles of the Internet (1996)
7. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – The Global Grid Forum (2001)
8. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. Intl. J. Software – Practice and Experience (SPE) **34**(7) (2004) 631–651
9. Berman, F., Wolski, R., Casanova, H., et al.: Adaptive Computing on the Grid Using AppLeS. IEEE Trans. Parallel and Distributed Systems **14**(4) (2003) 369–382
10. Buyya, R., D.Abramson, Giddy, J.: A Computational Economy for Grid Computing and Its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems **18**(8) (2002) 1061–1074
11. Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S.: Condor/G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing **5**(3) (2002) 237–246
12. Vadhiyar, S., Dongarra, J.: A Performance Oriented Migration Framework for the Grid. In: Proc. 3rd Intl. Symp. Cluster Computing and the Grid (CCGrid 2003), IEEE CS (2003) 130–137
13. Huedo, E., Bastolla, U., Montero, R.S., Llorente, I.M.: A Framework for Protein Structure Prediction on the Grid. New Generation Computing **23**(4) (2005) 277–290
14. Bastolla, U., Moya, A., Viguera, E., van Ham, R.: Genomic Determinants of Protein Folding Thermodynamics in Prokaryotic Organisms. Journal of Molecular Biology **343**(5) (2004) 1451–1466

Load Balancing Strategies in a Web Computing Environment*

Olaf Bonorden, Joachim Gehweiler, and Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute, Computer Science Departement,
Paderborn University, 33095 Paderborn, Germany
{bono, joge, fmadh}@uni-paderborn.de

Abstract. We compare different load balancing strategies for Bulk-Synchronous Parallel (BSP) programs in a web computing environment. In order to handle the influence of the fluctuating available computation power, we classify the external work load.

We evaluate the load balancing algorithms using our web computing library for BSP programs in Java (PUBWCL). Thereby we simulated the external work load in order to have repeatable testing conditions.

With the best performing load balancing strategy we could save 39% of the execution time averaged and even up to 50% in particular cases.

1 Introduction

Utilizing the unused computation power in a web computing environment, one has to deal with unpredictable fluctuations of the available computation power on the particular computers. Especially in a set of strongly coupled parallel processes, one single process receiving little computation power can slow down the whole application. A way to balance the load is to migrate these "slow" processes. For this approach, we have implemented and analyzed four different load balancing strategies for the *Paderborn University BSP-based Web Computing Library (PUBWCL)*.

The rest of the paper is organized as follows: In Sections 2 and 3, we give a short overview of PUBWCL and describe the implemented load balancing strategies. In Section 4, we analyze the external work load. In Section 5, we discuss the results of the experiments we have conducted. Section 6 concludes this paper.

2 The Web Computing Library

The BSP model. The *Bulk-Synchronous Parallel (BSP)* model [7] has been introduced by Leslie G. Valiant in order to simplify the development of parallel algorithms. A *BSP computer* is defined as a set of processors with local memory, connected by a network capable of point-to-point communcation, and a barrier synchronization mechanism.

* Partially supported by DFG-SFB 376 "Massively Parallel Computation".

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 839–846, 2006.

A *BSP program* consists of a set of *BSP processes* and a sequence of *supersteps* separated by the barrier synchronization. Within a superstep each process performs local computations and sends messages to other processes; these messages can be accessed by their recipients after the barrier synchronization, i. e., in the next superstep.

PUBWCL. The *Paderborn University BSP-based Web Computing Library (PUBWCL)* [1, 2, 5] is a library for parallel algorithms designed according to the BSP model and intended to utilize the unused computation power on computers distributed over the internet. Participants willing to donate their unused computation power have to install a PUBWCL client. Whenever a user wants to execute a BSP program, the system chooses a subset of these clients to run the program.

To provide for load balancing, PUBWCL can migrate BSP processes during the barrier synchronization (and additionally at arbitrary points specified by the developer of a BSP program). This is accomplished using *JavaGo RMI* [6, 4]. We also use this technique to backup the program state once per superstep by migrating into a file. Thus BSP processes can be restarted on other clients, when a PC crashes or disconnects from the internet unexpectedly. This feature is also used to enable load balancing strategies to abort a BSP process if it does not finish within some deadline, and restart it on a faster client.

3 The Load Balancing Strategies

We can derive the following constraint from the properties of a BSP algorithm: Since all the BSP processes are synchronized at the end of each superstep, we can reduce the scheduling problem for a BSP algorithm with n supersteps to n subproblems, namely scheduling within a superstep.

We assume that we only have to deal with "good" BSP programs, i. e., all of the p BSP processes require approximately the same amount of computational work. Thus the scheduler has to assign p equally heavy pieces of work.

Finally, we have another restriction: Due to privacy reasons we cannot access the breakdown of the CPU usage, i. e., we especially do not know how much computation power is consumed by the user and how much computation power is currently assigned to our BSP processes.

Parameters for scheduling. Since we cannot directly access the breakdown of the CPU usage, we have to estimate (1) how much computation power the BSP processes currently assigned to a client do receive, and (2) how much computation power a BSP process would receive when (additionally) assigned to a client.

As from the second superstep on, the first question can simply be answered by the ratio of the computation time consumed during the previous superstep and the number of concurrently running BSP processes.

In order to answer the second question, all clients regularly measure the *Available Computation Power (ACP).* This value is defined as the computation

power an additionally started BSP process would receive on a particular client, depending on the currently running processes and the external work load. We obtain this value by regularly starting a short simulation process and measuring the computation power it receives. Though it is not possible to determine the CPU usage from the ACP value, this value is platform independent and thus comparable among all clients.

Since the first approach is more accurate, we will use it wherever possible, i. e., mainly, to decide whether a BSP process should migrate or has to be restarted. The ACP value will be used to determine the initial distribution of the BSP processes and to choose clients as migration targets and as hosts for restarted BSP processes.

3.1 The Load Balancing Algorithms

We have implemented and analyzed the following four load balancing strategies, among them two parallel algorithms and two sequential ones.

Algorithm PwoR. The load balancing algorithm *Parallel Execution without Restarts (PwoR)* executes all processes of a given BSP program concurrently.

The initial distribution is determined by dint of the ACP values. Whenever a superstep is completed, all clients are checked whether the execution of the BSP processes on them took more than r times the average execution duration (a suitable value for r will be chosen in Section 5); in this case the BSP processes are redistributed among the active clients such that the expected execution duration for the next superstep is minimal, using as little migrations as possible.

Algorithm PwR. Using the the load balancing algorithm *Parallel Execution with Restarts (PwR)*, the execution of a superstep is performed in phases. The duration of a phase is r times the running time of the $\lceil s \cdot p^* \rceil$-th fastest of the (remaining) BSP processes, where p^* is the number of processes of the BSP program that have not yet completed the current superstep. Suitable values for the parameters $r > 1$ and $0 < s < 1$ will be chosen in Section 5. At the end of a phase, all incomplete BSP processes are aborted. In the next phase, they are restored on faster clients. Whereas too slow BSP processes are migrated only after the end of the superstep using the PwoR algorithm, they are restarted already during the superstep using PwR.

At the end of each (except the last) superstep, the distribution of the BSP processes is optimized among the set of currently used clients by dint of the processes' execution times in the current superstep. The optimization of the distribution is performed such that the number of migrations is minimal.

Algorithm SwoJ. While the two load balancing strategies PwoR and PwR execute all BSP processes in parallel, the load balancing algorithm *Sequential Execution without Just-in-Time Assignments (SwoJ)* executes only one process of a BSP program per client at a time; the other BSP processes are kept in queues.

Like PwR, SwoJ operates in phases. At the end of a phase all uncompleted BSP processes are aborted and reassigned. Thereby the end of a phase is reached after

r times the duration, in which the x-th fastest client has completed all assigned BSP processes, where x is a fraction s of the number of the affected clients.

At the end of a superstep, the distribution of the BSP processes is optimized like in PwR.

Algorithm SwJ. Like SwoJ, the load balancing algorithm *Sequential Execution with Just-in-Time Assignments (SwJ)* executes only one process of a BSP program per client at a time and keeps the other processes in queues, too. The main difference, however, is that these queues are being balanced. More precisely, whenever a client has completed the execution of the last BSP process in its queue, a process is migrated to it from the queue of the most overloaded client (if there exists at least one overloaded client). Thereby a client is named "overloaded" in relation to another client if completing all but one BSP processes in the queue with the current execution speed of the particular client would take longer than executing one BSP process on the other client. Thereby the execution speed of a client is estimated by dint of the running time of the BSP process completed on it most recently.

BSP processes are aborted only if the corresponding queues on all affected clients are empty and if the processes do not complete within r times the duration of a process on the x-th fastest client, weighted by the number of BSP processes on the particular clients; thereby, again, x is a fraction s of the number of the affected clients.

4 The External Work Load

In order to understand the fluctuation of the external work load, we have analyzed totaling more than 100 PCs in altogether four departments of three German universities over a period of 21 resp. 28 days. The CPU frequencies varied from 233 MHz to 2.8 GHz. The installed operating systems were Debian Linux, RedHat Enterprise Linux, and SuSE Linux.

While analyzing the load, we have noticed that the CPU usage typically shows a continuous pattern for quite a time, then changes abruptly, then again shows a continuous pattern for some time, and so on. The reason therefore is that many users often perform uniform activities (e. g. word processing, programming, and so on) or no activity (e. g. at night or during lunch break).

A given CPU usage graph (e. g. of the length of a week) can thus be split into blocks, in which the CPU usage is somewhat steady or shows a continuous pattern. These blocks typically have a duration of some hours, but also durations from only half an hour (e. g. lunch break) up to several days (e. g. a weekend) do occur.

Based on the above observations we have designed a model to describe and classify the external work load. We describe the CPU usage in such a block by a rather tight interval with radius $\alpha \in \mathbb{R}$ ($\alpha < \frac{1}{2}$) around a median load value $\lambda \in \mathbb{R}$ ($0 \leq \lambda - \alpha$, $\lambda + \alpha \leq 1$), as illustrated in Fig. 1. The rates for the upper and lower deviations are bounded by $\beta^+ \in \mathbb{R}$ resp. $\beta^- \in \mathbb{R}$ ($\beta^+, \beta^- < \frac{1}{2}$). We will refer to such a block as a $(\lambda, \alpha, \beta^+, \beta^-, T)$-*load sequence* in the following.

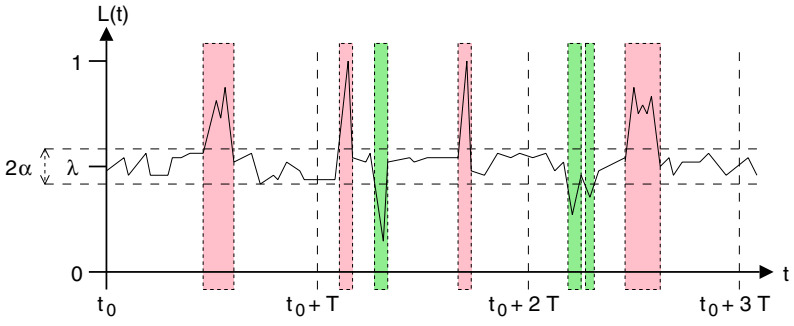


Fig. 1. A three-load-period-long interval of a load sequence

In order to describe the frequency and duration of the deviations, we subdivide the load sequences into small sections of length T, called *load periods*. The values β^+ and β^- must be chosen such that the deviation rates never exceed them for an arbitrary starting point of a load period within the load sequence.

Given that T is much shorter than the duration of a superstep, we can obtain this result:

Theorem 1. *If a superstep is executed completely within a* $(\lambda, \alpha, \beta^+, \beta^-, T)$*-load sequence, the factor between the minimal and maximal possible duration is at most* $q' \in \mathbb{R}^+$ *with:*

$$q' := \frac{1 - (1 - \beta^-)(\lambda - \alpha)}{1 - (\beta^+ + (1 - \beta^+)(\lambda + \alpha))}$$

For $q \in \mathbb{R}^+$, $q \geq q'$ *we call a* $(\lambda, \alpha, \beta^+, \beta^-, T)$*-load sequence* q-bounded.

The proof can be found in [3]. This result guarantees that the running times of BSP processes, optimally scheduled based on the execution times of the previous superstep, differ at most by a factor q^2 within a load sequence. This fact will be utilized by the load balancing strategies.

Evaluating the collected data. When sectioning a given CPU usage sequence into load sequences, our goal is to obtain load sequences with a q-boundedness as small as possible and a duration as long as possible, while the rate of unusable time intervals should be as small as possible. Obviously, these three optimization targets depend on each other. We have processed the data collected from our PCs described in the beginning of this section (over 6.8 million samples) with a Perl program which yields an approximation for this non-trivial optimization problem.

The results. The average idle time over a week ranged from approx. 35% up to 95%, so there is obviously a huge amount of unused computation power.

Time intervals of less than half an hour and such where the CPU is nearly fully utilized by the user or its usage fluctuates too heavily, are no candidates for a load sequence. The rate of wasted idle time in such intervals is less than 3%.

Choosing suitable values for the parameters of the load sequences, it was possible to section the given CPU usage sequences into load sequences such that the predominant part of the load sequences was 1.6-bounded.

On most PCs, the average duration of a load sequence was 4 hours or even much longer. Assuming the execution of a process, started at an arbitrary point during a load sequence, takes 30 minutes, the probability that it completes within the current load sequence is thus 87.5% or higher.

A detailed analysis of the results in each of the four networks can be found in [3].

Generating load profiles. In order to compare the load balancing strategies under the same circumstances, i. e., especially with exactly the same external work load, and to make the experimental evaluation repeatable, we have extracted totaling eight typical load profiles from two of the networks, each using these time spans: Tuesday forenoon (9:00 a.m. to 1:00 p.m.), Tuesday afternoon (2:00 p.m. to 6:00 p.m.), Tuesday night (2:00 a.m. to 6:00 a.m.), and Sunday afternoon (2:00 p.m. to 6:00 p.m.). Besides, we have generated four artificial load profiles according to our model, using typical values for the parameters. A detailed discussion of the load profiles can be found in [3].

5 Experimental Evaluation

We have conducted our experiments on 15 PCs running Windows XP Professional, among them 7 PCs with 933 MHz and 8 ones with 1.7 GHz. The PUB-WCL server and the client used to control the experiments ran on a seperate PC.

We have simulated the external work load according to the load profiles mentioned above and run the clients with the `/belownormal` priority switch, i. e., they could only consume the computation power left over by the load simulator.

The experiments were performed using a BSP benchmark program consisting of 80 equally weighted processes and 8 identical supersteps. Per superstep, each BSP process did $2 \cdot 10^9$ integer operations and sent (and received) 64 messages of 4 kB size each.

Results using PwoR. First we have to choose a suitable value for parameter r: At the beginning of a superstep the BSP processes are (re-) distributed over the clients such that they should complete execution at the same time. Supposing that, on any of the involved PCs, the current (q-bounded) load sequence does not end before completion of the superstep, none of the BSP processes should take longer than q times the average execution time. That means, choosing $r = q = 1.6$ guarantees that BSP processes are not migrated if the available computation power only varies within the scope of a q-bounded load sequence.

In comparison to experiments with no load balancing algorithm (i. e. initial distribution according to the ACP values and no redistribution of the processes during runtime), we could save 21% of the execution time averaged and even up to 36% in particular cases.

Comparing the execution times of the particular supersteps, we noticed that the execution time significantly decreases in the second superstep. The reason

therefore is that the execution times of the previous superstep provide much more accurate values for load balancing than the estimated ACP values.

Results using PwR. Our experiments with the PwR algorithm resulted in noticealby longer execution times than those with the PwoR algorithm. We could obtain the best results with the parameters set to $r = 2$ and $s = \frac{1}{8}$; other choices led to even worse results.

On the one hand, this result is surprising as one would expect that PwR performs better than PwoR because it restarts BSP processes after some threshold instead of waiting for them for an arbitrarily long time. But on the other hand, restarting a BSP process is of no advantage if it would have completed on the original client within less time than its execution time on the new client. Our results show that the external work load apparently is not 'sufficiently malicious' for PwR to take advantage of its restart feature.

Results using SwoJ. Like with the PwR algorithm, $r = 2$ and $s = \frac{1}{8}$ is a good choice for the parameters because: In Section 4 we showed that a load sequence does not end inside a superstep with a probability of 87.5% or more. Thus the probability that a new load sequence with *more* available computation power starts inside a superstep is about $\frac{1}{16}$ or less. As we will use the normalized BSP process execution time on the x-th fastest client (where x is a fraction s of the number of affected clients) as a reference value for the abortion criterion, we ensure that no new load sequence has begun on this client with high probability by setting $s = \frac{1}{8}$ (instead of $s = \frac{1}{16}$). Provided that no new load sequence begins during the superstep, the factor between the fastest and the slowest normalized BSP process execution time on the particular clients is at most q^2. For a q-boundedness of $q = 1.6$ this yields $q^2 = 2.56$. We have actually chosen $r = 2$ because of our defensive choice of s.

Using these parameters, we could save 14% of the execution time averaged and even up to 25% in particular cases in comparison to the experiments with the PwoR algorithm; the savings in comparison to the experiments with no load balancing algorithm were even 32% averaged and up to 45% in isolated cases.

Results using SwJ. For the choice of the parameters, the same aspects as in the SwoJ case apply. In comparison to our experiments with the SwoJ algorithm we could save another 10% of the execution time averaged and even up to 27% in isolated cases; the savings in comparison to the experiments with no load balancing algorithm were even 39% averaged and up to 50% in particular cases.

6 Conclusion

The load balancing strategy SwJ performs better than SwoJ which, in turn, performs better than PwoR (cf. Fig. 2). In comparison to using no load balancing, we can save up to 50% of the execution duration using SwJ.

In order to achieve even better results, we are working on an extension of PUBWCL which allows redundant execution of BSP processes, i. e., processes

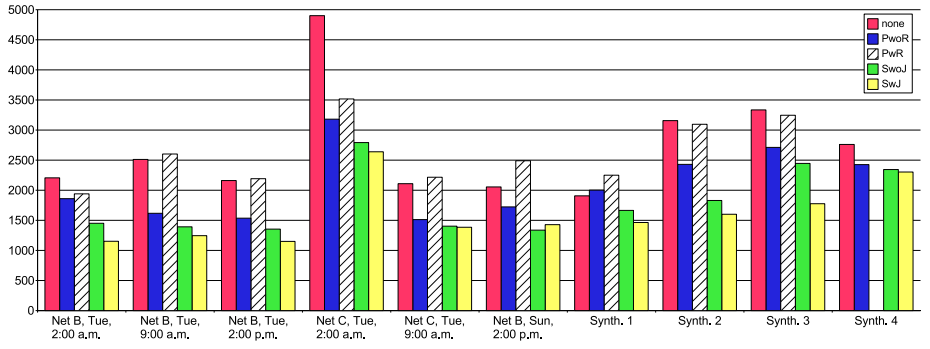


Fig. 2. Running times depending on the load balancing algorithm

are started redundantly, but only the results of the fastest one are committed whereas the remaining processes are aborted when the first one completes. This will allow us to improve the load balancing strategies by starting additional instances of slow BSP processes on faster clients instead of just restarting them. Since, using the SwJ algorithm, typically only a very small fraction of the BSP processes was restarted, this would mean only a low overhead but significantly reduce the probability that they would have to be restarted another time.

Furthermore, work is in progress to realize PUBWCL as a pure peer-to-peer system in order to dispose of the bottleneck at the server.

References

1. Bonorden, O., Gehweiler, J., Meyer auf der Heide, F.: A Web Computing Environmnent for Parallel Algorithms in Java. Proceedings of PPAM 2005, to appear.
2. Gehweiler, J.: Entwurf und Implementierung einer Laufzeitumgebung für parallele Algorithmen in Java. Studienarbeit, Universität Paderborn, 2003.
3. Gehweiler, J.: Implementierung und Analyse von Lastbalancierungsverfahren in einer Web-Computing-Umgebung. Diplomarbeit, Universität Paderborn, 2005.
4. JavaGo RMI. http://www.joachim-gehweiler.de/software/javago.php
5. The Paderborn University BSP-based Web Computing Library (PUBWCL). http://wwwcs.uni-paderborn.de/∼pubwcl/
6. Sekiguchi, T., Masuhara, H., Yonezawa, A.: A Simple Extension of Java Language for Controllable Transparent Migration and its Portable Implementation. Technical Report, University of Tokyo, 1999.
7. Valiant, L.: A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.

Multi-installment Divisible Load Processing in Heterogeneous Systems with Limited Memory

Maciej Drozdowski[1,2,⋆] and Marcin Lawenda[2,3]

[1] Institute of Computing Science, Poznań University of Technology,
ul. Piotrowo 3A, 60-965 Poznań, Poland
[2] Faculty of Mathematics and Computer Science, Adam Mickiewicz University,
ul. Umultowska 87, 61-614 Poznań, Poland
Maciej.Drozdowski@cs.put.poznan.pl
[3] Poznań Supercomputing and Networking Center,
ul. Noskowskiego 10, 61-704 Poznań, Poland
Marcin.Lawenda@man.poznan.pl

Abstract. Optimum divisible load processing in heterogeneous star system with limited memory is studied. We propose two algorithms to find multi-installment load distribution: Exact branch-and-bound algorithm and a heuristic using genetic search method. Characteristic features of the solutions and the performance of the algorithms are examined in a set of computational experiments.

1 Introduction

Divisible load model represents computations with fine granularity, and negligible dependencies between the grains of computations. Consequently, the computations, or the load, can be divided into parts of arbitrary sizes, and these parts can be processed independently in parallel. Divisible load theory (DLT) proved to be a versatile vehicle in modeling distributed systems [3, 4, 10].

In this paper we assume a star communication topology. Heterogeneous processors (or workers, clients) receive load from a central server (or an originator, master) only. The load is sent in multiple small chunks, rather than in one long message. Since the amount of processor memory is limited, the accumulated load cannot exceed the memory limit. The problem is to find the sequence of processor communications, and the sizes of the load chunks such that the length of the schedule is minimum. We propose two algorithms for this problem. Exact branch and bound algorithm, and a heuristic based on genetic search.

Similar problems have already been studied. Multi-installment distribution was proposed in [2, 3], but the sequence of communications was fixed, and memory limits were not considered. Memory limitations and single installment communications were studied in [5, 6, 7, 9]. For a fixed communication sequence a fast heuristic was proposed in [9], and an optimization algorithm based on linear programming was given in [5]. A hierarchic memory system was studied in [6].

⋆ This research has been partially supported by the Polish State Committee for Scientific Research.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 847–854, 2006.

A multi-installment load distribution with a fixed communication pattern was proposed to overcome out-of-core memory speed limitations. It was shown in [7] that finding optimum divisible load distribution in a system with limited memory sizes and affine communication delay is **NP**-hard. In [11] multi-installment divisible load processing with limited memory was studied, but the computer system was homogeneous and communication sequence was fixed. In this paper we assume that the sequence of communications can be arbitrary. The load is processed in multiple chunks by a heterogeneous system. To our best knowledge, none of the already published papers addressed the exact problem we study.

The rest of the paper is organized as follows. In Section 2 problem is formulated, in Section 3 solution methods are outlined, and results of computational experiments are reported in Section 4.

2 Problem Formulation

A set of processors $\{P_1, \ldots, P_m\}$ is connected to a central server P_0. By a processor we mean a processing element with CPU, memory, and communication hardware. Each processor P_i is defined by the following parameters: communication link startup time S_i, communication transfer rate C_i, processing rate A_i, and memory limit B_i. The time to transfer x load units from P_0 to P_i is $S_i + xC_i$. The time required to process the same amount of load is xA_i. Let n denote the number of load chunks sent by the originator to the processors. We will denote by α_j the size of chunk j. The total amount of load to process is V. Hence, $\sum_{j=1}^{n} \alpha_j = V$. The problem consists in finding the set of used processors, the sequence of their activation, and the sizes of the load chunks α_j such that schedule length C_{max}, including communication and computations, is the shortest possible. For the sake of conciseness we will mean both selecting the set of processors and their activation sequence while saying activation sequence. The optimum activation sequence must gear to the speeds of the processors, and available memory. Let d_j be the index of the destination processor for load chunk j, and $\overline{d} = (d_1, \ldots, d_n)$ a vector of processor destinations.

It is assumed that the amount of memory available at the processor is limited. If the new chunks arrive faster than the load is processed, then the load may accumulate in the processor memory. The method of memory management has influence on the conditions that must be met to satisfy memory limitations. Below we discuss some options. In all cases we assume that memory is allocated before the communication with the arriving load starts.

1) When load chunk j starts arriving, a memory block of size $\alpha_j \leq B_{d_j}$ is allocated. After processing chunk j the memory block is released to the operating system. This approach was assumed in the earlier papers [5, 6, 7, 9, 11]. Unfortunately, though each chunk uses less memory than B_{d_j}, the total accumulated memory consumption may be bigger. Hence, this method is not very effective in the case of multi-installment processing.

2) When load chunk j starts arriving, many small blocks of memory are allocated from the memory pool. The size of each small block is equal to the size

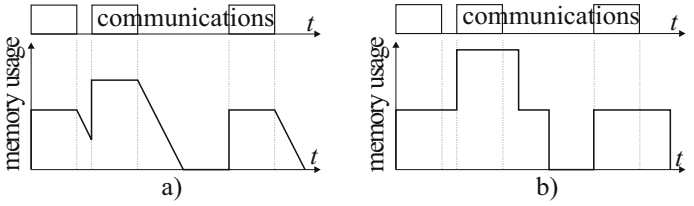


Fig. 1. Memory usage for a) management method 2, b) management method 3

of the grain of parallelism. The total allocated memory size is α_j. As processing of the load progresses, the memory blocks are gradually released to the operating system. This method of memory management is illustrated in Fig.1a.

3) As in the first case, memory block of size α_j is allocated when chunk j starts arriving. It is released after processing chunk j. However, it is required that the total memory allocated on the processor never exceeds limit B_{d_j}. This method of memory management is illustrated in Fig.1b.

For the sake of simplicity of mathematical representation, in this work we use the second method of memory management. For the same reason we assume that: the time of returning the results of computations is negligible, and that processors cannot compute and communicate simultaneously. Consequently, computations are suspended by communications. We assume that the sequence of processor destinations $\overline{d}$ is given. Hence, we know number n_i of load chunks sent to processor P_i, and function $g(i,k) \in \{1,\ldots,n\}$ which is the global number of a chunk sent to processor P_i as k-th for $k = 1,\ldots,n_i$. Let t_j denote the time moment when sending load chunk j starts. We will denote by x_{ik} the amount of load that accumulated on processor P_i at the moment when communication k to P_i starts. The problem of the optimum chunk size selection can be formulated as the following linear program:

$$\min \quad C_{max}$$

$$t_j + S_{d_j} + \alpha_j C_{d_j} \le t_{j+1} \qquad j = 1,\ldots,n-1 \tag{1}$$

$$x_{i,k-1} + \alpha_{g(i,k-1)} - \frac{t_{g(i,k)} - (t_{g(i,k-1)} + S_i + C_i\alpha_{g(i,k-1)})}{A_i} \le x_{ik}$$
$$i = 1,\ldots,m, k = 2,\ldots,n_i \tag{2}$$

$$x_{ik} + \alpha_{g(i,k)} \le B_i \qquad i = 1,\ldots,m, k = 1,\ldots,n_i \tag{3}$$

$$t_{g(i,n_i)} + S_i + C_i\alpha_{g(i,n_i)} + A_i(\alpha_{g(i,n_i)} + x_{in_i}) \le C_{max} \quad i = 1,\ldots,m \tag{4}$$

$$\sum_{j=1}^{n} \alpha_j = V \tag{5}$$

$$x_{i1} = 0, x_{ik} \ge 0 \qquad i = 1,\ldots,m, k = 2,\ldots,n_i \tag{6}$$

$$t_j, \alpha_j \ge 0 \qquad j = 1,\ldots,n \tag{7}$$

In the above formulation inequality (1) guarantees that communications do not overlap. The amount of load accumulated on P_i at the moment when chunk k

starts arriving must satisfy inequality (2). By inequality (3) memory limit is not exceeded. Computations finish before the end of the schedule by constraint (4), and the whole load is processed by equation (5). The above formulation can be adjusted to the case when simultaneous receiving a new load chunk and computation on the previous one is possible. In such a situation the following constraint should be added: $x_{ik} \geq \alpha_{g(i,k-1)} - \frac{1}{A_i}(t_{g(i,k)} - (t_{g(i,k-1)} + S_i + C_i\alpha_{g(i,k-1)}))$. We conclude that the optimum load distribution can be found provided that the sequence of processor communications $\overline{d}$ is given. In the next section we propose two methods of constructing $\overline{d}$.

3 Selecting Processor Activation Sequence

3.1 Branch and Bound Algorithm

Branch-and-bound (BB) algorithm is an enumerative method that generates and searches the space of possible solutions while eliminating these subsets of solutions which are infeasible or worse than some already known solution.

The search space consists of possible sequences $\overline{d}$. Solutions are generated expanding partial sequence $\overline{d(i)} = (d_1, \ldots, d_i)$, for $i < n$, by adding all possible destinations $d_{i+1} \in \{1, \ldots, m\}$, until obtaining complete sequences of length n. The generation of the sequences can be imagined as a construction of a tree with at most m^n leaves. Some branches of the tree can be pruned. Therefore, for each partial sequence $\overline{d(i)}$ a lower bound on the length of all $\overline{d(i)}$'s descendants is calculated. The lower bound is C_{max} obtained from linear program (1)-(7) by assuming that each load chunk $i+1, \ldots, n$ is sent to an ideal processor. An ideal processor P_{id} has all the best parameters in the processor set, i.e. $A_{id} = \min_{i=1}^{m}\{A_i\}, C_{id} = \min_{i=1}^{m}\{C_i\}, S_{id} = \min_{i=1}^{m}\{S_i\}, B_{id} = \max_{i=1}^{m}\{B_i\}$. If the lower bound is greater than or equal to the length of some already known solution then there is no hope that any of $\overline{d(i)}$ descendants will improve the schedule. If the resulting linear program (1)-(7) is infeasible then it means that volume V is greater than the available processor memory. In both cases $\overline{d(i)}$ is not expanded, and in this way the search tree is pruned.

3.2 Genetic Algorithm

Genetic algorithm (GA) is a randomized search method which implicitly discovers the optimum solution by randomly combining pieces of good solutions [8]. Here we present basics of our implementation of GA only.

A set of G solutions is a population. Solutions are encoded as strings $\overline{d} = (d_1, \ldots, d_n)$ of chunk destinations. The quality of solution $\overline{d}$ is the schedule length $C_{max}(\overline{d})$ obtained as a solution of the linear program (1)-(7) formulated for $\overline{d}$.

Solutions of the population are subject of genetic operators. We used three operators: crossover, mutation, and selection. Single-point crossover was applied. The number of offspring generated by the crossover is Gp_C, where p_C is a tunable parameter which we will call crossover probability. Mutation changes Gnp_M randomly selected chunk destinations in the whole population. p_M is a tunable

parameter which we call mutation probability. In the selection operation a new population is assembled. The best half of the old population solutions is always preserved. For the second half of the population some solution $\overline{d_j}$ is selected with probability $\frac{1}{C_{max}(\overline{d_j})} / \sum_{j=1}^{G} \frac{1}{C_{max}(\overline{d_j})}$. The populations are modified iteratively. The number of iterations is bounded by an upper limit on the total number of iterations, and an upper limit on the number of iterations without quality improvement.

4 Computational Experiments

4.1 Experiment Setting

BB and GA were implemented in Borland C++ 5.5 and tested in a set of computational experiments run on a PC computer with MS Windows XP. Linear programs were solved using simplex code derived from `lp_solve` [1]. Unless stated otherwise the instance parameters were generated with uniform distribution from ranges $[0, 2]$ for parameters A, C, S, and $[0, \frac{2V}{n}]$ for parameter B. Infeasible instances with $(n \max_{i=1}^{m}\{B_i\}) < V$ were discarded. We applied the following GA tuning procedure. 100 random instances with $m = 4$, and $n = 8$ were generated and solved by BB and GA. The average relative distance of GA solutions from the optimum calculated by BB was a measure of the tuning quality. First, population size $G = 40$ was selected as increasing G beyond 40 did not improve solution quality significantly. Second, crossover probability $p_C = 50\%$, and then mutation probability $p_M = 5\%$ were selected for which best quality was obtained in minimum number of iterations. Finally, the limits of iteration numbers 100 (without quality improvement), and 1000 (in total) were selected for which solution distance from optimum was better than 0.1%.

Now let us discuss features of the optimum solutions. In Fig.2 an optimum schedule constructed by BB algorithm is presented. It is a typical situation that memory buffers are filled to the maximum capacity. We observed that if the number of messages is small then memory buffers were empty when a new message arrived (i.e. $x_{ik} = 0$). With the increasing number of messages n we observed an increase in the number of the optimum solutions in which the old load is

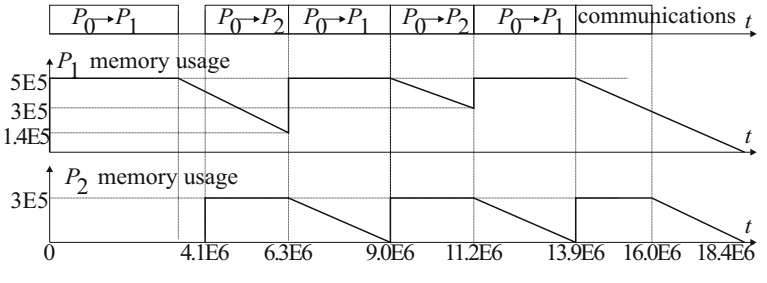


Fig. 2. Solution for $m = 2, n = 6, V$ =2E6, $A_1 = A_2 = 8.98, C_1 = C_2 = 7.39, S_1 = 2.01, S_2 = 3.02, B_1$ =5E5, B_2 =3E5

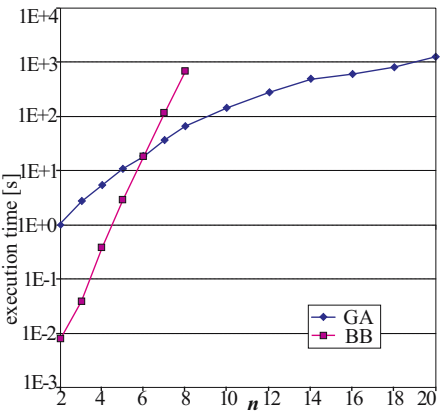


Fig. 3. Execution time vs. n, for $m = 4$

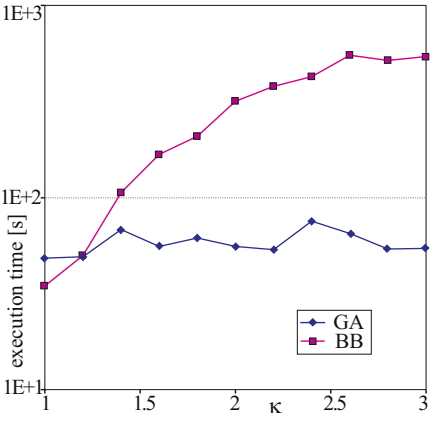


Fig. 4. Execution time vs. memory size, for $m = 4, n = 8, V = 1E6$

not completely processed on the arrival of the new load ($x_{ik} > 0$). Intuitively, this seems reasonable because when n is small each message must carry nearly a maximum load. If a message sent to processor P_i carries maximum load B_i, then the old load must be completely processed before receiving a new chunk.

4.2 Running Times

Dependence of the BB and GA execution times on n, B are shown in Fig.3, Fig.4, respectively. Each point in these diagrams is an average of at least ten instances. The worst case number of leaves visited in a BB search tree is m^n. Thus, the execution time of BB is exponential in n for fixed m (Fig.3). The execution time of GA grows with n because the length of the solution encoding and sizes of linear programs increase with n (Fig.3). GA running time dependence on m is weaker: 10-fold increase of m resulted in 60% increase of the execution time. In Fig.4 dependence of the running time on memory size is shown. For this diagram processor memory sizes (B_i) were generated with uniform distribution from range $[0, \kappa\frac{2V}{n}]$, where κ is shown along horizontal axis. With growing κ the size of available memory is growing on average. Consequently, more solutions are feasible, less branches can be cut in BB, and BB execution time grows. When κ is big, infeasibility of a solution becomes rare, and it is not limiting BB search tree. Hence, dependence of BB execution time on κ levels-off.

4.3 Quality of the Solutions

We observed that GA is very useful in deriving optimum, and near-optimum solutions. Over 55% of the instances were solved to the optimality by GA. The biggest observed relative distance from the optimum was 1.2%.

In Fig.5, Fig.6 we show dependence of the solutions quality on the range of C, B, respectively. Along the vertical axis a relative distance from the optimum

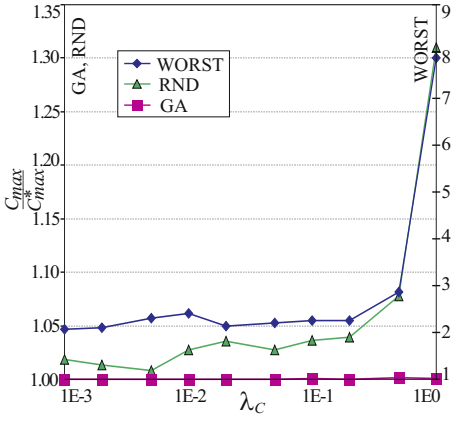


Fig. 5. Quality of the solutions vs. range of C, $m = 4, n = 8, V = 1\text{E}6$

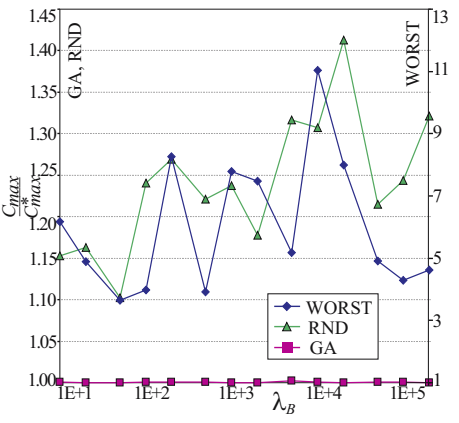


Fig. 6. Quality of the solutions vs. range of memory sizes, $m = 4, n = 8, V = 1E6$

is shown for three kinds of solutions: an average for the genetic algorithm (denoted GA), an average for a randomly selected sequence of destinations (RND), and for the worst case sequence ever observed (WORST). Load distributions for RND, WORST sequences were found from (1)-(7). A dedicated axis is used for the relation WORST. RND, and WORST are indicators of the characteristic features inherent in the problem itself. For Fig.5, the communication rates C_i were generated from range $[1-\lambda_C, 1+\lambda_C]$. The remaining parameters were generated as described above. Thus, with growing λ_C heterogeneity of the communication system was growing. Values of λ_C are shown along the horizontal axis. As it can be seen in Fig.5 with growing heterogeneity of parameter C the quality of both random and the worst case solutions is decreasing. Note, that this dependence is growing especially fast when C variation (λ_C) is big. For Fig.6 the memory sizes were generated from range $[\frac{2V}{n} - \lambda_B, \frac{2V}{n} + \lambda_B]$, for fixed value of V. The value of λ_B is shown along the horizontal axis. A trend of decreasing solution quality can be observed for growing λ_B. Note that in Fig.6 the distance from the optimum of RND, and WORST solutions is bigger than in Fig.5. Similar distance has been observed in the case of varying A. This demonstrates that narrowing the diversity of the system parameters simplifies obtaining a good solution, and communication rate C is a key parameter in performance optimization. Furthermore, the distance between WORST, RND, and GA or BB solutions can be used as an estimate of the gain from finding the optimum, or near-optimum, sequence of processor activations. It can be inferred that this kind of gain is $\approx$ 10-40% on average (RND), and $\approx$ 10-fold in the worst case.

5 Conclusions

In this paper we studied optimum multi-installment divisible load processing in a heterogeneous distributed system with limited memory. A linear programming

formulation has been proposed for a fixed processor activation sequence. Two algorithms were proposed to find an optimum, or near optimum processor activation sequences. The algorithm running times, and quality of the solutions were compared in a series of computational experiments. It turned out that the proposed genetic algorithm is very effective in finding near-optimum solutions. The impact of the system heterogeneity on the solution quality has been also studied. It appears that with growing system heterogeneity good solutions are harder to be found. Especially small communication speed diversity simplifies obtaining good solutions.

References

1. Berkelaar, M.: `lp_solve` - Mixed Integer Linear Program solver. `ftp://ftp.es.ele.tue.nl/pub/lp_solve` (1995)
2. Bharadwaj, V., Ghose, D., Mani, V.: Multi-installment Load Distribution in Tree Networks With Delays, IEEE Transactions on Aerospace and Electronic Systems **31** (1995) 555-567
3. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos CA (1996)
4. Drozdowski, M.: Selected problems of scheduling tasks in multiprocessor computer systems,. Poznań Univ. of Technology, Series: Monographs, No 321, Poznań (1997). Downloadable from `http://www.cs.put.poznan.pl/mdrozdowski/txt/h.ps`
5. Drozdowski, M., Wolniewicz, P.: Divisible load scheduling in systems with limited memory. Cluster Computing **6** (2003) 19-29
6. Drozdowski, M., Wolniewicz, P.: Out-of-Core Divisible Load Processing. IEEE Trans. on Parallel and Distributed Systems **14** (2003) 1048-1056
7. Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory, accepted for publication in European Journal of Operational Research (2004)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
9. Li, X., Bharadwaj, V., Ko, C.C.: Optimal divisible task scheduling on single-level tree networks with buffer constraints. IEEE Trans. on Aerospace and Electronic Systems **36** (2000) 1298-1308
10. Robertazzi, T.: Ten reasons to use divisible load theory, IEEE Computer **36** (2003) 63-68
11. Wolniewicz, P., Drozdowski, M.: Processing Time and Memory Requirements for Multi-installment Divisible Job Processing. In R.Wyrzykowski (et al. eds.): Proceedings of 4th Int. Conf. PPAM 2001. Lecture Notes in Computer Science, Vol. 2328. Springer-Verlag, Berlin Heidelberg New York (2002) 125-133

Chromatic Scheduling of 1- and 2-Processor UET Tasks on Dedicated Machines with Availability Constraints*

Krzysztof Giaro and Marek Kubale

Department of Algorithms and System Modeling,
Gdańsk University of Technology, Poland
{giaro, kubale}@eti.pg.gda.pl

Abstract. We address a generalization of the classical 1- and 2-processor UET scheduling problem on dedicated machines. In our chromatic model of scheduling machines have non-simultaneous availability times and tasks have arbitrary release times and due dates. Also, the versatility of our approach makes it possible to generalize all known classical criteria of optimality. Under these constraints we show that the problem of optimal scheduling of sparse instances can be solved in polynomial time.

1 Introduction

In recent years, a number of new approaches to the problem of parallel computer systems have been proposed. One of them is scheduling of *multiprocessor* task systems [6]. According to this model any task may require for its processing more than one processor at a time. There are two main classes of problems in multiprocessor task scheduling. In the first class of problems, it is assumed that the *number* of simultaneously required processors is important [2, 15] and a task requires a certain prespecified number of processors. In the second class of multiprocesor scheduling problems, the *set* of simultaneously required processors is assumed to be important [1, 3, 13, 14]. In this case either a certain *fixed set* of processors, or a *family* of processor sets on which the task can be executed is given. This paper is concerned with a fixed set scheduling problem.

Furthermore, we assume that for each task the above-mentioned fixed set is either 1-element or 2-element. In other words, we assume that each task is either uniprocessor (it requires a single dedicated processor) or biprocessor (it requires two dedicated procesors simultaneously). For brevity, we will name uniprocessor tasks as 1-*tasks* and biprocessor tasks as 2-*tasks*.

Since the problem of scheduling 2-tasks is NP-hard subject to all classical optimality criteria, in this paper we are concerned with a special case that the duration of every task is the same. We will call such tasks *unit execution time* (UET) tasks. Consequently, all data are assumed to be positive integers and

* This research was partially supported by KBN grant 4T11C 04725.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 855–862, 2006.

deterministic. Later on we will see that such a restriction, although remaining NP-hard instances in general, does allow for polynomial-time algorithms in numerous special cases.

The third important assumption concerns availability constraints. Namely, we assume that the availability of tasks is restricted and, in addition, some machines are available only in certain intervals called *time windows*. Time windows may appear in the case of computer breakdowns or maintenance periods. Moreover, in any multitasking computer system and in hard real-time systems in particular, urgent tasks have high priority and are pre-scheduled in certain time intervals, thus creating multiple time windows of availability. Scheduling in time windows was considered in e.g. [1, 2, 8].

The model of scheduling considered in this paper is justified by various technological and efficiency reasons. For example, in fault-tolerant systems tasks may be duplicated to obtain results surely [7]. Mutual testing of processors needs two processors working in parallel [12]. The same can be said about file transfers, which require two corresponding processors simultaneously: the sender and the receiver [4]. Another example is resource scheduling in a batch manufacturing system, where jobs require two dedicated resources for processing [5].

The remaining of this paper is organized as follows. In Section 2 we set up the problem more formally and model it as a list edge-coloring problem. Section 3 is devoted to the case when there are fewer tasks than processors. We show that our problem can be solved efficiently by a tree coloring technique. In Section 4 we generalize these results to the case when the number of tasks is $O(1)$ greater than the number of processors.

2 Mathematical Model

A collection $J = \{J_1, \ldots, J_j, \ldots, J_n\}$ of n tasks has to be executed by m identical processors $M_1, \ldots, M_i, \ldots, M_m \in M$. Each task J_j requires the simultaneous use of a set fix_j of one or two prespecified (unique) processors for its execution but each processor can execute at most one such task at a time. Repetition of tasks is not allowed. Task $J_j, j = 1, \ldots, n$ requires processing during a given time p_j. All tasks are independent, nonpreemptable and of the same length. For the sake of simplicity we assume that $p_j = 1$ for $j = 1, \ldots, n$. Time is divided into unit-length time slots. Due to availability constraints, each task J_j has a list $L(J_j)$ of time slots during which the task is available for processing. Each slot on list $L(J_j)$ has its own weight (cost), independent of other weights of slots of all the lists. Our aim is to check whether a solution exists, and if the answer is affirmative, we are interested in finding a schedule with the minimum total cost. This criterion of optimality generalizes all known classical criteria: $C_{\max}$, $L_{\max}$, $T_{\max}$, $\overline{F}$, $\sum w_j C_j$, $\sum w_j L_j$, $\sum w_j T_j$, $\sum w_j U_j$, etc.

Using the three-field notation we can describe our problem as $P|win, p_j = 1, |fix_j| \leq 2|criterion$, where we add a word *win* in the second field since time windows are imposed on tasks rather than machines. The word *criterion* in the third field is used to emphasize the versatility of our approach.

The problem considered here can be modeled as list edge-coloring of a pseudograph $G = (V, E)$. There is a one-to-one correspondence between the vertex set $V = \{v_1, \ldots, v_m\}$ and the processor set M as well as the edge set $E = \{e_1, \ldots, e_n\}$ and the task set J, in the sense that a loop $e_j = \{v_i\} \in E$ corresponds to 1-task J_j to be executed on M_i, and edge $e_j = \{v_i, v_k\} \in E$ corresponds to 2-task J_j to be executed on machines M_i and M_k. Such a pseudograph will be called a *scheduling graph*. To each edge e of G there is assigned a list $L(e) \subseteq N$ (where N is the set of positive integers) specifying which slots (colors) are available to e. Moreover, to each $e \in E$ there is assigned an arbitrary (not necessary increasing) *cost function* $f_e \colon L(e) \to N \cup \{0\}$ specifying the cost of coloring edge e with a particular color $i \in N$. The functions f_e are assumed to be computable in $O(1)$ time. The sum of costs among all edges is the *cost of coloring* of graph G. We will attempt to minimize this parameter. It is easy to see that any solution to the $P|win, p_j = 1, |fix_j| \leq 2|$- is equivalent to a list edge-coloring of the associated scheduling graph G. Therefore, from here on we will speak interchangeably of colorings and schedules.

Illustration of the above concepts is given in Fig. 1. We show over there an example of problem instance, the corresponding scheduling graph and its optimal coloring as well as Gantt diagram of the optimal solution. As already mentioned, we will consider sparse scheduling graphs, i.e. with $|E| = |V| + O(1)$, since such graphs allow polynomial-time scheduling algorithms. In terms of scheduling this means that we restrict instances to those in which the number of tasks n is $O(1)$ greater than the number of processors m. More precisely, the scheduling graphs that we consider may have two kinds of cycles: short 1-edge cycles (loops), and long cycles of 3 or more edges. In the next section we consider scheduling graphs without long cycles. Section 4 is devoted to scheduling graphs with few long cycles.

3 Scheduling Graphs Without Long Cycles

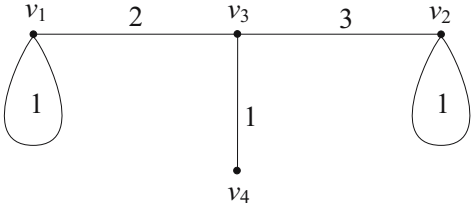
In the following by $\gamma(G)$ we denote the *cyclomatic number* of G, i.e. $|E|-|V|-\lambda+l$, where λ is the number of loops and l is the number of connected components of G. In this section we consider *forests*, i.e. scheduling graphs with $\gamma(G) = 0$. The case of graphs with $\gamma(G) \leq k$ will be considered in the next section. Suppose that a scheduling graph G is in shape of forest, i.e. it is a collection of trees $T_1, T_2, \ldots$. In this case we can color the forest by coloring each T_i separately. For this reason it suffices to consider the problem of coloring the edges of a tree T. Accordingly, let T be a tree on m vertices with $\lambda \leq m$ loops $e_{i_1}, e_{i_2}, \ldots, e_{i_\lambda}$. We replace each loop $e_{i_j} = \{v_l\}$ by a pendant edge $e_{i_j} = \{v_l, v_{m+j}\}$, $j = 1, ..., \lambda$, thus obtaining a new tree with the same number of edges and λ new vertices, but now there are no loops at vertices. Such a transformation is polynomial and does not change the problem, i.e. optimal coloring of the new tree is equivalent to optimal coloring of T.

The authors showed in [10] the following

(a) $M = \{M_1, M_2, M_3, M_4\}$
$J = \{J_1, J_2, J_3, J_4, J_5\}$

$J_1 \rightarrow M_1$	$L(J_1) = (1,2)$	$f_1(x) = x$
$J_2 \rightarrow M_2$	$L(J_2) = (1,3)$	$f_2(x) = 5x$
$J_3 \rightarrow M_1, M_3$	$L(J_3) = (1,2,3)$	$f_3(x) = 5\|x-2\|$
$J_4 \rightarrow M_2, M_3$	$L(J_4) = (2,3,4)$	$f_4(x) = x$
$J_5 \rightarrow M_3, M_4$	$L(J_5) = (1,3)$	$f_5(x) = 4x$

(b)



optimal cost = 13

(d)

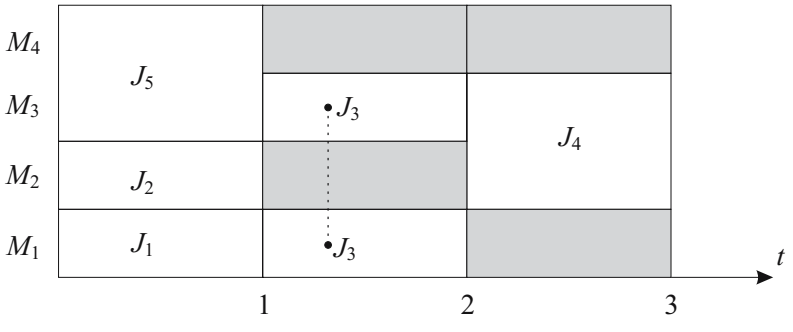


Fig. 1. Example: (a) problem instance; (b) scheduling graph; (c) Gantt diagram

Lemma 1. *For every fixed k the cost edge-coloring problem with increasing cost functions for graphs G with the cyclomatic number at most k can be solved in time $O(m\Delta^{1.5+k}\log(mC))$, where $C = \max_{e\in E(G)} f_e(2\Delta-1)$ and $\Delta \leq m$ stands for the maximum vertex degree in G.* □

We will show that Lemma 1 can be generalized to list cost coloring of such graphs with arbitrary cost functions. Our approach develops a technique for finding optimal edge sum coloring of trees [9] and uses an improvement useful in reducing its time complexity [16].

Theorem 1. *An optimal list cost coloring of the edges of tree T can be found in time $O(m\Delta^2\log(mC))$, where $C = \max_{e\in E(G), i\in L(e)} f_e(i)$.*

Proof. First of all, we say that a collection of lists L is *exact* for a graph G if $\forall_{\{u,v\}\in E}|L(\{u,v\})| = \deg(u) + \deg(v) - 1$. It is easy to see that there is a simple polynomial time reduction of a general list-cost coloring problem to its subcase of exact lists L (in this reduction we only have to delete the most expensive colors from lists "too long" and add appropriate new "very expensive" colors to lists "too short"). So in the following we can assume that L is exact.

We shall sketch a procedure for determining the minimum cost of list coloring of a tree T. One can easily extend it to finding the corresponding coloring. For a pair of adjacent vertices v and v' let H stand for the largest subtree of T such that $\{v, v'\}$ belongs to H and v is a leaf. Function $Val(v, v') : L(\{v, v'\}) \to \{0\} \cup N$ is defined so that $Val(v, v')(i)$ is the minimal cost of $L|_{E(H)}$-list coloring of the edges of H with the same cost functions as in T, in which edge $\{v, v'\}$ gets color i. If we recursively find the values of $Val(u, u')$ for a leaf u of T, then the requested optimal cost will be equal to $\min_{i\in L(\{u,u'\})} Val(u, u')(i)$.

If H contains only one edge $\{v, v'\}$ then obviously $Val(v, v') = f_{\{v,v'\}}$. So suppose that in the succeeding step of the procedure we have a subtree H as shown in Fig. 2 and the functions $Val(v', w_l), l = 1, .., k$ are already known.

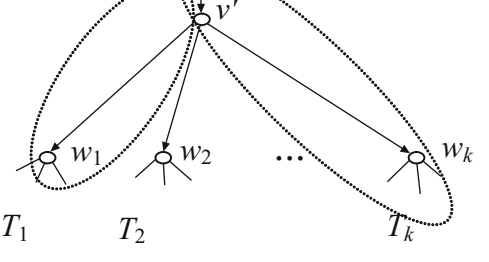


Fig. 2. Tree H and its subtrees

To find the value of $Val(v, v')$ we construct a bipartite graph $K^{v,v'}$ whose vertices in the first partition are $w_1, ..., w_k$ and the vertices in the second partition are colors $L(\{w_1, v'\}) \cup \cdots \cup L(\{w_k, v'\})$. There is an edge $\{w_l, x\}$ in $K^{v,v'}$ if and only if $x \in L(\{w_l, v'\})$ and then we put the weight $Val(v', w_l)(x)$ on it. In that case $Val(v, v')(j)$ is equal to the weight of the lightest k-edge matching in subgraph $K^{v,v'} - \{j\}$ (i.e. the cost of coloring of $T_1, \ldots, T_k$) plus $f_{\{v,v'\}}(j)$, since edge $\{v, v'\}$ has to be given color j.

Finding the value of $Val(v, v')$ requires prior calculating the weights of k-edge matchings in all graphs $K^{v,v'} - \{j\}$ for $j \in L(\{v, v'\})$. By using the algorithm mentioned in [11] for graph $K^{v,v'}$, for which the numbers of vertices and edges are $O(\deg_T(v')\Delta(T))$, we can do this in polynomial time $O(\deg_T(v')^{3/2}\Delta(T)^{3/2} \log(m\Delta(T)C))$. We get the overall complexity of the coloring algorithm by summing up among all vertices v', which results in $O(m\Delta^2 \log(mC))$. □

Reformulating Theorem 1 in terms of scheduling and introducing the symbol $G = forest$ to mean that the associated scheduling graph of the system is a forest, we obtain the following easy instance of our basic problem.

Theorem 2. *The $P|win, p_j = 1, |fix_j| \leq 2, G = forest|criterion$ problem can be solved in polynomial time, where 'criterion' stands for any of the following: $C_{\max}$, $\sum w_j C_j$, $L_{\max}$, $\sum w_j L_j$, $T_{\max}$, $\sum w_j T_j$, $\sum w_j U_j$, etc.*

Proof. We will show how to minimize the solution subject to the first two criteria: $C_{\max}$, $\sum w_j C_j$. Finding an optimal solution with respect to the remaining criteria can be accomplished similarly.

Case 1: $criterion = \sum w_j C_j$. For each edge $e \in E(G)$ we set $f_e(i) = iw_e, i = 1, 2, \ldots$. Next we find an optimal coloring of the forest G by repeatedly finding an optimal coloring of its trees.

Case 2: $criterion = C_{\max}$. For each edge $e \in E(G)$ we set $f_e(i) = i$, $i \in N$. We find an optimal coloring of the forest G. Let l be the length of this solution. We remove colors $s \geq l$ from all lists $L(e_1), L(e_2), \ldots$ and invoke the procedure for optimal list edge-coloring of G once more. If it fails, the previous solution is the best possible. If not, we continue the process of shrinking a schedule, possibly using a binary search to speed up the procedure. □

4 Scheduling Graphs with Few Long Cycles

In this section we assume that $n - m = O(1)$. As previously, suppose that graph G is loopless and connected, and the cyclomatic number $\gamma(G) \leq k$. Under these assumptions we have the following

Theorem 3. *For every fixed k there is a polynomial-time algorithm of complexity $O(m\Delta^{2+k} \log(mC))$, where $C = \max_{e \in E(G), i \in L(e)} f_e(i)$ for a list cost coloring of the edges of G with the cyclomatic number at most k.*

Proof. Let $G(V, E)$ be a graph with $\gamma(G) = k$, L its exact lists and let $A = \{e_1, ..., e_k\}$ be a set of the edges whose deletion form G results in a spanning tree. Moreover, let $U = \cup_{e \in A} e$ be the set of all vertices incident with the edges of A. All we need is an $O(m\Delta(G)^2 \log(mC))$-time procedure which for a given $L|_A$-list coloring d: $A \to N \cup \{0\}$ of graph (U, A) finds its cost optimal extension to an L-list edge-coloring of G. We will use the algorithm of Theorem 1. On the basis of graph G, the collection of lists L and cost functions we build a tree T, as follows.

Step 1: Delete the edges of A from G.

Step 2: For each deleted edge $\{u, v\} \in A$ introduce two new pendant edges $\{u, u_{new}\}$, $\{v, v_{new}\}$ with 1-element lists of the form: $L^T(\{v, v_{new}\}) = L^T(\{u, u_{new}\}) = \{d(\{u, v\})\}$. Next, define two cost functions for them, namely: one $f^T_{\{v,v_{new}\}} = f_{\{v,u\}}$, and the other $f^T_{\{u,u_{new}\}} = 0$.

Step 3: Leave costs and lists of the remaining edges unchanged.

The optimal cost of list edge-coloring of T and the extension of function d to list cost coloring of G are equal: edges $\{u, u_{new}\}$ and $\{v, v_{new}\}$ correspond to the 'halves' of split edge $\{u, v\}$ and 1-element lists of these pendant edges enforce the validity of the extension of d.

As far as the complexity considerations are concerned note that the number of edge-colorings of graph (U, A) is less than $(2\Delta)^k$, which is $O(\Delta^k)$ since k is fixed. Moreover, $|V(T)| = |V(G)| + 2k$ and $\Delta(T) = \Delta(G)$, which completes the proof. □

As previously, on the basis of Theorem 3 we get the following easy instance of our scheduling problem.

Theorem 4. *For every fixed k the $P|win, p_j = 1, |fix_j| \leq 2, G = k\text{-}cyclic|$ criterion problem can be solved in polynomial time.* □

Can the chromatic approach presented in the paper be extended to other instances of the UET multiprocessor tasks scheduling? Consider the same problem but with 1-tasks, 2-tasks and 3-tasks. Now 3-tasks can be modeled by hyperedges consisting of three vertices. Thus our pseudograph becomes a hypergraph with 3 kinds of hyperedges: loops, edges and triangles. Clearly, no two hyperedges having a vertex in common can be colored the same. If, in addition to general sparseness of the system, the number of such triangles is constant, then we may apply the same technique as that used in the proof of Theorem 3. Consequently, we arrive at a polynomial-time algorithm for optimal scheduling of k-processor ($k \leq 3$) UET tasks on dedicated processors with arbitrary availability constraints.

References

1. L. Bianco, J. Błażewicz, P. Dell'Olmo, M. Drozdowski, "Preemptive multiprocessor task scheduling with release times and time windows", *Ann. Oper. Res.* 70 (1997) 43-55.
2. J. Błażewicz, P. Dell'Olmo, M. Drozdowski, P. Mączka, "Scheduling multiprocessor tasks on parallel processors with limited availability", *Euro. J. Oper. Res.* 149 (2003) 377-389.
3. J. Błażewicz, P. Dell'Olmo, M. Drozdowski, M.G. Speranza, "Scheduling multiprocessor tasks on three dedicated processors", *Infor. Process. Lett.* 41 (1992) 275-280; Corrigendum IPL 49 (1994) 269-270.
4. E.G. Coffman, Jr., M.R. Garey, D.S. Johnson, A.S. LaPaugh, "Scheduling file transfers", *SIAM J. Comput.* 14 (1985) 744-780.
5. G. S. Dobson, U.S. Karmarkar, "Simultaneous resource scheduling to minimize weighted flow times", *Oper. Res.* 37 (1989) 592-600.
6. M. Drozdowski, "Scheduling multiprocessor tasks - An overview", *J. Oper. Res.* 94 (1996) 215-230.
7. E.F. Gehringer, D.P. Siewiorek, Z. Segall, *"Parallel Processing: The Cm* Experience*, Digital Press, Bedford (1987).
8. A. Gharbi, M. Haouari, "Optimal parallel machines scheduling with availability constraints", *Disc. Appl. Math.* 148 (2005) 63-87.

9. K. Giaro, M. Kubale, "Edge-chromatic sum of trees and bounded cyclicity graphs", *Infor. Process. Lett.* 75 (2000) 65-69.
10. K. Giaro, M. Kubale, K. Piwakowski, "Complexity results on open schop scheduling to minimize total cost of operations", *Intern. J. Comp. Sys. Sign.* 3 (2002) 84-91.
11. M. Kao, T. Lam, W. Sung, H. Ting, "All-cavity maximum matchings", *Proc. Inf. Syst.* E-87 (2004) 364-373.
12. H. Krawczyk, M. Kubale, " An approximation algorithm for diagnostic test scheduling in multicomputer systems", *IEEE Trans. Comp.* 34 (1985) 869-872.
13. M. Kubale, "The complexity of scheduling independent two-processor tasks on dedicated processors", *Infor. Process. Lett.* 24 (1987) 141-147.
14. M. Kubale, "Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated prosessors", *Euro. J. Oper. Res.* 94 (1996) 242-251.
15. E.L. Lloyd, "Concurrent task systems", *Oper. Res.* 29 (1981) 189-201.
16. X. Zhou, T. Nishizeki, "Algorithms for the cost edge-coloring of trees", *LNCS* 2108 (2001) 288-297.

Task Scheduling for Look–Ahead Reconfigurable Systems in Presence of Conditional Branches

Eryk Laskowski[1] and Marek Tudruj[1,2]

[1] Institute of Computer Science, Polish Academy of Sciences,
01-237 Warsaw, Ordona 21, Poland
[2] Polish-Japanese Institute of Information Technology,
02-801 Warsaw, Koszykowa 86, Poland
{laskowsk, tudruj}@ipipan.waw.pl

Abstract. A new program structuring algorithm for dynamically look–ahead reconfigurable multi–processor systems is presented in the paper. The presented algorithm uses a new kind of graph representation of parallel programs with conditional branches (Branching Task Graph, BTG). The BTG captures the *data–flow* and *control–flow* properties of parallel programs. It extends the scope of parallel programs optimized for execution in look–ahead reconfigurable systems beyond static DAG graphs. The new program graph structuring algorithm for BTG graphs is based on a two–phase approach. It consists of a new list task scheduling heuristics, which incorporates branch optimization techniques such as detection of mutually–exclusive subgraphs and scheduling of most–often–used paths based on branch probabilities. In the second phase, program partitioning into sections executed with the look–ahead created connections is done, based on the modified iterative clustering heuristics.

1 Introduction

This paper extends the scope of the program scheduling algorithms designed for the look–ahead configured systems so far [3, 4] and presents new program scheduling algorithms which contain conditional branches. These algorithms use new parallel program graph representations, which allow for expression of both data and control dependencies among program tasks. Based on these representations, new heuristics that encounter the existence of conditionally executed computations and communication have been proposed for task scheduling and graph partitioning. The proposed algorithms extend the algorithmic methods based on list scheduling and iterative task clustering assumed in [3, 4], with heuristics influenced by the most–often–used–path approach [7]. The algorithms determine at the compile time the program schedule, the partition into sections and the number of crossbar switches that provide time transparency of inter–processor connection reconfiguration.

The look–ahead dynamic connection reconfiguration is based on anticipated connection setting in some redundant communication resources provided in the system. These redundant resources can be link connection switches (crossbar

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 863–870, 2006.

switches, multistage connection networks), processor sets and processor link sets. In the paper, we investigate a system with multiple crossbars as redundant communication resources. The new parallel program graph representation, which captures both data and control flow, extends the scope of parallel algorithms that could be optimized and executed in such a kind of environment.

The paper consists of three parts. The first part describes the look–ahead dynamically reconfigurable system and parallel program model. In the second part, we introduce new program graph representations used for new formulation of the program scheduling problem. The third part presents program structuring algorithms for the assumed architecture.

2 Parallel Program Representation

In the paper we investigate a multiprocessor system with distributed memory and with communication based on message passing, Fig. 1. Inter–connections between processors $P_1 \ldots P_N$ are set in crossbar switches $S_1 \ldots S_X$ by a reconfiguration control subsystem (RCS) in parallel with program execution (including communication) and remain fixed during section execution. RCS collects messages on the section execution states in worker processors sent via the Control Communication Path. A hardware Synchronization Path synchronizes the states of processors which will execute next program sections. At section boundaries, relevant processor's communication links are switched to the crossbar switch(es) where the connections had been prepared.

Program scheduling and partitioning algorithms for the look–ahead reconfigurable system presented so far [3, 4] use static, directed acyclic graphs (DAGs) as a representation of application programs. Application programs often contain both data and control statements and thus cannot be effectively represented by a classic DAG. This problem has been noticed and several parallel program representations, which capture data and control dependencies have been proposed,

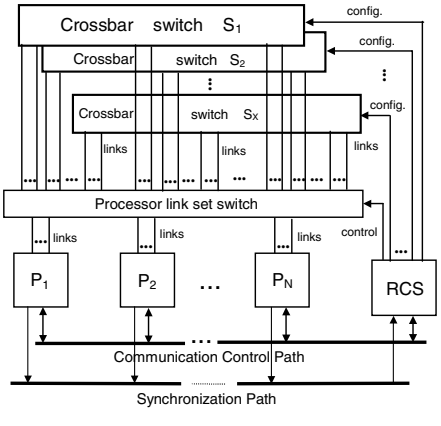


Fig. 1. Look–ahead reconfigurable system with multiple connection switches

especially in the system synthesis area [5, 6]. In the paper we present the BTG graph, a new effective representation of data and control flow in parallel programs. The BTG makes it possible to extend parallel program optimizations for execution in look–ahead reconfigurable systems, which are now not limited to classical DAG graphs with fully static control.

A Branching Task Graph (BTG) of a parallel program is a weighted directed acyclic graph $G(V, V_C, E)$. The BTG consists of two kinds of nodes. Node $n_i \in V$ represents a task, an atomic unit of execution. The weight of the node represents the task execution time. Node $c_k \in V_C$, $V_C \subset V$, represents a conditional branch, in which the control flow of a program forks according to the value of a control statement, computed in the node. Tasks execute according to the *macro–dataflow* model. A directed edge $e_{i,j} \in E$ represents communication that corresponds to data or control dependencies among nodes n_i and n_j. The weight of an edge is the communication cost. Alternative paths starting from a conditional node meet in a conditional merge node $c_m \in V_C$. Such paths can not be connected to any other subgraphs of a program. A merge node is activated after data from one of the alternative paths have arrived.

We assume the BTG graph is static and deterministic. The probability distribution of different values of each conditional statement is known prior to the program execution. A BTG of an exemplary program is shown in Fig. 2. Task nodes are shown as circles, conditional nodes are shown as diamonds. Execution times are written below node's numbers. Edge's labels show the communication costs.

A schedule is defined as task–to–processor and communication–to–link assignment, with specification of starting time of each task and each communication. In the look–ahead reconfigurable environment, the schedule also consists of program partition into sections, executed with the look–ahead created connections. In the presented algorithms, a program with a specified schedule

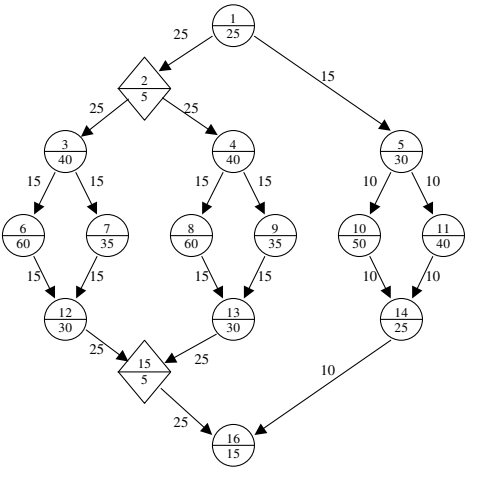
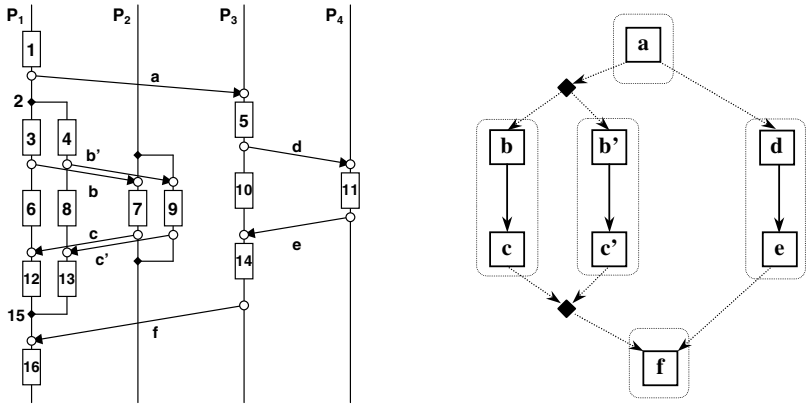


Fig. 2. An example of BTG



(a) expressed in the form of the XAPG graph

(b) Extended Communication Activation Graph partitioned into sections

Fig. 3. A schedule of an exemplary BTG from Fig. 2

is expressed in terms of an Extended Assigned Program Graph (XAPG), see Fig. 3(a). The XAPG is based on the APG representation, introduced in [3]. XAPG assumes the synchronous communication model (CSP–like). There are two kinds of nodes in an XAPG: code nodes (which correspond to tasks in BTG, shown as rectangles in Fig. 3(a)) and communication nodes (circles in Fig. 3(a)). Activation edges are shown as vertical lines in Fig. 3(a), communication edges as horizontal lines. Each processor has activation paths built of activation edges and nodes, which are scheduled on this processor. Activation paths can fork. The conditional fork and join points in the XAPG (solid diamonds in Fig. 3(a)) correspond to the conditional and merge nodes in the BTG, respectively.

The Extended Communication Activation Graph (XCAG) is used in the program graph partitioning into sections. This graph is composed of nodes, which correspond to communication edges of the XAPG program graph, and of edges, which correspond to activation paths between communication edges of the XAPG, Fig. 3(b). The fork and join nodes (shown as solid diamonds in Fig. 3(b)) represent the fork and join points from the XAPG, respectively. Program sections are defined by identification of such subgraphs in the XAPG or XCAG that the following validity conditions hold:

a) Each vertex of the XCAG graph belongs to one and only one section.
b) The edges, which connect nodes contained in a section subgraph define a connected subgraph when considered as undirected.
c) All nodes on each path, which connects two nodes belonging to a section subgraph belong to the same section.
d) Processor link connections inside section subgraphs do not change.
e) Fork nodes don't belong to any section. Nodes incoming to or outgoing from the fork node are always section boundaries.

3 Program Structuring Algorithms

Program structuring algorithms, presented in the paper, exploit control flow properties of BTG to detect mutually exclusive tasks and communications. It allows effective sharing of processors and processor links in the look–ahead reconfigurable environment, and thus, leads to reduction of total program execution time. The presented algorithm uses a two–phase approach to obtain the schedule and partition of a program graph [3,4]. In the first phase, a list scheduling algorithm is applied to obtain a program schedule with reduced number of communications and minimized program execution time. In the second phase, scheduled program graph is partitioned into sections.

The scheduling algorithm is based on the ETF (Earliest Task First) heuristics [1]. Our algorithm introduces new communication model: instead of fixed inter–processor network topology, as in the original ETF, we investigate system with look–ahead dynamically created connections. We take into account a limited number of links, link switches and links contention.

The second improvement consists of the new methodology for conditional branch scheduling. It includes the *detection of mutually–exclusive paths* [6] and scheduling of *most–often–used paths* based on branch probabilities [7].

In our modified ETF heuristics, we use the new formula for evaluation of the earliest starting time of a task ($Ready(n_i, P_i)$ procedure, which returns the time when the last message for task n_i arrives at processor P_i, see [1]), Fig. 4. In this procedure, additional overhead (c_R in Fig. 4) is introduced into total communication time when network topology should be changed and there is no sufficiently long time gap after last communication to do reconfiguration in advance without delaying program execution. This link reconfiguration time overhead is minimized by reduction of the number of link reconfigurations.

In program graph schedules, tasks and communications belonging to subgraphs implied by *mutually exclusive* conditional branch paths can share the same resources. It is accomplished by their overlapping assignments in the same time slot on a target system resource. The algorithm for detection of mutually exclusive nodes and edges is based on the branch labeling method (see [6] for

```
Procedure Ready(n_i, P_i)
Time := 0
For each n_j ∈ Predecessors n_i
    T_Arrive := finishing time of task n_j
    P_j := processor which task n_j is
                              scheduled on
    If P_j ≠ P_i Then
        T_Arrive := T_Arrive + c_j,i
        {cost of communication from T_j to T_i}
        If P_i and P_j are connected Then
            send := link of P_j connected to P_i
            recv := link of P_i connected to P_j
        Else
            send := last recently used link of P_j
            recv := last recently used link of P_i
            If time since last use of link
                L_j,send or link L_i,recv in
                previous configuration < c_R
                Then
                    T_Arrive := T_Arrive + c_R
                EndIf
            EndIf
            Allocate temporarily communication
            e_j,i on links L_j,send and L_i,recv
            (sharing the links with mutual
              exclusive communications, if possible)
        EndIf
        If Time < T_Arrive Then
            Time := T_Arrive
    EndIf
EndFor

Allocate temporarily task n_i on processor P_i
(sharing the time slot with mutual exclusive
tasks, if possible)

Return Time
```

Fig. 4. The *Ready* procedure used in scheduling algorithm

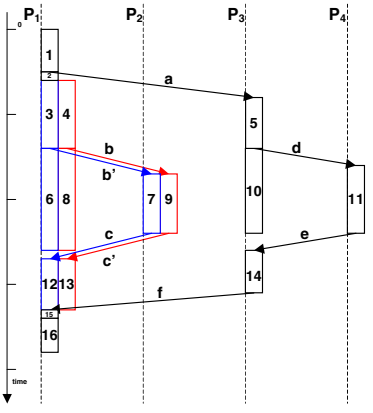


Fig. 5. Gantt chart of a schedule of an exemplary BTG from Fig. 2

details). An exemplary schedule for program from Fig. 2 is shown in Fig. 5. The algorithm detected two exclusive branches (nodes 3, 6, 7, 12 and 4, 8, 9, 13). Communications b, c and b', c' share the same link since their time slots are overlapping and they are scheduled on the same processor link.

To improve schedule quality, the structuring heuristics is based on the most–often–used paths approach [7]. The most–often–used path is a subgraph starting from a conditional node, whose execution probability is above the given threshold p_{MOU}. Nodes belonging to such paths have always the highest priority among ready nodes, including critical path of the graph (paths that are traversed less frequently have little impact on the average program execution time so they can be allowed to execute longer). When there is no most–often–used path in a conditional branch (i.e. paths have uniform or close probabilities), selection of the nodes for scheduling behaves according to the standard ETF paradigm.

The second phase of the program structuring is the graph partitioning algorithm into sections [3, 4]. The goal of this phase is to find program graph partition into sections executed with pre–configured inter–processor connections and to assign a crossbar switch to each section, Fig. 6. At the beginning of the algorithm, each section is built of a single communication, all sections are assigned to the same crossbar switch. In each step, the algorithm selects a vertex of XCAG and it tries to include this vertex to a union of existing sections determined by incoming edges of the current vertex. The heuristics tries to find such a union of sections, which doesn't break rules of graph partitioning. Nodes belonging to subgraphs implied by different conditional paths (or from conditional and unconditional paths) are placed in separate sections, thus the deadlock condition during program execution is avoided. The union, which gives the shortest program execution time is selected or, when there is no execution time improvement, the section of the current vertex is left untouched. The vertices can be visited many times. The algorithm stops when all vertices have been visited and there hasn't been any program execution time improvement in a number of steps.

```
Begin
B := initial set of sections, each section is
composed of single communication and assigned to
crossbar 1
curr_x := 1   {current number of switches used}
finished := false
While not finished
    Repeat until each vertex of XCAG is visited
        and there is no execution time
        improvement during last β steps {1}
      v := vertex of CAG which maximizes the
         selection function and which is not
         placed in tabu list
      S := set of sections that contain
         communications of all predecessors of v
      M := Find_sections_for_clustering(v, S)
      If M ≠ Ø Then
        B := B - M
        Include to B a new section built of v and
          communications that are contained in
          sections in M
      Else
        s := section that consists of
                              communication v
        Assign crossbar switch (from 1..curr_x)
                                    to section s
        If reconfiguration introduces time overheads
        Then
          curr_x := curr_x + 1
          Break Repeat
        EndIf
      EndIf
    EndRepeat
    finished := true
EndWhile
End
```

Fig. 6. The general scheme of the graph partitioning algorithm

Estimation of the program execution time is based on simulated execution of the partitioned graph in a look–ahead reconfigurable system. For this purpose, a XAPG graph with a valid partition is extended by subgraphs, which model the look–ahead reconfiguration control. Weights of the graph nodes correspond to latencies of respective control actions, such as crossbar switch reconfiguration, bus transfer latency, and similar. During simulated execution of a XAPG, we assume that, at the conditional fork points, the most–often–used conditional paths are selected for execution. If there is no most–often–used path (equal or similar path probabilities), all outgoing paths are simulated, and the worst (the longest) execution time is selected. During program execution, the connection reconfiguration subsystem first creates connections for the section, which contains the most–often–used path. The connections for sections from less probable paths are created in descending order of their execution probabilities. For equal or similar path probabilities the reconfiguration selection order is arbitrary.

Taking as an example the graph shown in Fig. 2, we have compared schedules obtained with our algorithm with schedules based on the generally known ETF heuristics adapted by a "brute force" method. To deal with conditional branches in the program meant for execution in the look ahead reconfigurable environment, this ETF method maps conditional program subgraphs in whole to separate processors. Compared with the standard ETF list scheduling algorithm, which uses DAGs as program representation, our structuring algorithm allows for more fine–grain analysis of parallel program and thus leads to better (shorter) program execution times. We expect that this property can be generalized and will hold also for larger programs. Some comparison results of program structuring by both methods are shown in the table below:

algorithm:	"brute force" ETF	proposed in the paper
program representation	DAG	BTG
nb. of communications	4	6
nb. of link reconfigurations	2	3
schedule length	215	185

4 Conclusions

A new kind of the program graph structuring algorithm for the execution in the look–ahead reconfigurable multi–processor system with redundant communication resources has been presented in the paper. It enables task scheduling of programs, which include conditional branches in application programs. The new graph representation of programs captures data and control dependencies between tasks. The presented algorithm is based on improved list scheduling and iterative section clustering heuristics, which have been shown to give better results than the one with greedy vertex selection heuristics and have the same time complexity. The new representation enables applying new branch optimization techniques based on detection of *mutually–exclusive paths* in programs and the heuristics scheduling according the *most–often–used paths* approach based on branch taking probabilities. It allows efficient processor and processor links sharing in the look–ahead reconfigurable environment for execution of exclusive program parts. Thus, reduction of total program execution can be achieved.

References

1. Jing–Jang Hwang et al, *Scheduling Precedence Graphs in Systems with Interprocessor Communication Times*, Siam J. Comput., Vol. 18, No. 2, 1989.
2. M. Tudruj, *Look–Ahead Dynamic Reconfiguration of Link Connections in Multi–Processor Architectures*, Parallel Computing '95, Gent, Sept. 1995, pp. 539-546.
3. E. Laskowski, M. Tudruj, *A Testbed for Parallel Program Execution with Dynamic Look–Ahead Inter–Processor Connections*, Proc. of the PPAM '99, Sept. 1999.
4. E. Laskowski *Program Structuring Algorithms for Dynamically Reconfigurable Parallel Systems Based on Redundant Connection Switches*, Proc. of the 3^{rd} Int. Symposium on Parallel and Distributed Computing, 2004, Cork, Ireland, pp. 248-255.
5. Dong Wu, Bashir Al–Hashimi, Petru Eles *Scheduling and Mapping of Conditional Task Graph for the Synthesis of Low Power Embedded Systems* IEEE Proceedings – Computers and Digital Techniques, Vol. 150, Issue 5, September 2003, pp. 303-312.
6. Yuan Xie, Wayne Wolf *Allocation and scheduling of conditional task graph in hardware/software co–synthesis.* DATE 2001, pp. 620-625.
7. C. Murphy, X. Wang, *Most Often Used Path Scheduling Algorith,* Proc. of the 5th World Multi–Conference on Systemics, Cybernetics and Informatics (SCI 2001), July 22-25, 2001, Orlando, USA. Vol. XII, pp. 289-295.

Scheduling Task Graphs for Execution in Dynamic SMP Clusters with Bounded Number of Resources

Lukasz Masko

Institute of Computer Science of the Polish Academy of Sciences,
ul. Ordona 21, 01–237 Warsaw, Poland
masko@ipipan.waw.pl

Abstract. The paper presents an algorithm for scheduling parallel tasks in a parallel architecture based on multiple dynamic SMP clusters, in which processors can be switched between shared memory modules at runtime. Memory modules and processors are organized in computational System–on–Chip (SoC) modules of a fixed size and are inter–connected by a local communication network implemented in a Network–on–Chip technology (NoC). Processors located in the same SoC module can communicate using data transfers on the fly. A number of such SoC modules can be connected using a global interconnection network to form a larger infrastructure. The presented algorithm schedules initial macro dataflow program graphs for such an architecture with a given number of SoC modules, assuming a fixed size of a module. First, it distributes program graph nodes between processors. Then it transforms and schedules computations and communication to use processor switching and read on the fly facilities. Finally, it divides the whole set of processors into subsets of a given size, which then are mapped to separate SoC modules.

1 Introduction

The paper deals with the problem of scheduling program graphs in a system with dynamic processor switching and data transfers on the fly. The architecture of such system is based on SMP clusters, which connect processors to shared memory modules. Processors and memory modules are grouped in blocks and implemented as "system–on–chip" modules [5]. A system is built of a number of such modules, connected via a global network connecting all processors with all memory modules. Every SoC module includes a number of processors and a number of shared memory modules. Every memory module has its own, dedicated memory bus. At a time, every processor in a SoC can be connected to some of these memory modules. A set of processors connected to the same shared memory module constitutes a cluster. Inside SoC modules, processors can be dynamically switched between clusters at runtime, enabling dynamic distribution of computations and communication between processor clusters. The collective communication capabilities of an intra–cluster communication network allow to use data reads on the fly for efficient intra–cluster data transfers. This technique

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 871–878, 2006.

consists in parallel reads of data which are written or read to/from a memory module through an internal data exchange network, by many processors in this cluster [3, 4].

In general, program scheduling for parallel systems is NP–hard. In the presented architecture it must take into account such problems as cache–driven macro dataflow execution principle, which imposes constraints on processor data caches, as well as multi–level communication methods (transferring data in processors' data caches between nodes, standard and on the fly data transfers via local memory buses and communication through a global interconnection network). Some scheduling algorithms for unbounded number of resources have already been presented [6, 7]. The paper presents an algorithm, which takes into account both the number of SoC modules in a system and the number of processors in a single SoC module. The algorithm works in three phases. First, all computing nodes of the program graph are distributed between all available processors in the system, regardless their placement in actual SoC modules. Then, the graph is structured to convert all possible data transfers to reads on the fly. Finally, the whole set of processors, together with program graph nodes mapped to them, is distributed between all available SoC modules.

The paper is composed of 3 parts. In the first part, the assumed system architecture is outlined. The second part presents an extended macro dataflow program graph notation. The third part describes the scheduling algorithm for the assumed architecture and its application to exemplary program graph of a numerical algorithm.

2 SMP Architecture with Processor Switching and Data Transfers on the Fly

The general structure of the assumed system is presented in Fig.1a. The basic system module consists of a number of processors (P) and shared memory modules (M) connected via a local interconnection network (Local Network), implemented as a system–on–chip (SoC) module. A number of such modules is connected via global interconnection network (Global Network), which allows every processor to read data from any memory module in the system.

The structure of a single SoC module is presented in Fig.1b. A SoC module consists of a number of processors and shared memory modules. Every memory module has its own, dedicated local network. Every processor has a multi–ported data cache memory. It can be connected to many memory modules at a time via their local networks. A set of processors connected to the same local memory network constitutes a cluster. A processor can be dynamically switched between clusters on demand. This architecture is based on the idea presented in [3, 4] with assumption, that a single module has a limited number of processors and memory modules. This is derived from the fact, that memory bus, which is assumed here as a local network, has a limited throughput.

A local memory bus allows multiple, simultaneous reads of the same data by many processors connected to this bus. This feature is used for a very efficient

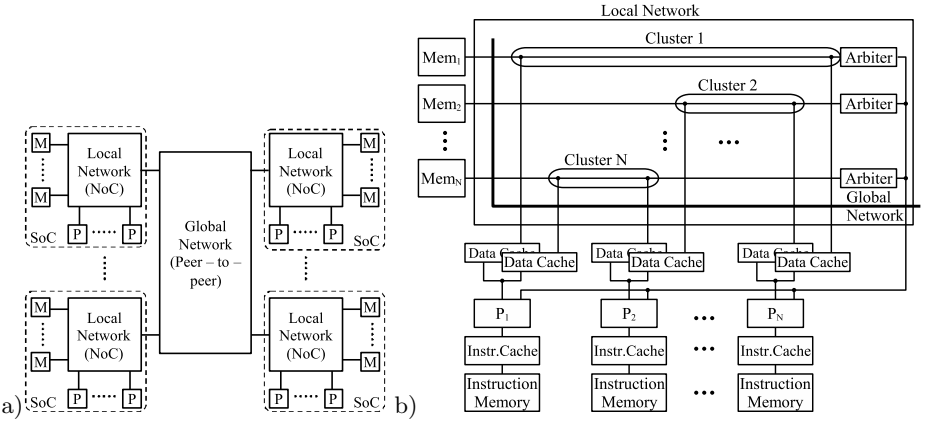


Fig. 1. General system structure (a) and architecture of a single SoC module (b)

intra–cluster communication method called read on the fly. It consists in parallel reads of data, which are being read from or written to the shared memory module by many processors, which need this data for their computations.

Each local memory bus is supervised by its arbiter, which manages all data transfer requests and processor switching operations. Each processor is equipped with a Bus Request Controller (BRC), which is connected to all memory bus arbiters. It manages local queues of processor's memory access requests and performs data transfers on the fly.

In order to avoid uncontrolled data cache reloads during program execution we have introduced a new program execution principle called cache–driven macro dataflow model. According to it, each block of a program (corresponding to computation node in the program graph) is constructed in such a way, that it can be executed without data transfers to or from shared memory, using only the information stored in processor's data cache. Therefore, before a block can be executed, all data required for it must be read from shared memory. After the block is completed, all the results required for other processors must be transferred back to the shared memory module.

3 Extended Macro Dataflow Graph Program Notation

The presented algorithm uses a program graph structure based on a standard macro dataflow graph program notation. In order to model additional operations (processor switching, synchronization, data reads on the fly), the graph is expanded by new nodes end edges. An exemplary extended macro dataflow program graph (EMDG) is shown in Fig. 2a. Basic nodes are the computation (rectangles) and communication nodes (circles). A computation node represents a sequence of computation instructions in the program. According to cache–controlled execution principle, it is performed without any data transfers from or to data cache during execution. All data required for this operation are read

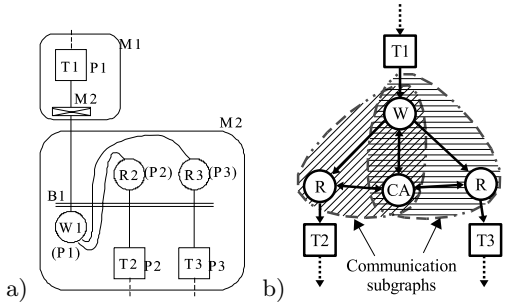


Fig. 2. Nodes of the program graph (a) and communication subgraphs (b)

before execution of the node and held in processor's data cache. The results required for other nodes are written back to the shared memory module after the node execution is finished. Basic communication nodes are write and read nodes. They represent data transfers from processor's data cache to a memory module or from a shared memory module to a data cache respectively. Such an operation consists in placing a proper request in processor's BRC and waiting for a data transfer completion. In the graph, these operations are denoted by circles. All the nodes are marked with labels of processors, which execute them.

Processor switching is denoted as a crossed rectangle with a label of the memory module, to which a processor is switched (if a processor can be simultaneously connected to many local memory buses, the label consists of the pair — from which bus to disconnect and to which to connect a processor). Processor switching includes a communication between a given processor and bus arbiters. First communication aims in disconnecting a processor from a bus). Second one is for connecting the processor to a new bus (by a proper arbiter). Switching can take place only, when there is no traffic on an appropriate bus. Synchronization (barrier operation) is denoted by a parallel line crossing some (at least two) of the paths in the graph. It is also marked with barrier's label (i.e. B1). Rounded polygons marked with memory module's label mean a set of nodes executed using the same shared memory module, i.e. in the same processor cluster.

In order to model a read on–the–fly operation, read nodes should have different meaning than the standard ones described above. In the situation, like shown on Fig.2a, they should be considered as just placing requests in processors' BRCs with the read from the memory bus performed after a barrier, executed by a processor. Curved edges mean actual data transmission. While data are written from the writing processor's (P1) data cache with instruction W1, processors P2 and P3 read the data to their data caches using read on–the–fly technique based on bus snooping. If such data transmission is preceded by processor switching of the writing processor, communication on–the–fly is performed.

In a standard macro dataflow program graph, an edge corresponds to communication, composed of both data send and receive operations. In a shared memory system, both writing data from processor's data cache to a shared memory module and reading them back to data cache of another processor, constitute separate

operations. They may include additional synchronization and interaction with bus arbiters. Therefore, it is necessary to introduce to an EMDG an equivalent of the "communication edge". It is called "communication subgraph", which is a subgraph containing a read node, a write node which precedes this read in the graph and nodes of arbiters, which control both transmissions (together with all additional edges and nodes between them, see Fig.2b).

4 Scheduling Algorithm

The presented algorithm aims at finding a time–optimal schedule of the input program graph in the assumed architecture. It includes three following steps:

- First, nodes of the initial program graph are mapped to processors in a system. This step assumes a simplified architecture model, in which all the processors are inter–connected with a fully connected network and data transfers are performed using simple point–to–point communications. A standard macro dataflow program graph notation is used.
- The second step transforms communication in such a way, that it is performed using data transfers on the fly. It also assumes such constraints as the number of local memory buses, to which a processor may be simultaneously connected. In this step, all the processors are still inter–connected by one large interconnection network, which is now a bus–based network with local and global memory buses connecting processors with shared memory modules. Here, the extended macro dataflow program graph notation is used.
- In the third step, the structured program graph obtained in previous steps is mapped to a target system, in which the whole set of processors is divided into subsets and each such subset is mapped to a separate SoC module.

4.1 Distribution of Nodes of a Program Between Processors

At the beginning, all the computation nodes of the initial program graph are distributed among all the processors in the system. In this step, a standard macro dataflow notation of the program is used, in which program graph nodes correspond to computations and edges correspond to communications. It is assumed that all N processors communicate with each other via a fully connected network. If two adjacent computing nodes are mapped to the same processor, communication between them has cost 0. Otherwise, the cost is equal to the weight of an edge between these nodes. This is similar to a standard clustering technique [2] and corresponds to transformation of data transmissions via shared memory modules to transfers of data in processors' data caches. The nodes–to–processors distribution is performed using a list scheduling algorithm with ETF heuristics [1].

4.2 Structuring a Mapped Program Graph

In this step a scheduled program graph is structured to convert as much communication between processors as possible into data transfers on the fly. This

process is based on atomic subgraphs and their basic transformations, as described in [6]. The algorithm consists in traversing a program graph, selecting basic structures in it and transforming them into equivalent forms, in which all possible data transfers are performed on the fly. The traversal of the program graph is performed using heuristics based on a Dominant Sequence. A basic structure is a local subgraph corresponding to one of the following communication schemes: one to many communication scheme (broadcast), many to one communication scheme (reduction), many to many communication scheme (butterfly).

For a given program graph, the presented algorithm calculates a dominant sequence using simulated execution of a graph. Then, it selects the heaviest unexamined communication subgraph on this path. Next, a basic structure is selected, which contains this subgraph. Finally, this structure is a subject to proper transformation. As a result, an equivalent program graph is obtained, in which the most important (the heaviest) communication is transformed from a communication via shared memory to data transfers on the fly. Transformations try to convert standard data transfers into reads and communication on the fly by introducing barriers, converting standard read nodes to proper read on the fly nodes and adding processor switchings, if required [6].

4.3 Distribution of Processors Between SoC Nodules

The aim of the third part of the algorithm is to distribute a set of processors between SoC modules, together with tasks scheduled on them. After the first and the second part of the algorithm, it is assumed that all processors can be connected to any local memory bus in the system via a local network, which is hardly possible to be achieved due to technical limitations. Therefore, all the processors are divided into subsets of the size not greater than the size of a SoC module. Then, each such subset is mapped to a separate SoC module. Before this distribution, almost all communication was performed using local memory buses. After distribution, a part of communication must be executed using a global interconnection network. One of the goals of the third step is to minimize the cost of such communications.

This part of the algorithm on based on a genetic approach. Each chromosome in a population corresponds to one distribution scheme. A chromosome is a sequence of N integers $(a_1 \dots a_N)$ (N is the number of processors), where $a_i \in (1 \dots M)$ (M being the number of available SoC modules). Each a_i determines a placement of processor i in the SoC module a_i. It also determines the placement of the shared memory module i in the same SoC module. Because the size of a SoC module is fixed and such module may contain not more than K processors and K memory modules ($N = M \times K$), for every module m, the number of indexes i such that $a_i = m$ is not greater then K. According to such placement, for every pair of processors i and j, if $a_i \neq a_j$ all communication between these processors is performed through the global interconnection network. If $a_i = a_j$, then communication between these processors use local communication networks, which implies, that they can be executed using data transfers

on the fly. For a given chromosome, the structured graph determined by this chromosome, is described as follows:

- Each processor i is mapped to the module a_i, together with all program graph nodes mapped to it in step 1 of the algorithm, with structuring determined in step 2.
- Each communication between processors i and j is executed in a way determined in step 2 of the algorithm, if $a_i = a_j$. For $a_i \neq a_j$, communications are converted to use the global interconnection network and are executed in a standard way.

The fitness function. The value of a fitness function Fit for a chromosome C is determined by the following equation:

$$Fit(C) = const - T(C)$$

where $T(C)$ is the execution time of a structuring determined by the chromosome C and $const$ is a constant, such that $Fit(C) > 0$ for every possible C (it can be for instance the product of the number of all processors N and the sum of weights of all the program graph nodes).

The crossover operator. The crossover operator for two chromosomes $A = (a_1 \dots a_N)$ and $B = (b_1 \dots b_N)$ is defined as follows:

- The index $1 < n < N$ is chosen at random to determine the position, at which the initial chromosomes will be divided.
- From the initial chromosomes A and B two new are obtained, according to the following scheme:

$$X' = (x'_1 \dots x'_N) = (a_1 \dots a_n b_{n+1} \dots b_N)$$
$$Y' = (y'_1 \dots y'_N) = (b_1 \dots b_n a_{n+1} \dots a_N)$$

- Finally, the chromosomes X' and Y' must be repaired to obey the rule, that the number of processors mapped to each of the SoC modules is not greater than K. It is obtained by distribution of extra processors (chosen at random from all mapped to the considered module) among other SoC modules, which are not full yet. This selection is done according to the "Best Fit" rule and tries to place as many processors as possible in the same module.

As the result, two new chromosomes X and Y are obtained, which define the new mappings of processors to SoC modules.

The mutation operator. The mutation operator consists in exchange of mapping of two processors randomly chosen from two different SoC modules. Such transformation obeys constraints on the number of processors in SoC modules, so no further chromosome reparation is required.

5 Conclusions

The paper presents an algorithm for scheduling programs given as macro dataflow graphs in parallel architecture based on dynamic SMP clusters with communication on the fly. The algorithm schedules the initial program graph taking into account constraints on the number of SoC modules in the target system, as well as the number of processors inside a single SoC module. The scheduling algorithm has three phases. In the first one, nodes of a program graph are distributed among all available processors. Then, the mapped program graph is structured to transform standard communication between processors using shared memory to reads on the fly. Finally, the processors are distributed among available SoC modules. As a result, an extended macro dataflow program graph is obtained, which is transformed and scheduled for the target system. The algorithm schedules program graphs to a system with a bounded number of both SoC modules and processors. It reduces global communication by converting it into reads on the fly and data transfers in processors' data caches. Further works include introduction of additional constraints on the size of processor's data cache.

References

1. J.–J. Hwang, Y.–C. Chow, F. D. Anger, C.–Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, SIAM Journal on Computing Vol. 18, No. 2, Society for Industrial and Applied Mathematics, 1989.
2. T. Yang, A. Gerasoulis, DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 9, 1994.
3. M. Tudruj, L. Masko, Program execution control for communication on the fly in dynamic shared memory processor clusters, Proceedings of the 3rd International Conference on Parallel Computing in Electrical Engineering, PARELEC 2002, September 2002, Warsaw, IEEE Computer Society Press.
4. M. Tudruj, L. Masko, Communication on the Fly and Program Execution Control in a System of Dynamically Configurable SMP Clusters, 11^{th} Euromicro Conference on Parallel Distributed and Network based Processing, February, 2003, Genova – Italy, IEEE CS Press, pp. 67–74.
5. Ch. Rowen, Engineering the Complex SOC, Fast, Flexible Design with Configurable Processors, Prentice Hall PTR, 2004.
6. L. Masko, Atomic operations for task scheduling for systems based on communication on–the–fly between SMP clusters, 2^{nd} International Symposium on Parallel and Distributed Computing ISPDC, Ljubljana, October 2003, IEEE Computer Society Press.
7. L. Masko, Program graph scheduling for dynamic SMP clusters with communication on the fly, International Symposium on Parallel and Distributed Computing ISPDC 2004, Cork, 5-7 July 2004, IEEE Computer Society Press.

Scheduling Moldable Tasks for Dynamic SMP Clusters in SoC Technology

Lukasz Masko[1], Pierre–Francois Dutot[2], Gregory Mounie[2], Denis Trystram[2], and Marek Tudruj[1,3]

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

Abstract. The paper presents an algorithm for scheduling parallel programs for execution in a parallel architecture based on dynamic SMP processor clusters with data transfers on the fly. The algorithm is based on the concept of moldable computational tasks. First, an initial program graph is decomposed into sub–graphs, which are then treated as moldable tasks. So identified moldable tasks are then scheduled using an algorithm with warranted schedule length.

1 Introduction

The paper concerns task scheduling in parallel programs based on the notion of moldable computational tasks. A moldable task (MT) is a computational task, which can be executed using an arbitrary number of parallel processors. Such tasks have been used as atomic elements in program scheduling algorithms with warranty of schedule length [1, 2]. Parallel execution cost of MTs for variable number of executing processors is characterized by its penalty function, which determines task execution efficiency versus an ideal parallel execution on the given number of processors. For program execution, a special shared memory system architecture is used. It is based on dynamic processor clusters, organized around shared memory modules. Data reads on the fly are provided in these clusters which means that processors in a cluster can read data, which are written or read to/from a memory module through an internal data exchange network. Processors can be switched between clusters with data in their data caches. After switching, a processor can write data from its cache to the memory, allowing the data to be read on the fly by other processors in the cluster. Dynamic shared memory (SMP) clusters are organized inside integrated system on chip (SoC) modules [8] which are connected by a central global interconnection network. The global network enables direct global communication between all processors in the system and all memory modules. The presented algorithm aims at minimization of the program execution time. It assigns tasks to processors

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 879–887, 2006.

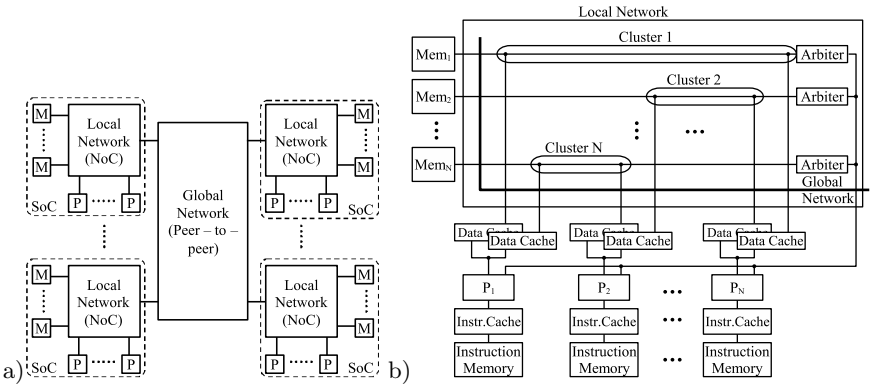


Fig. 1. General system structure (a) and the structure of a SoC module (b)

and inter–task shared memory communication to dynamic processor clusters. It converts standard communication into reads on the fly. Communication is analyzed at the level of data exchanges between processor data caches.

The paper is composed of two parts. First, the assumed system architecture is outlined. Then the algorithm for scheduling an application program graph in SoC modules containing dynamic SMP clusters using MT approach is described.

2 Dynamic SMP Clusters and Communication on the Fly

General structure of the assumed SMP system is presented in Fig.1a. A system consists of multiprocessor SMP modules containing processors (P) and shared memory modules (M) connected via local interconnection networks (Local Network), implemented as system on chip integrated modules (SoC). A number of such modules is connected via a global interconnection (Global Network).

The structure of a single SoC module is presented in Fig.1b. A module consists of a number of processors and shared memory modules. Every memory module has its own, dedicated local network. This architecture is based on the idea presented in [5] with assumption, that a single module has a limited number of processors and memory modules. This constraint is derived from limited communication efficiency of a memory bus for a bigger number of processors. Every processor has a multi–ported data cache memory. It can be connected to many memory modules at a time via local networks. A set of processors connected to the same local network (memory bus) constitutes a cluster. A processor can be dynamically switched between clusters on demand. Inside a cluster, a very efficient intra–cluster communication method called reads on the fly is provided. It consists in parallel reads of data by many processors connected to the same memory bus, while the data are read from or written to a shared memory module. All data transfers as well as processor switching between busses are supervised by memory bus arbiters. On processors' side, data transfers are organized by processors' Bus Request Controllers (BRC). The BRCs communicate with all

memory bus arbiters, supervise local queues of processor's requests and organize data transfers between data caches and memory modules. This includes management of processor switching and data reads on the fly. The details of the assumed cluster architecture are given in [5, 6].

For adequate modeling of program execution in the assumed architecture, an Extended Macro Dataflow Program Graph notation (EMDG) was introduced [5]. It extends a standard macro dataflow program graph with new types of nodes and edges, which allow modeling of such operations as processor switching, reads on the fly and synchronization.

3 The Scheduling Algorithm

We are given a program macro data–flow graph and an executive system of interconnected dynamic SMP clusters, with N processors and M memory modules. We want to determine the time–optimal schedule of the graph in the assumed SMP system. The proposed scheduling algorithm consists of the following steps:

1. Definition of MTs inside the program graph. This part is based on a standard Dominant Sequence Clustering (DSC) algorithm [3]. The basic assumption is that the nodes which correspond to the same MT are mapped to the same SoC module. The size of such task depends on the size of a SoC module. All defined MTs also fulfill certain conditions on their external communication.
2. Determining the penalty function for each moldable task. This step consists in scheduling the obtained MTs for a range of available resources. For each number of available processors the best scheduling of each MT is found.
3. Assignment of resources (allotment) to each MT and their scheduling. In this step both processors and memory buses are assigned. So obtained tasks are then scheduled using a modified list scheduling algorithm assuming that all processors which are assigned to a task belong to the same SoC.

3.1 Step 1 – Definition of Moldable Tasks

In this step, the initial program graph is divided into subgraphs corresponding to moldable tasks. Every such task can be characterized by the following rules:

1. The smallest task subgraphs consist of only one computation node.
2. All external reads can be performed only at the beginning of the task: for any node v from the considered MT, if the task subgraph includes any predecessor u of the node v, it must include all predecessors of v.
3. All external writes can be performed only at the end of the task: for any node u from the considered MT, if the task includes any successor v of the node u, it must include all successors of u.
4. A single MT is executed entirely inside a single SoC module, which means, that global communications can exist only between separate MTs.

The algorithm is based on task clustering. In the initial structuring, all the nodes from the macro dataflow program graph constitute separate MTs. Together with precedence constraints imposed by initial program graph, they form an initial moldable task graph (MT graph is a graph, in which nodes correspond to MTs, while edges denote precedence dependencies between them). It is assumed, that all data transfers between separate MTs are executed via global communication network. The algorithm creates new larger MTs by merging smaller tasks. The reduction in execution time is obtained by transforming global communication between separate small tasks into local communication performed inside a larger task. Such local communication may be executed on the fly, which reduces its execution time.

Everytime a set of MTs is merged to create a larger MT, all the nodes from this set are substituted by one single "meta–node" representing the new task. This node must remember its internal structure, which is used in case of further mergings. The weight of this meta–node is equal to optimal execution time of the subgraph which it represents. In order to obtain MTs, which obey rules 2 and 3 described above, a new "merging" operation of MTs must be defined. The proposed algorithm which merges two MTs is presented as Algorithm 1.

In a standard DSC algorithm, merging of two parallel clusters and mapping them onto the same processor reduces communication between the clusters (edge weights become 0), while all computation nodes have to be performed sequentially. In the presented algorithm, mapping parallel subgraphs onto the same SoC module means only, that these subgraphs are executed using the same set of processors. If this set is large enough, these tasks can be still executed in parallel. This introduces a new problem – how to decide, whether two parallel

Algorithm 1

Input data: MTs graph G and its two nodes u and v. There exists an edge from u to v in G. Assign boolean attributes d and p to every node of G and mark both of them as not checked.
Create two empty sets D and P. Insert u into D and mark it as visited.
$T = \emptyset$ is a subgraph, which will contain the result of merging of u and v.
while any of sets D and P is not empty **do**
  for each element t of D **do**
    add t to the task subgraph T, mark its d attribute as checked.
    insert all unvisited descendants of t, which have their p attribute unchecked, into set P and remove t from D.
  end for
  for each element t of set P **do**
    add t to the task subgraph T, mark its p attribute as checked.
    insert all unvisited predecessors of t, which have their d attribute not checked, into set D and remove t from P.
  end for
end while
Subgraph T defines a new MT, which merges initial tasks u and v. Find execution time of T by symbolic evaluation of a subgraph, which corresponds to it.

subgraphs should be mapped onto the same SoC module. The decision depends on the graph as well as on the number of resources available inside the SoC module (i.e. processors and memory modules). A set of MTs can be merged to create a new MT only if a resulting task can be scheduled on a single SoC module in an efficient way, therefore the size of a single MT must be limited, depending on both computations and communication. The constraints on a number of required processors can be easily checked using symbolic execution of a macro dataflow graph of a task assuming, that all communication is performed on the fly. In order to check communication constraints, a Communication Activation Graph (CAG, see Fig.2c) is defined in the following way:

- All the communication inside a task is performed using reads on the fly. Only initial data reads from the shared memory and final writes to the shared memory are executed in a standard way.
- The nodes in a CAG correspond to communications (subgraphs of the transfers on the fly or standard communication edges). The weight of each node is equal to execution time of its corresponding subgraph (or a single communication in case of standard data transfers).
- Edges in a CAG correspond to computation paths between communications in EMDG. Their weights are equal to path execution times.

For a given task T, two following functions are defined:

• $F^T_{comp}(t)$ — this function describes the number of potentially concurrent computation nodes, which are executed at time point t. It is determined by projection of execution times of computation nodes of task T on a time axis.
• $F^T_{comm}(t)$ — this function describes the number of concurrent communications on the fly, which are executed at time point t. It is determined by projection of execution times of nodes in a CAG of task T.

The execution time of a CAG can be determined in parallel with execution time of nodes and edges in a standard macro dataflow program graph. The

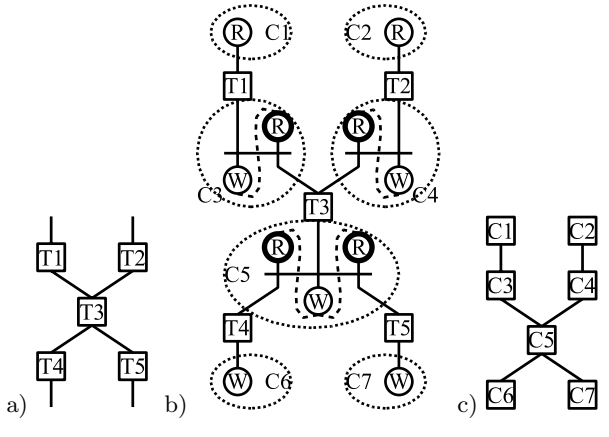


Fig. 2. Example of a task graph (a), its EMDG (b) CAG (c)

algorithm assumes, that, for any task T, at any moment of its execution, these functions fulfill the following constraints:

$$F^T_{comp}(t) \leq N, \quad F^T_{comm}(t) \leq M$$

where N is a number of processors and M is a number of shared memory modules (memory busses) in a SoC module (in the paper it is assumed, that $N = M$). This implies, that all potentially parallel computations and communications of task T will have enough resources to be executed in parallel.

The first part of the scheduling algorithm is presented as Algorithm 2. After it is executed on the initial program graph, the set of computation nodes is divided into subsets. These subsets will constitute Moldable Tasks for further steps of the presented scheduling algorithm.

3.2 Step 2 – Determining the Penalty Function for Defined MTs

The goal of this step is to determine the penalty function for each MT created in the previous step. For every MT, this operation consists in finding the best schedule of this task for a range of processors $n \in (1 \ldots N)$, where N is the number of processors in a SoC module. For a given pair (T, n) (T is an MT, n is a considered number of processors), the 2–step algorithm is used, which first maps all computation nodes of this task to n processors inside a SoC module, and then finds a structuring of so mapped nodes, which gives the best execution time. It assumes, that all initial communication coming into a considered MT is ready and may be executed. It also assumes, that all final data transfers to shared memory modules can be executed as soon as the data are ready.

Distribution of program nodes between processors. In the first phase of step 2, nodes of a program graph of the considered moldable task T are

Algorithm 2

Define the Moldable Task graph G from the initial macro dataflow program graph.
Mark all edges in G as unvisited.
while there exists an unvisited edge in G **do**
  Find a critical path of the graph using its symbolic execution.
  Select the heaviest unvisited edge from the critical path. If there is no such edge, select the heaviest edge from the other unvisited edges in G. The selected edge defines two nodes u and v from G, which it connects.
  Find a new MT T by "merging" nodes u and v using Algorithm 1.
  Determine communication activation graph from the EMDG of T. Determine functions $F^T_{comp}(t)$ and $F^T_{comm}(t)$.
  if for all t, $F^T_{comp}(t) \leq N$ and $F^T_{comm}(t) \leq N$ **then**
    Create a new MT graph G' from G, in which all the nodes corresponding to task T are substituted by node of task T. Assign $G := G'$
  else
    Reject task T. Mark the selected edge as visited.
  end if
end while

mapped onto a limited number of processors. This step assumes a standard macro dataflow program graph notation. It also assumes a simple architectural model, in which all processors are interconnected by a fully connected network and data transfers are performed using point–to–point communication. If two adjacent computing nodes are mapped to the same processor, communication between them has cost 0. Otherwise, it is equal to the weight of an edge between these nodes. This is similar to a standard clustering technique [3] and corresponds to transformation of data transmission via shared memory modules into transfers of data through processors' data caches. The distribution is performed using a list scheduling algorithm with ETF heuristics [4].

Structuring a mapped program graph. In the second phase of step 2, a scheduled task program graph is structured to convert as many communications between processors as possible into data transfers on the fly. This step is based on atomic subgraphs and their basic transformations, as described in [7].

The algorithm consists in traversing a program graph, selecting basic subgraphs in it and transforming them into equivalent forms, in which all possible data transfers are performed on the fly. The traversal of the program graph is performed using a heuristic based on an Extended Dominant Sequence, which consists not only of computation and communication nodes, but it also includes interaction of processors with arbiters (both data transfers and processor switching between memory modules) and synchronization. Communication subgraph in an EMDG is an equivalent of a communication edge in a standard macro dataflow program graph. It includes a read node, a write node which precedes this read in the graph and nodes of arbiters, which control both transmissions (together with all edges connecting these nodes).

The algorithm consists of the following steps executed in a loop:

- An extended dominant sequence is determined in a considered MT using simulated execution of its EMDG. Then, the heaviest unexamined communication subgraph on the dominant sequence is chosen.
- Next, a basic structure is selected, which contains this subgraph. A basic structure is a local EMDG subgraph corresponding to communication patterns such as: broadcast, reduction and many to many communication.
- Finally, the selected subgraph is subject to proper transformation. As a result, an equivalent program graph is obtained, in which the heaviest communication is transformed from a transfer via shared memory into a data transfer on the fly. The transformations try to convert standard data transfers into reads and communications on the fly by introducing barriers, converting standard read nodes to proper read on the fly nodes and adding processor switchings, if required.

The algorithm terminates when there is no unexamined communication subgraph on the dominant sequence of the scheduled graph.

3.3 Step 3 – Resources Allotment for MTs and Final Scheduling

In this step, every MT obtained in previous steps is assigned a number of resources which will be used for its execution. It can be performed with the algorithm described in [2]. In this algorithm, allotment is found using dynamic programming in polynomial time. Definition of MTs from previous steps assures, that every task will be assigned to no more processors than the size of a SoC module. As a result, a MT graph is transformed into a standard task graph with each node structured for execution on a fixed number of processors. To schedule it, a modified list scheduling algorithm is used [1, 2].

After the MT graph has been structured, every computation node is assigned to a processor and all communication is assigned to intra–cluster (communication between MTs mapped to the same SoC module) or inter–cluster communication networks (communication between MT mapped to different SoC modules). Finally, communication between consecutive MTs mapped to the same SoC module should be converted back to data transfers on the fly.

4 Conclusions

The paper has presented an algorithm for scheduling parallel programs given in a form of task graphs for a parallel architecture based on dynamic SMP processor clusters with data transfers on the fly. The algorithm uses the concept of moldable tasks. It decomposes an initial program graph into sub–graphs, which correspond to optimally defined moldable tasks. It is done according to communication layout in the program graph, trying to reduce global communication between SoCs. So defined MTs are scheduled using an algorithm with warranty of schedule length.

References

1. C. Jansen, H. Zhang, Scheduling Malleable Tasks with Precedence Constraints, Proceedings of the 17th annual ACM symposium on Parallelism in algorithms and architectures SPAA'05, Las Vegas, Nevada, USA, ACM Press, 2005.
2. R. Lepere, D. Trystram, G. J. Woeginger, Approximation algorithms for scheduling malleable tasks under precedence constraints, 9th Annual European Symposium on Algorithms, LNCS, Springer Verlag, 2001.
3. T. Yang, A. Gerasoulis, DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 9, 1994.
4. J.–J. Hwang, Y.–C. Chow, F. D. Anger, C.–Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, SIAM Journal on Computing Vol. 18, No. 2, Society for Industrial and Applied Mathematics, 1989.
5. M. Tudruj, L. Masko, Communication on the Fly and Program Execution Control in a System of Dynamically Configurable SMP Clusters, 11^{th} Euromicro Conference on Parallel Distributed and Network based Processing, February, 2003, Genova, Italy, IEEE CS Press.

6. M. Tudruj, L. Masko, Dynamic SMP Clusters with Communication on the Fly in NoC Technology for Very Fine Grain Computations, International Symposium on Parallel and Distributed Computing ISPDC 2004, Cork, Ireland, July 2004, IEEE CS Press
7. L. Masko, Atomic operations for task scheduling for systems based on communication on–the–fly between SMP clusters, 2^{nd} ISPDC, Ljubljana, October 2003, IEEE CS Press.
8. Ch. Rowen, Engineering the Complex SOC, Fast, Flexible Design with Configurable Processors, Prentice Hall PTR, 2004.

Model Driven Scheduling Framework for Multiprocessor SoC Design

Ashish Meena and Pierre Boulet

Laboratoire d'Informatique Fondamentale de Lille,
Cité scientifique, 59 655 Villeneuve d'Ascq, Cedex, France
Ashish.Meena@lifl.fr, Pierre.Boulet@lifl.fr

Abstract. The evolution of technologies is enabling the integration of complex platforms in a single chip, called a System-on-Chip (SoC). Modern SoCs may include several CPU subsystems to execute software and sophisticated interconnect in addition to specific hardware subsystems.

Designing such mixed hardware and software systems requires new methodologies and tools or to enhance old tools. These design tools must be able to satisfy many relative trade-offs (real-time, performance, low power consumption, time to market, re-usability, cost, area, etc).

It is recognized that the decisions taken for *scheduling and mapping* at a high level of abstraction have a major impact on the global design flow. They can help in satisfying different trade-offs before proceeding to lower level refinements.

To provide good potential to scheduling and mapping decisions we propose in this paper a *static scheduling framework for MpSoC design*. We will show why it is necessary to and how to integrate different scheduling techniques in such a framework in order to compare and to combine them. This framework is integrated in a model driven approach in order to keep it open and extensible.

1 Introduction

As compared to previous small embedded controllers designing, the designing complexity of today's embedded system (SoC, MpSoC, etc) has grown exponentially. This complexity rise has been seen due to complex integration of multiple resources (DSP, GPP, FPGA, etc) on a single die, the miniaturization of electronic and mechanical devices, the changes in the approaches to design as determined by the cognitive complexity of the task, the massive use of test automation tools, and the pervasive use of embedded system in all kinds of applications.

Real-time systems, beside high performance computing, are about strict timing constraints (deadline, periodicity), application-specific design, and high reliability (fault tolerance). The current trend of using multiple processor (SoC, MpSoC etc) has been promoted to satisfying the real-time systems needs. This trend has made real-time systems a special case of chip level high performance distributed computing. Using multiple processors is an attractive solution because of its price-performance ratio, locality constrains (data processing must

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 888–895, 2006.

take place close to sensors or actuators), reliability (replicating of computing resources provides fault-tolerance), high throughput (parallel task execution) and high schedulability (many scheduling possibilities).

Scheduling and mapping optimization is at the core of the design process of such systems. It is recognized that scheduling and mapping at the highest abstraction level has the most important optimization potential compared to low-level optimizations. Furthermore, as time-to-market constraints become tighter and tighter, one should take these decisions as earlier as possible in the design flow in order to start software and hardware development concurrently and the earliest. These decisions should be definitive as a late modification would induce a too long re-engineering process that would cause to miss time-to-market constraints.

Theses scheduling and mapping optimizations have to take into account many factors such as real-time constraints, power consumption, various costs (area, fabrication, verification, etc), re-usability, adaptability, etc.

As stated scheduling and mapping optimizations are one of the key factors of the whole co-designing process. Keeping all the above mentioned problems in mind we propose here a framework for scheduling and mapping optimization. This framework will be used and integrated within our project Gaspard2 [11], a model driven co-design environment for computation intensive embedded systems. As we focus on intensive applications, we limit ourselves to static scheduling. Dynamic scheduling is another class of problems that we do not handle for the moment.

In the past many attempts has been made to devise scheduling and mapping frameworks or tools for real-time embedded system. As stated about the current and future trends of embedded systems, among them many of the tools were not designed to cope with the rising complexities. For instance, tools such as [2, 15, 1, 16, 13, 12] etc,implement a collection of common scheduling algorithms and protocols for communication and resource access. Among them tools like [1, 16, 13] provides a GUI with their own modeling language and one has used UML base modeling [15]. Here some have focused on desired behavior of the system rather than implementation choices, such as which scheduling policy to use. Further, most of the existing frameworks are not considering multi-criteria constrains, flexibility and scalability to apply different newly or existing scheduling and mapping techniques [2, 17, 8, 14, 12] etc.

In contrast to tools or framework listed above, Our framework in addition, is targeted to certain required futuristic features, which makes it unique to the best of our knowledge. It applying scheduling and mapping algorithms on a global task graph with three approaches first simultaneously, second iteratively, third with user concerns. It evaluates, compare and merges their relative results while respecting different constraints (multi-objective).

It will provide the flexibility of plug-ins and interoperability with legacy tools and with new methods/formats as and when they appear for scheduling and mapping related optimization in MpSoC research domain.

The rest of this paper is structured as follows: in section 2, we discuss the need for a model driven scheduling and mapping framework; in section 3, we present the common extensible graph structure that will allow cooperation between several scheduling and mapping heuristics; then in section 4, we will give heuristic integration examples; we will finally conclude in section 5.

2 Why a Framework?

2.1 Variety of Problems and Solutions

Some proposed scheduling and mapping algorithms are mono-objective, most of them minimize latency; some others take into account several parameters such as computation time, communication time and power consumption. Others give answer to only part of the problem such as mappability for core selection. Another class of algorithms targets some restricted application domain, such as loop nests, for which they are more efficient than generic ones. In order to benefit from the strength of all these heuristics, we need to integrate them in a single framework.

To be able to combine them and to compare their solutions, we need a common application and hardware platform description, based on which we can feed the optimization algorithms their inputs and integrate their results. We propose such a graph based description in section 3.

Furthermore, in order to use different integrated scheduling and mapping algorithms, the problem of algorithm selection, for which part of the application (data or task parallelism...) and for what objective (fast execution, low power...) is a highly critical issue.We propose different approaches to this problem in section 4, the goal being to give to the designer of a complex heterogeneous SoC some help in making one of the most crucial decisions of its design that no current tool provides.

2.2 Model Driven Engineering

What are the benefits of using a model driven approach for building this kind of framework? One is purely technical, the other is user friendliness. Indeed, model driven engineering (MDE) is really about interoperability and abstraction.

Interoperability. MDE is becoming a more and more used approach to develop software. Thus many tools are becoming available to handle models, metamodels and model transformations. These tools often rely on standards that allow interoperability between them. Using model transformation tools, it is relatively easy to extract part of a common model to generate the input file format of a given scheduling tool. Many software libraries exist to manipulate models; we leverage all these to build our framework.

Abstraction. The other strong point of MDE is abstraction. Indeed, models are built to be more readily understandable to humans than programs. The user

can design its application, hardware and their association in a piecewise way, using a top-down or bottom-up approach or a combination thereof, as he wishes and reuse part of previous designs easily.

Obviously, to be able to integrate different scheduling and mapping heuristics, one has to share common models for the application, the hardware architecture and their association. These models have to be extensible to allow for adaptation of future techniques. MDE helps in this aspect by allowing very easily to extend models. We describe in the next section what should these models be like.

3 Graph Based Input Models

3.1 Hierarchical Graphs

Most scheduling and mapping heuristics work on graph based description of the application and the architecture. In order to be able to handle large and complex systems, we should have a hierarchical description of, at least, the application. It is also important to have a hierarchical description of the hardware architecture in order to accommodate different granularity views. One can thus start with a coarse grain optimization and then refine it by going deeper in the hierarchy to reveal more details.

Another required structure of these models is some kind of compact way of representing repetition, such as loops in the application or SIMD units in the hardware. Indeed such a compact representation helps a lot in generating efficient code for these repetitive structures. In this way, we are able to benefit from both task parallelism and data parallelism. Depending on the optimization algorithm used to map and schedule these repetitive structures they may be seen as an indivisible entity or a structured (regular) repetition or even an unstructured (irregular) set of tasks.

We have proposed such repetitive, hierarchical graph models to describe computation intensive embedded applications and SoC hardware platforms in [9, 10, 4]. These models are based on UML2 profiles and are incorporated in the Gaspard2 co-modeling tool [11].

3.2 Characterization

In order to completely define the application and hardware architecture parameters, the graph models need to be characterized. These characteristics are attributes of the nodes and edges of the graphs.

Some of these characteristics are application specific, as real-time constraints of needed computing power. Some are architecture dependent, as instruction set of processors, operating frequency range, power consumption, memory size, communication bandwidth, etc. And some characterize the association of application components to architecture components such as execution time estimations (worst-case or average)[1] or mapping constraints. These are represented as links

[1] Finding good approximations is beyond the scope of this paper and is a research problem in itself.

between application graph elements (nodes or edges) and architecture graph elements.

3.3 Handling Optimization Results

As the optimization results have to be combined, we need a repository to store them. The model driven approach helps us in this prospect as a model is in itself a repository of concepts and links between concepts. We keep a set of models representing partial optimization results. These results are represented as mapping constraints, for the mapping, and as artificial dependencies (links between nodes of the application) for the scheduling. Some specific constructs are used to keep a compact representation of repetitive (or cyclic) schedules and mappings of data-parallel application components onto repetitive hardware (grids of processors, SIMD units or FPGAs).

4 Integration of Techniques

4.1 Framework Usage Schemes

Our framework is environment-sensitive and configurable for applying and selecting scheduling and mapping algorithms. It means, after analyzing application and platform all together the user can follow either of approaches **(A)**,**(B)** or **(C)** below.

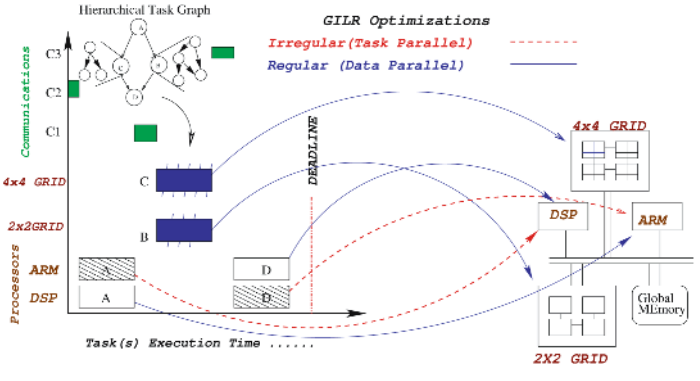


Fig. 1. Simultaneously scheduling and mapping approach (GILR)

(A) *Combine different scheduling and mapping algorithm* on different hierarchical (see figure 1) nodes of the global application model. Here the decision drawn for selecting specific scheduling and mapping algorithms is based on past stored simulation results or on the structure of the application and architecture models or even on results provided by some existing research work. We have already presented this approach as the GILR (Globally Irregular and Locally Regular) heuristic [5]. Here we have shown the combined use of

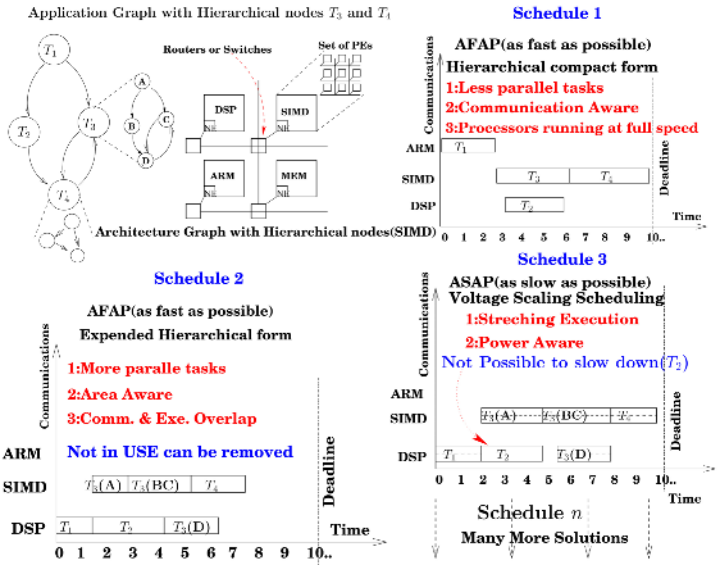


Fig. 2. Iterative scheduling and mapping approach

a list heuristic on a global task graph, and if we find loops on any node, we applied data parallel scheduling and mapping algorithms for loops and map them on the available repetitive hardware components.

(B) *Compare different scheduling and mapping algorithms* on the same hierarchical nodes of the application model. Consider text written in figure 2 as an explation of techniques (Area, Communication & Power Aware etc). Here we are not sure of the quality of results, and we do not have related information in database. Here we spend more time but comparison supports us to find best scheduling and mapping algorithm. The results of scheduling and mapping will be saved in a repository for future considerations.

(C) *User specific*, In this approach the user is fully allowed to take all decisions to select among different scheduling and mapping algorithms, iteratively or simultaneously.

Our repository based approach will allow us to choose the most adapted algorithm for each level of the hierarchy in function of the application structure, available hardware, available characteristics or even optimization criteria.

As a global solution we are expecting to help satisfying multi-criteria problems. This framework will also provide built-in decision capabilities to select one or more scheduling and mapping techniques at a time.

4.2 Implementation

The framework is implemented as a set of common metamodels and APIs. The metamodels define the graph based models described in section 3. The APIs

facilitate the manipulation of the models and the use of existing scheduling and mapping algorithms and the implementation of user specific required scheduling and mapping optimizations. The framework will be flexible and extensible enough to implement newly available scheduling and mapping algorithms in the research domain of MpSoC.

Frameworks like Simgrid [7] and GridSim [6] exists in the field of distributed GRID computing. Such frameworks shares some common basic aspects for scheduling and mapping compared to on-chip distributed computing (MpSoC). Scheduling and mapping algorithms such as list scheduling, critical path scheduling, genetic algorithms, communication scheduling, etc, are common. Where as comparing the architecture platform, MpSoC typically consist of heterogeneous processors among them some nodes can be homogeneous (SIMD) array of processors. Communication networks (NoC) [3] consist of Switches and Routers. But in this communication platform nodes of SoCs are physically close to each other, have high link reliability and low latency.

5 Conclusion

We have sketched[2] in this paper a scheduling and mapping framework for multiprocessor system-on-chip design. This framework aims at integrating various scheduling and mapping heuristics handling part of the optimization problem, either part of the objectives or part of the application domain. In this quest, a recurring issue has been how to apply multiple optimization concepts to the specific problem at hand. Moreover, optimization theory often neglects that aspect which is crucial for the success, namely the models. Therefore, the first step, before any algorithmic properties can be considered, is to establish a model capable of expressing the problem accurately enough.

We have presented common graph based models which are at the heart of the framework. This framework is based on model driven engineering, allowing it to be user friendly and expandable. Several usage schemes of the framework have been proposed.

More precise experiments on the integration of different specific heuristics is underway. In the end, for real-time system design more cost-effective solutions can be found and developed. Furthermore, the design process will be shorten which results in reduced development cost.

References

1. T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: a tool for schedulability analysis and code generation of real-time systems. In *1st International Workshop on Formal Modeling and Analysis of Timed Systems, FORMATS 2003*, Marseille, France, September 6-7 2003.
2. J. Andrew. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. In *15th Artificial Intelligence and Cognitive Science Conference*, Castlebar, Mayo, Ireland., 2004.

[2] More details will be available in a research report.

3. L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, 2002.
4. P. Boulet, J.-L. Dekeyser, C. Dumoulin, and P. Marquet. MDA for System-on-Chip design, intensive signal processing experiment. In *FDL'03*, Fankfurt, Germany, Sept. 2003.
5. P. Boulet and A. Meena. The case for globally irregular locally regular algorithm architecture adequation. In *Journées Francophones sur l'Adéquation Algorithme Architecture (JFAAA'05)*, Dijon, France, Jan. 2005.
6. R. Buyya and M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. In *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, pages Volume 14,13–15,, Wiley Press, Nov.-Dec 2002.
7. H. Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 430, Washington, DC, USA, 2001. IEEE Computer Society.
8. J. Chin and M. Nourani. SoC test scheduling with power-time tradeoff and hot spot avoidance. In *DATE*, pages 710–711, 2004.
9. A. Cuccuru, P. Marquet, and J.-L. Dekeyser. Uml 2 as an adl : Hierarchichal hardware modeling. In *WADL'04*, Toulouse, France, august 2004.
10. C. Dumoulin, P. Boulet, J.-L. Dekeyser, and P. Marquet. UML 2.0 structure diagram for intensive signal processing application specification. Research Report RR-4766, INRIA, Mar. 2003.
11. Gaspard2: Graphical array specification for parallel and distributed computing. `http://www.lifl.fr/west/gaspard/`.
12. J. Jonsson and J. Vasell. Evaluation and comparison of task allocation and scheduling methods for distributed real-time systems. In *ICECCS*, pages 226–229, 1996.
13. J. W. S. Liu, J.-L. Redondo, Z. Deng, T.-S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W.-K. Shih. PERTS: A prototyping environment for real-time systems. Technical Report UIUCDCS-R-93-1802, University of Illinois at Urbana-Champaign, 1993.
14. J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 444–449, New York, NY, USA, 2001. ACM Press.
15. J. L. M. Pasaje, M. G. Harbour, and J. M. Drake. MAST real-time view: A graphic UML tool for modeling object-oriented real-time systems, March 2001.
16. J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, and M. Humphrey. VEST: An aspect-based composition tool for real-time systems. In *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, Canada, May 2003.
17. C.-P. Su and C.-W. Wu. A graph-based approach to power-constrained SOC test scheduling. *J. Electron. Test*, 20(1):45–60, 2004.

Asymmetric Scheduling and Load Balancing for Real-Time on Linux SMP*

Éric Piel, Philippe Marquet, Julien Soula, and Jean-Luc Dekeyser

Laboratoire d'informatique fondamentale de Lille,
Université des sciences et technologies de Lille, France
{Eric.Piel, Philippe.Marquet, Julien.Soula, Jean-Luc.Dekeyser}@lifl.fr

Abstract. The ARTiS system, a real-time extension of the GNU/Linux scheduler dedicated to SMP (Symmetric Multi-Processors) systems is proposed. ARTiS exploits the SMP architecture to guarantee the preemption of a processor when the system has to schedule a real-time task.

The basic idea of ARTiS is to assign a selected set of processors to real-time operations. A migration mechanism of non-preemptible tasks insures a latency level on these real-time processors. Furthermore, specific load-balancing strategies allows ARTiS to benefit from the full power of the SMP systems: the real-time reservation, while guaranteed, is not exclusive and does not imply a waste of resources.

ARTiS have been implemented as a modification of the Linux scheduler. This paper details the evaluation of the performance we conduct on this implementation. The level of observed latency shows significant improvements when compared to the standard Linux scheduler.

1 Real-Time Scheduling on Linux SMP

Nowadays, several application domains require hard real-time support of the operating system: the application contains tasks that expect to communicate with dedicated hardware in a time constrained protocol, for example to insure real-time acquisition. Many of those same real-time applications require large amount of computational power. A well known and effective solution to face this requirement is the usage of SMP (Symmetric Multi-Processors).

Furthermore, to ensure the durability of application developments, one may prefer to target an operating system that conforms to standards. Despite the definition of a standard POSIX interface for real-time applications [3], each vendor of real-time operating system comes with a dedicated API. From our point of view, this segmented market results from the lack of a major player in the real-time community. To face this situation, we believe in the definition of an Open Source operating system that may federate the real-time community. An real-time extension of the well established GNU/Linux operating system is an attractive proposition. Additionally, it will allow the cohabitation of real-time and general purpose tasks in the system.

* This work is partially supported by the ITEA project 01010, HYADES.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 896–903, 2006.

At a first glance, real-time properties relies on the proposition of a dedicated scheduling policy. Three kinds of modification to the Linux scheduler have been proposed. One consists in running the real-time tasks in a special designed kernel running in parallel, this is what does RTAI [2]. The drawback is that the programming model and configuration methods are different from the usual one: Linux tasks are not real-time tasks and real-time activities can not benefit of the Linux services.

The second way taken is to directly modify the Linux scheduler to minimize the path from a blocking context to the re-scheduling of a real-time task. This is, for instance, the base of the work that Ingo Molnar currently carries on. The patch, called "preempt-rt", focuses on hard real-time latencies (which is new, as all the patches before only focused on soft real-time constraints). The objective is to allow every part of the kernel to be preempted, including critical sections and interrupt handlers. The drawback is the degradation of performance for some system calls as well as the high technical difficulty to write and verify those modifications.

The third approach relies on the shielded processors or Asymmetric Multi-Processing principle (AMP). On a multi-processor machine, the processors are specialized to real-time or not. Concurrent Computer Corporation RedHawk Linux variant [1] and SGI REACT IRIX variant [8] follow this principle. However, since only RT tasks are allowed to run on shielded CPUs, if those tasks are not consuming all the available power then there is free CPU time which is lost. The ARTiS scheduler extends this second approach by also allowing normal tasks to be executed on those processors as long as they are not endangering the real-time properties.

2 ARTiS: Asymmetric Real-Time Scheduler

Our proposition is a contribution to the definition of a real-time Linux extension that targets SMPs. Furthermore, the programming model we promote is based on a user-space programming of the real-time tasks: the programmer uses the usual POSIX and/or Linux API to define his applications. These tasks are real-time in the sense that they are identified with a high priority and are not perturbed by any non real-time activities. For these tasks, we are targeting a maximum response time below 300μs. This limit was obtained after a study by the industrial partners concerning their requirements.

To take advantage of an SMP architecture, an operating system needs to take into account the shared memory facility, the migration and load-balancing between processors, and the communication patterns between tasks. The complexity of such an operating system makes it look more like a general purpose operating system (GPOS) than a dedicated real-time operating system (RTOS). An RTOS on SMP machines must implement all these mechanisms and consider how they interfere with the hard real-time constraints. This may explain why RTOS's are almost mono-processor dedicated. The Linux kernel is able to efficiently manage SMP platforms, but it is agreed that the Linux kernel has not

been designed as an RTOS. Technically, only soft real-time tasks are supported, via the FIFO and round-robin scheduling policies.

The ARTiS solution keeps the interests of both GPOS's and RTOS's by establishing from the SMP platform an **A**symmetric **R**eal-**Ti**me **S**cheduler in Linux. We want to keep the full Linux facilities for each process as well as the SMP Linux properties but we want to improve the real-time behavior too. The core of the ARTiS solution is based on a strong distinction between real-time and non-real-time processors and also on migrating tasks which attempt to disable the preemption on a real-time processor.

Partition of the Processors and Processes. Processors are partitioned into two sets, an NRT CPU set (Non-Real-Time) and an RT CPU set (Real-Time). Each one has a particular scheduling policy. The purpose is to insure the best interrupt latency for particular processes running in the RT CPU set.

Two classes of RT processes are defined. These are standard RT Linux processes, they just differ in their mapping:

- Each RT CPU has one or several bound RT Linux tasks, called **RT0** (a real-time task of highest priority). Each of these tasks has the guarantee that its RT CPU will stay entirely available to it. Only these user tasks are allowed to become non-preemptible on their corresponding RT CPU. This property insures a latency as low as possible for all RT0 tasks. The RT0 tasks are the hard real-time tasks of ARTiS. Execution of more than one RT0 task on one RT CPU is possible but in this case it is up to the developer to verify the feasibility of such a scheduling.
- Each RT CPU can run other RT Linux tasks but **only** in a preemptible state. Depending on their priority, these tasks are called RT1, RT2... or RT99. To generalize, we call them **RT1+**. They can use CPU resources efficiently if RT0 tasks do not consume all the CPU time. To keep a low latency for the RT0 tasks, the RT1+ tasks are automatically migrated to an NRT CPU by the ARTiS scheduler when they are about to become non-preemptible (when they call `preempt_disable()` or `local_irq_disable()`). The RT1+ tasks are the soft real-time tasks of ARTiS. They have no firm guarantees, but their requirements are taken into account by a best effort policy. They are also the main support of the intensive processing parts of the targeted applications.
- The other, non-real-time, tasks are named "Linux tasks" in the ARTiS terminology. They are not related to any real-time requirements. They can coexist with real-time tasks and are eligible for selection by the scheduler as long as the real-time tasks do not require the CPU. As for the RT1+, the Linux tasks will automatically migrate away from an RT CPU if they try to enter into a non-preemptible code section on such a CPU.
- The NRT CPUs mainly run Linux tasks. They also run RT1+ tasks which are in a non-preemptible state. To insure the load-balancing of the system, all these tasks can migrate to an RT CPU but only in a preemptible state. When an RT1+ task runs on an NRT CPU, it keeps its high priority above the Linux tasks.

ARTiS then supports three different levels of real-time processing: RT0, RT1+ and Linux. RT0 tasks are implemented in order to minimize the jitter due to non-preemptible execution on the same CPU. Note that these tasks are still user-space Linux tasks. RT1+ tasks are soft real-time tasks but they are able to take advantage of the SMP architecture, particularly for intensive computing. Eventually, Linux tasks can run without intrusion on the RT CPUs. Then they can use the full resources of the SMP machines. This architecture is adapted to large applications made of several components requiring different levels of real-time guarantees and of CPU power.

Migration Mechanism. A particular migration mechanism has been defined. It aims at insuring the low latency of the RT0 tasks. All the RT1+ and Linux tasks running on an RT CPU are automatically migrated toward an NRT CPU when they try to disable the preemption. One of the main requirement is a mechanism without any inter-CPU locks. Such locks are extremely dangerous for the real-time properties if an RT CPU have to wait after an NRT CPU. To effectively migrate the tasks, an NRT CPU and an RT CPU have to communicate via queues. We have implemented a lock-free FIFO with one reader and one writer to avoid any active wait of the ARTiS scheduler based on the algorithm proposed by Valois [9].

Load-Balancing Policy. An efficient load-balancing policy allows the full power of the SMP machine to be exploited. Usually a load-balancing mechanism aims to move the running tasks across CPUs in order to insure that no CPU is idle while tasks are waiting to be scheduled. Our case is more complicated because of the introduction of asymmetry and the heavy use of real-time tasks. To minimize the latency on RT CPUs and to provide the best performances for the global system, particular asymmetric load-balancing algorithms have been defined [7].

3 ARTiS Current Implementation

A basic ARTiS API has been defined. It allows the deployment of applications on the current implementation of the ARTiS model, defined as a modification of the 2.6 Linux kernel. A user defines its ARTiS application by configuring the CPUs, identifying the real-time tasks and their processor affinity via a basic `/proc` interface and some system calls (`sched_setscheduler()`...).

The current implementation[5] first consists of a migration mechanism that ensures only preemptible code is executed on an RT CPU. This migration relies on a task FIFO implemented with lock-free access [9]: one writer on the RT CPU and one reader on the NRT CPU.

The load-balancer implementation was carried out by improving or specializing several parts of the original Linux one. The main modification was to change from a “pull” policy to a “push” policy: it is the over-loaded CPUs which send tasks to the under-loaded ones. Although it slightly decreases performances because idle CPUs might spend more time idle, this permitted the removal of

inter-CPU locks. The load estimation of a processor has also been enhanced in order to correctly estimate a load of a real-time task (which will never share CPU power with other tasks). This enhancement permit better equity among Linux tasks. Additionally, the designation criteria has been modified to favor the presence of RT1+ (resp. Linux) tasks on RT CPUs (resp. NRT CPUs) because that is where latencies are the smallest. Similarly, tasks which are unlikely to block the interrupts soon (according to statistics about their previous behavior) will be preferred for going to an RT CPU.

A specific tool was designed to test the load-balancer correctness. It allows to run a specific set of tasks characterized by properties (CPU usage, scheduler priority, processor affinity...) to be launched in a very deterministic way. Some statistics about the run are provided but the interpretation of the results is not straightforward. Additional studies need to be done on the load-balancer implementation.

4 Performance Evaluation

While implementing the ARTiS kernel, some experiments were conducted in order to evaluate the potential benefits of the approach in terms of interrupt latency. We distinguished two types of latency, one associated with the kernel and the other one associated with user tasks.

Measurement Method. The experiment consisted of measuring the elapsed time between the hardware generation of an interrupt and the execution of the code concerning this interrupt. The experimentation protocol was written with the wish to stay as close as possible to the common mechanisms employed by real-time tasks. The measurement task sets up the hardware so it generates the interrupt at a precisely known time, then it gets unscheduled and wait for the interrupt information to occur. Once the information is sent, the task is woken up, the current time is saved and the next measurement starts. For one interrupt there are four associated times, corresponding to different locations in the executed code (figure 1):

- t'_0, the interrupt programming,
- t_0, the interrupt emission, it is chosen at the time the interrupt is launched,
- t_1, the entrance in the interrupt handler specific to this interrupt,
- t_2, the entrance in the user-space RT task.

We conducted the experiments on a 4-way Itanium II 1.3GHz machine. It ran on a instrumented Linux kernel version 2.6.11. The interrupt was generated with a cycle accurate precision by the PMU (a debugging unit available in each processor [6]).

Even with a high load of the computer, bad cases leading to long latencies are very unusual. Thus, a large number of measures are necessary. In our case, each test was run for 8 hours long, this is equivalent to approximately 300 million measures. Given such duration, the results are reproducible.

Interrupt Latency Types. From the three measurement locations, two values of interest can be calculated:

- The **kernel latency**, $t_1 - t_0$, is the elapsed time between the interrupt generation and the entrance into the interrupt handler function. This is the latency that a driver would have if it was written as a kernel module.
- The **user latency**, $t_2 - t_0$, is the elapsed time between the interrupt generation and the execution of the associated code in the user-space real-time task. This is the latency of a real-time application entirely written in user-space. In order to have the lowest latency, the application was notified via a blocking system call (a `read()`).

The real-time tasks designed to run in user-space are programmed using the usual and standard POSIX interface. This is one of the main advantage that ARTiS provides. Therefore, within the ARTiS context, user latency is the most important latency to study and analyze.

Measurement Conditions. The measurements were conducted under four configurations. Those configurations were selected for their relevance toward latency. First of all, the standard (vanilla) kernel was measured without and with load. Then, a similar kernel but with the preemption activated was measured. When activated, this new feature of the 2.6 Linux kernel allows tasks to be rescheduled even if kernel code is being executed. Finally, the current ARTiS implementation was measured. Only the first kernel is also presented when idle because the results with the other kernels are extremely similar.

In the experiments, the system load consisted of busying the processors by user computation and triggering a number of different interruptions in order to maximize the activation of the inter-locking and the preemption mechanisms. Five types of program corresponding to five loading methods were used:

- **Computing load:** A task that executes an endless loop without any system call is pinned on each processor, simulating a computational task.
- **Input/output load:** The `iodisk` program reads and writes continuously on the disk.

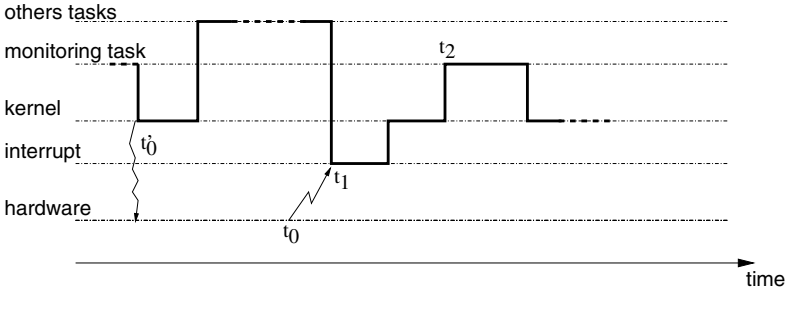


Fig. 1. Chronogram of the tasks involved in the measurement code

- **Network load:** The `ionet` program floods the network interface by executing `ICMP` echo/reply.
- **Locking load:** The `ioctl` program calls the `ioctl()` function that embeds a *big kernel lock*.
- **Cache miss load:** The `cachemiss` program generates a high rate of cache misses on each processors. This adds latencies because a cachemiss implies a full cache line is read from the memory, blocking the CPU for a long time.

Observed Latencies. The table 1 summarizes the measurements for the different tested configurations. Two values are associated to each latency type (kernel and user). "Maximum" corresponds to the highest latency noticed along the 8 hours. The other column displays the maximum latency of the 99.999% best measures. For this experiment, this is equivalent to not counting the 3000 worse case latencies.

Table 1. Kernel/User latencies of the different configurations

		Kernel		User	
Configurations		99.999%	Maximum	99.999%	Maximum
standard Linux	idle	1μs	6μs	5μs	78μs
standard Linux	loaded	6μs	63μs	731μs	49ms
Linux with preemption	loaded	4μs	60μs	258μs	1155μs
ARTiS	loaded	8μs	43μs	18μs	104μs

The study of the idle configuration gives some comparison points when measured against the results of the loaded systems. While the kernel latencies are nearly unaffected by the load, the user latencies are several orders bigger. This is the typical problem with Linux, simply because it was not designed with real-time constraints in mind. We should also mention that for all the measurement configurations the average user latency was under 4μs.

The kernel preemption does not change the latencies at the kernel level. This was expected as the modifications focus only on scheduling faster user tasks, nothing is changed to react faster on the kernel side. However, with regard to user-space latencies, a significant improvement can be noticed in the number of observed high latencies: 99.999% of the latencies are under 238μs instead of 731μs. This improvement is even better concerning the maximum latency, which is about forty times smaller. This enhancement permits soft real-time with better results than the standard kernel, still, in our case (latencies always under 300μs), this cannot be considered as a hard real-time system.

The ARTiS implementation reduces again the user latencies, with a maximum of 104μs. The results obtained from the measurements of the current ARTiS should not change as the additional features will only focus on performance enhancements, not on latencies. Consequently, the system can be considered as a hard real-time system, insuring real-time applications very low interrupt response.

5 Conclusion

We have proposed ARTiS, a scheduler based on a partition of the multiprocessor CPUs into RT processors where tasks are protected from jitter on the expected latencies and NRT processors where all the code that may lead to a jitter is executed. This partition does not exclude a load-balancing of the tasks on the whole machine, it only implies that some tasks are automatically migrated when they are about to become non-preemptible.

An implementation of ARTiS was evaluated on a 4-way IA-64 and a maximum user latency as low as 104μs can be guaranteed (against latencies in the 1100μs range for the standard 2.6 Linux kernel). The implementation is now usable. The ARTiS patches for the Linux 2.6 kernel are available for Intel i386 and IA-64 architectures from the ARTiS web page [4].

A limitation of the current ARTiS scheduler is the consideration of multiple RT0 tasks on a given processor. Even if ARTiS allows multiple RT0 tasks on one RT processor, it is up to the programmer to guarantee the schedulability. We plan to add the definition of usual real-time scheduling policies such as EDF (earliest deadline first) or RM (rate monotonic). This extension requires the definition of a task model, the extension of the basic ARTiS API and the implementation of the new scheduling policies. The ARTiS API would be extended to associate properties such as periodicity and capacity to each RT0 task. A hierarchical scheduler organization would be introduced: the current highest priority task being replaced by a scheduler that would manage the RT0 tasks.

References

1. Steve Brosky and Steve Rotolo. Shielded processors: Guaranteeing sub-millisecond response in standard Linux. In *Workshop on Parallel and Distributed Real-Time Systems, WPDRTS'03*, Nice, France, April 2003.
2. Pierre Cloutier, Paolo Montegazza, Steve Papacharalambous, Ian Soanes, Stuart Hughes, and Karim Yaghmour. DIAPM-RTAI position paper. In *Second Real Time Linux Workshop*, Orlando, FL, November 2000.
3. Bill Gallmeister. *POSIX.4, Programming for the Real World*. O'Reilly & Associates, 1994.
4. Laboratoire d'informatique fondamentale de Lille, Université des sciences et technologies de Lille. ARTiS home page. `http://www.lifl.fr/west/artis/`.
5. Philippe Marquet, Éric Piel, Julien Soula, and Jean-Luc Dekeyser. Implementation of ARTiS, an asymmetric real-time extension of SMP Linux. In *Sixth Realtime Linux Workshop*, Singapore, November 2004.
6. David Mosberger and Stéphane Eranian. *IA-64 Linux Kernel : Design and Implementation*. Prentice-Hall, 2002.
7. Éric Piel, Philippe Marquet, Julien Soula, and Jean-Luc Dekeyser. Load-balancing for a real-time system based on asymmetric multi-processing. In *16th Euromicro Conference on Real-Time Systems, WIP session*, Catania, Italy, June 2004.
8. Sillicon Graphics, Inc. REACT: Real-time in IRIX. Technical report, Sillicon Graphics, Inc., Mountain View, CA, 1997.
9. John D. Valois. Implementing lock-free queues. In *Proceedings of the Seventh International Conference on Parallel and Distributed Computing Systems*, Las Vegas, NV, October 1994.

Artificial Immune Systems Applied to Multiprocessor Scheduling

Grzegorz Wojtyla[1], Krzysztof Rzadca[2,3,⋆], and Franciszek Seredynski[2,4,5]

[1] Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
[2] Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warsaw, Poland
[3] Laboratoire Informatique et Distribution,
51 avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France
rzadca@imag.fr
[4] Institute of Computer Science, University of Podlasie,
Sienkiewicza 51, 08-110 Siedlce, Poland
[5] Institute of Computer Science, Polish Academy of Sciences,
Ordona 21, 01-237 Warsaw, Poland
sered@ipipan.waw.pl

Abstract. We propose an efficient method of extracting knowledge when scheduling parallel programs onto processors using an artificial immune system (AIS). We consider programs defined by Directed Acyclic Graphs (DAGs). Our approach reorders the nodes of the program according to the optimal execution order on one processor. The system works in either learning or production mode. In the learning mode we use an immune system to optimize the allocation of the tasks to individual processors. Best allocations are stored in the knowledge base. In the production mode the optimization module is not invoked, only the stored allocations are used. This approach gives similar results to the optimization by a genetic algorithm (GA) but requires only a fraction of function evaluations.

1 Introduction

Parallel computers, ranging from networks of standard PCs to specialized clusters, supercomputers and the Grid, are becoming very popular and widely used. However, such systems are more difficult to use than the standard ones. Especially, mapping and scheduling individual tasks of a parallel program onto available resources is, excluding some very bounded conditions, NP-hard.

Heuristic approaches to scheduling have been extensively studied [7] [6]. However, usually those algorithms are not very robust and work well only on certain types of programs. Search-based methods, which employ some global optimization meta-heuristics, were also used [10]. Nevertheless, those algorithms suffer from a large execution time which usually does not compensate the increased

⋆ Krzysztof Rzadca is partly supported by the French Government Grant number 20045874.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 904–911, 2006.

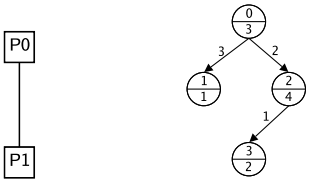


Fig. 1. An example system (a) and program (b) graphs. On the program graph, the top number shows the label of the task, the bottom one – the time necessary for computation. $V_s = \{P0, P1\}; E_s = \{P0 \leftrightarrow P1\}$; $V_p = \{0, 1, 2, 3\}; E_p = \{0 \rightarrow 1, 0 \rightarrow 2, 2 \rightarrow 3\}$.

quality of the solutions [9]. Certainly a fast scheduler which could adapt to the conditions on a particular system would be more efficient.

In this paper we propose a novel approach to scheduling parallel programs based on AIS. Our system performs population-based optimization in order to find an approximation of the optimal schedule. It also extracts the knowledge from individual schedules, stores it and uses it later to provide near-optimal schedules for similar parallel programs without running the optimization module.

This paper is organized as follows. Section 2 gives a description of the scheduling problem considered. Section 3 outlines the concept of AIS. In Section 4 we present a scheduler which uses AIS. Experiments and results are described in the Section 5. We conclude the paper and present some directions of further works in Section 6.

2 Scheduling Parallel Programs

The architecture of the system on which the program is scheduled is described by a system graph, $G_s = (V_s, E_s)$ (Figure 1a). The nodes V_s, called processors, represent processing units. The edges E_s represent connections between processors. We assume homogeneous model – the time needed for computation of a task is the same on each processor.

The parallel program to be scheduled is represented by a directed acyclic graph (DAG), $G_p = (V_p, E_p)$, where $V_p = \{v_p\}$ is a set of $n \equiv |V_p|$ nodes representing individual, indivisible tasks of the program (Figure 1b). Later we will use the terms node and task interchangeably. The weight of a node v_i gives the time needed for the computation of this task. $E_p = \{e_{i,j}\}$ is the set of edges which represent dependencies and communication between the nodes. If the nodes v_i and v_j are connected by an edge $e_{i,j}$, the task v_j cannot start until v_i finishes and the data is transmitted. During the execution of the program a task v_j is called a *ready* task, if all such v_i are finished. The weight of an edge $e_{i,j}$ defines the time needed to send the data from the task v_i to v_j if those two tasks execute on neighboring processors. If those tasks execute on the same processor we assume that the time needed for the communication is neglectable. If there is no direct link between the processors, the time needed

for communication is a product of the edge's weight and the minimum number of hops (distance) between processors. We assume that, once task's execution has begun, it cannot be interrupted until the task finishes (i.e. non-preemptive model). We also assume that each task is executed on exactly one processor – we do not allow task duplication.

Our system minimizes the total execution time (makespan) T of the program. We assume that T is a function of *allocation* and *scheduling policy*: $T = f(alloc, policy)$. The scheduling policy determines the order of the execution of conflicting tasks on processors. Firstly, it ranks the tasks according to some criteria based on the program graph, such as the size of the sub-tree of the node (which will be used later in our experiments), the sum of computation costs on the longest path between the node and the sink node (s-level), or others [8]. If, during the program's execution, two or more tasks assigned to the same processor are ready, the one with higher rank is chosen. We consider that the policy is given a priori. The allocation is a function which determines the processor on which each task of a program will be executed: $alloc : V_p \rightarrow V_s$. Given the allocation and the policy it is trivial to schedule tasks onto processors.

3 Artificial Immune Systems

Biological immune systems can be viewed as a powerful distributed information processing systems, capable of learning and self-adaptation [2]. They have a number of interesting features [4], which include totally distributed architecture, adaptation to previously unseen antigens (molecules foreign to the system), imperfect detection (system detects antigens which match receptors even partially) and memory. They are a source of constant inspiration to various computing systems, called Artificial Immune Systems (AIS). Some of the previous attempts to use an AIS to solve computing problems include binary pattern recognition, scheduling [5], or intrusion detection in computer networks.

AIS act upon a population of antibodies (responsible for antigen detection and elimination) which are modified in function of the affinity between antibodies and antigens [3]. In pattern recognition, the degree of affinity is proportional to the match between an antibody and an antigen, randomly chosen from a set of patterns to recognize. In optimization, where there is no explicit antigen population, antibody represents a point in the search space, so its affinity is related to the value of the objective function for this point. The antibodies which match the antigen best (which have highest affinity) are cloned. The number of clones produced is proportional to the affinity of the antibody. Then the clones undergo a process called hypermutation, with the probability of mutation inversely proportional to the parent's affinity. The clones are then evaluated and some of the best ones replace their parents, if they have better affinity. The worst antibodies are replaced by new ones, either randomly initialized or created by immunological cross-over [1].

4 Immune-Based Scheduler

In real-world applications, similar programs are presented to the scheduler repeatedly. This can be caused by typical development cycle (run – debug – modify), or by running the same program on similar datasets. This suggests that one can store previous solutions to the scheduling problem and try to use it when scheduling new programs.

Immune-based scheduler (IBS) is composed of the coding module, the knowledge base (KB) and the optimization module. The coding module encodes the DAG of every program entering the system by reordering individual tasks according to the schedule produced for a single processor by a list scheduling algorithm. The system then acts on encoded programs. KB stores the information about previous allocations, so it can be reused when allocating similar programs. The optimization module is responsible for searching the possible allocation space and delivering the approximation of the optimal allocation.

Our system works in either learning or production mode. The learning mode can be viewed as a vaccination of the system by new antigens – programs to be allocated. The system learns how to deal with the antigen (how to allocate it) by producing antibodies (possible allocations) and evaluating them. The shorter the schedule produced by allocating the program with the antibody, the better the affinity of the antibody. The best antibody found is then saved in the KB. In the production mode the system does not have time to perform the optimization. Based on the knowledge stored in KB, it must quickly provide an immunological answer (an allocation) for a given program.

4.1 Learning Mode

When a new program (containing of $n \equiv |V_p|$ tasks) is presented to the system running in the learning mode, after the initial encoding, the optimization module is invoked. We use an artificial immune system to find an approximation of the optimal allocation. One can, however, use any global search method, such as a GA, which returns an approximate optimal allocation of a given program.

In our implementation, each individual represents a possible allocation of the program. The kth gene of the individual determines the allocation of the kth encoded task (a task v_j which would be executed as kth on a single processor). Such an encoding does not produce any unfeasible individuals.

The initial population of solutions is either totally random or formed partly by random individuals and partly by individuals taken from the knowledge base (adjusted to the length of the program by the algorithm described in the next section). Firstly, the algorithm computes the makespan T_i for each individual i by simulating the execution of the program with the allocation encoded in the individual i. Then the population is ranked – the best individual (the one with the shortest makespan) is assigned the rank $r = 0$. Then, a following optimization algorithm is run:

REPEAT UNTIL endAlgorithm

1. **clonal selection:** the best I individuals are cloned. The number of clones C_i produced for the individual i is inversely proportional to the rank r of the individual: $C_i = \frac{C_0}{2^r}$, where C_0, the number of clones for the best individual, is a parameter of the operator.
2. **hypermutation:** each clone is mutated. The mutation ratio of the individual i is inversely proportional to the rank r of the clone's parent. The number k of genes to mutate is given by
   $k = \text{round}\left(P_0 * (1 + \Delta P * r) * n\right)$,
   where P_0 is the average probability of gene's mutation in the best individual, increased by ΔP for each following individual. The algorithm then randomly selects k genes and sets them to a random value.
3. **clones' evaluation:** the makespan T is computed for each clone.
4. **merging**: K best clones are chosen. Each one of them replaces its parent in the population if the length of the clone's schedule is less than parent's.
5. **worst individual replacement:** L worst individuals from the population are replaced by randomly initialized ones.
6. **immunological cross-over:** M worst individuals of the population (excluding those modified in the previous step) are replaced by individuals created in the cross-over process. Each new individual has three parents, randomly chosen from the population and sorted by their ranks. If the best two parents have the same value of a gene, it is copied to the child. If they differ, the value for the gene is selected at random from the values of this gene in two best parents. However, the value equal to the value of this gene in the worst (third) parent has much smaller probability.

We end the algorithm after a limited number of iterations or sooner if the best individual does not change for a certain number of iterations. The result of the algorithm is the best allocation found for a given program. The individual representing the best allocation is then stored in the KB.

4.2 Production Mode

In the production mode when a new program enters the system the optimization module is not invoked. Firstly, the program is encoded. Then a temporary population of allocations is created. The system copies to the temporary population each allocation from the KB adjusted to the length of the program. If the length k of the allocation taken from the KB is equal to the number n of tasks in the program, the unmodified allocation is copied to the temporary population. If $k > n$, the allocation in the temporary population is truncated – the first n genes are taken. If $n < k$, the system creates $\text{CloneNum} * ((n - k) + 1)$ allocations in the temporary population (CloneNum is a parameter of the algorithm). In each newly created allocation the old genes are shifted (so that every possible position is covered). Values for the remaining $n - k$ genes are initialized randomly. In the next step the system finds the best allocation in the temporary population. The execution of the program is simulated with each allocation and the resulting makespan T is computed. The allocation which gives the minimum makespan T is returned as the result.

Table 1. Comparison of the IBS running in the learning mode, the EFT and the GA

program	T_{best}			T_{avg}		*evaluations* $[*10^3]$	
graph	ETF	GA	IBS	GA	IBS	GA	IBS
tree15	9	9	9	9	9	**10.2**	17.6
g18	46	46	46	46	46	**10.2**	17.6
gauss18	52	44	44	44.3	**44**	**11.4**	19.9
g40	81	80	80	80	80	**10.4**	17.8
g25-1	530	495	495	495	495	**11.1**	18.6
g25-5	120	97	97	97.8	**97**	**11.6**	23.5
g25-10	78	62	62	70.2	**65.4**	**12.8**	27.1
g50-1	938	890	890	890	890	**12.1**	19.9
g50-5	235	**208**	210	**213.3**	215.1	**19.7**	39.9
g50-10	181	139	139	**146.4**	148.4	**16.6**	44.4
g100-1	1526	1481	1481	1481	1481	**11.9**	20.2
g100-5	460	**404**	415	**409.7**	421.1	**32.7**	43.1
g100-10	207	**176**	177	**178.3**	180.9	**20.3**	42.3

5 Experimental Results

We tested our approach by simulating the execution of programs defined by random graphs and graphs that represent some well-known numerical problems. The name of the random graph contains an approximate number of nodes and the communication to computation ratio (CCR). We focused on the two-processor system. We compared our approach (IBS) with a list-scheduling algorithm (EFT, homogeneous version of HEFT [9]) and a GA. The GA we used operates on the same search space as the IBS and is a specific elitist GA, with cross-over only between the members of the elite. We used sub-tree size as the scheduling policy in both the GA and the IBS. The parameters of the GA were tuned for the most difficult problem we considered (*g100-1*). The IBS parameters where then fixed so that both algorithms perform a similar number of function evaluations in one iteration. The parameters of the IBS are as follows: PopulationSize = 200, $I = 60$, $C_0 = 75$, $P_0 = 0.10$, $\Delta P = 0.05$, $K = 60$, $L = 70$, $M = 75$. In the production mode, only one allocation was produced for each possible shift (CloneNum = 1). Both algorithms were ended if the best individual was the same during NoChangeIter = 50 iterations, but not later than after 1000 iterations.

We run each algorithm on each program graph ten times. The shortest makespan found in ten runs T_{best} and the average from the ten makespans T_{avg} are presented. We also present the average number of function evaluations in ten runs, as this value gives a good estimation of the run time of the algorithm.

In the first set of experiments we compared the results of the GA and the IBS running in the learning mode (Table 1). As the values returned by both algorithms are very similar, we suppose that the search space is explored well by both approaches. On the easier graphs (*tree15,g18*) both algorithms found the optimal values in the first or the second iteration. However, in the harder

graphs (e.g. *g100-1*) when we tried to limit the number of function evaluations performed by the IBS algorithm (for example by reducing the NoChangeIter parameter), the values returned were deteriorating quickly. When comparing the results with the EFT algorithm, we observed that on harder graphs the difference in the length of schedules produced is about 5% to 30%. One can see that the higher the CCR, the longer makespan is produced by the EFT. The results of this experiments show that the global search method we use in the learning mode efficiently scans the search space and delivers results as good as a GA. They do not, however, show that our search algorithm performs better that the GA, because our algorithm performs twice as many function evaluations.

Table 2. Comparison of the IBS running in the production mode, the EFT and the GA

program	T_{best}			T_{avg}		*evaluations* $[*10^3]$	
graph	ETF	GA	IBS	GA	IBS	GA	IBS
tree15→g18	55	55	55	**55**	55.7	10.4	**0.1**
g40→g18	127	**126**	127	**126**	129.4	11	**0.3**
g40→g18→tree15	136	**135**	138	**135**	139.4	11.5	**0.4**
g25-1→g50-1	1468	**1385**	1424	**1385.2**	1436	18.7	**0.6**
g25-1→gauss18	582	**539**	550	**541.7**	565.9	14.8	**0.2**
g40→g50-1	1019	**970**	978	**970**	988	15.1	**0.7**
g25-1→g100-1	2056	**1976**	2002	**1976.5**	2025.4	19.7	**1.1**
g50-1→g50-5	1173	**1102**	1146	**1106**	1156.1	35.4	**0.9**

In the second set of experiments we tested the immunological features of our system. Firstly we had run the IBS system in the learning mode on all the graphs. The best allocation found for each graph had been stored in the KB. Then we prepared some new graphs by joining all the exit nodes of one program graph with all the entry nodes of the other graph with edges with 0 communication costs. Table 2 summarizes the results obtained. We can see that our system is able to store knowledge and reuse it to schedule efficiently new programs. The results returned by the IBS running in the production mode are very close to those obtained by the GA. However, the IBS needs only a fraction of function evaluations needed by the GA. One can argue that the function evaluations "saved" in the production mode are used in the learning phase. We cannot agree with this. In the typical parallel systems schedulers are invoked repeatedly and with very tight time limits. Such a system really needs a fast scheduler in order not to waste too much processor time just to preprocess users' programs. The other factor is that the overall load of the system is very uneven. On one hand, there are peaks, when many jobs are waiting in the queue (e.g. before a deadline of an important conference). On the other, periods (e.g. normal weekends) when most of the processors are unused. System administrator can use those free periods to run the learning phase of the IBS. Comparing the IBS and the EFT results, our system performs constantly better on harder graphs, even when taking into account the average from the length of schedule returned. Nevertheless, the advantage the IBS had in learning mode (5% to 30%) is reduced to 2%–5%.

6 Conclusion and Future Work

This paper presents a new approach for scheduling programs given by a DAG graph. The most important result presented here is the ability to extract the knowledge from previous schedules and to use it when scheduling new, potentially similar, programs. After the initial phase of learning our system can provide a near-optimal schedule for previously unseen problems quickly.

The results presented here open very promising possibilities for further research. We plan to construct new schedules from the knowledge base more carefuly. We also plan to deconstruct both the system and the program graph. Instead of storing schedules of complete programs on the whole system, we would like to schedule parts of graphs on locally-described parts of system.

References

1. H. Bersini. The immune and the chemical crossover. *IEEE Trans. on Evol. Comp.*, 6(3):306–314, 2002.
2. D. Dasgupta. Information processing in the immune system. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 161–165. McGraw-Hill, London, 1999.
3. L.N. de Castro and F.J. Von Zuber. Learning and optimization using the clonal selection principle. *IEEE Trans. on Evol. Comp.*, 6(3):239–251, 2002.
4. P.K. Harmer, P.D. Williams, G.H. Gunsch, and G.B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Trans. on Evol. Comp.*, 6(3):252–280, 2002.
5. E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In *GECCO-99: Proceedings of the Genetic and Evol. Comp. Conference*, pages 1559–1565. Morgan Kaufmann, 1999.
6. E. Hart, P. Ross, and D. Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.
7. Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.
8. F. Seredynski and A.Y. Zomaya. Sequential and parallel cellular automata-based scheduling algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 13(10):1009–1023, 2002.
9. H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, 2002.
10. A.S. Wu, H. Yu, S. Jin, K.-C. Lin, and G. Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 15(9):824–834, 2004.

Toward an Application Support Layer: Numerical Computation in Unified Parallel C

Jonathan Leighton Brown[1,*] and Zhaofang Wen[2,**]

[1] University of Michigan, Ann Arbor MI 48109, USA
jonlbro@eecs.umich.edu
[2] Sandia National Laboratories, Albuquerque NM 87185, USA
zwen@sandia.gov

Abstract. Unified Parallel C is a parallel language extension to standard C. Data in UPC are communicated through shared arrays, which are physically distributed. Thus, data regions have locality, or affinity, to particular threads of execution. This affinity concept engenders a non-uniformity in shared memory accesses by a particular thread. Affinity should be considered when building data structures in algorithms and applications, but UPC provides limited tools for data locality management. We propose the creation of an application support layer to support a wide variety of common data decompositions and programming idioms. We present here a first step for this layer with a selection of mapping functions and packages for numerical computation and dense matrix operations. These are driven by specific algorithms from linear algebra and numerical computation, and could be readily incorporated in such an application support layer.

1 Introduction

Virtual shared memory programming offers the advantages of shared memory programming in terms of programmer ease and reduced man hours with the promise of performance competitive with message passing schemes. Global Address Space programming languages, including Unified Parallel C, Co-Array Fortran, and Titanium, provide language extensions to C, Fortran, and Java, respectively, to add explicit parallel constructs [7, 11, 15]. These implementations rely on runtime and compiler support to achieve good performance relative to MPI. Our work focuses on Unified

* This research was performed while on appointment as a U.S. Department of Homeland Security (DHS) Fellow under the DHS Scholarship and Fellowship Program, a program administered by the Oak Ridge Institute for Science and Education (ORISE) for DHS through an interagency agreement with the U.S Department of Energy (DOE). ORISE is managed by Oak Ridge Associated Universities under DOE contract number DE-AC05-00OR22750. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of DHS, DOE, or ORISE. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

** Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 912–919, 2006.

Parallel C, which has seen wide endorsement, with compilers available from most major parallel system vendors and active research at national laboratories and universities.

A barrier to more widespread acceptance of UPC is the lack of library support for algorithm developers. UPC uses a simple data structure – a shared array accessible to all threads of execution – for parallel data sharing. Management of data accesses to minimize remote accesses and achieve good data decomposition is necessary to get the maximal performance out of a UPC program. UPC requires the programmer to explicitly manage mapping of data to the UPC shared array structure. We propose the development of an application support layer to provide useful abstractions and provide a ready toolkit for UPC developers to focus on developing solutions to problems instead of the underlying physical data layout.

Application support layers are quite common in MPI. Several domain-specific support libraries have been developed: PLAPACK is a dense matrix manipulation library for MPI that takes an algorithm-driven approach to data decomposition [13]. METIS provides graph manipulation and decomposition packages; its ParMETIS package is a parallel graph partitioning library built on top of MPI [9]. More general-purpose libraries have been developed, such as Zoltan, a lightweight library for data decomposition and parallelization [1]. Zoltan places few restrictions on the algorithm data structures, instead providing a handful of interface functions to be defined by the application programmer, and places a wrapper over all MPI communications to hide explicit message passing and data movement. Framework projects, like Trilinos, include specific application service libraries as well as support for data decomposition and management built on MPI [8]. Recent work has focused on more elaborate and dynamic data structure support for MPI with automatic load balancing [12].

We present here a first step toward an application support layer for UPC: static decompositions for dense matrix operations. We present mapping functions for manipulation of one-dimensional arrays. As an added bonus, these harmonize the view of shared array structures between UPC and Co-Array Fortran. We then provide algorithm-motivated decompositions of two-dimensional, shared arrays and conclude with an extended example of how these can be used to manage locality of data accesses. These can be made standard tools; this added expressiveness would make UPC a more attractive environment and help performance by providing structured communication and seeking to make clearer the separation between local and remote data in shared objects.

The mapping functions are presented as an add-on library. Recent work (e.g., [4]) has focused on improving the performance of UPC shared pointers with local look-up tables and other private data structures. The proposal presented here is compatible with improved shared pointers, as it allows programmers to write code that respects data locality, saving on communication costs incurred after shared pointer dereferencing.

The rest of the paper is organized as follows. Section 2 includes constructs for manipulation of one-dimensional, shared arrays in UPC. In Sect. 3, we extend these to two-dimensional arrays. We discuss an extended example, iterative solvers, in Sect. 4. In Sect. 5, we conclude with some final remarks and a discussion of future work.

2 Mapping Functions for One-Dimensional Arrays

We first consider two mapping functions useful for manipulating one-dimensional arrays in UPC. UPC shared one-dimensional arrays are accessed by a single global offset, and the runtime is responsible for determining the owning thread and the offset on that thread [7]. By contrast, Co-Array Fortran views a shared array as mirrored across the threads of execution, indexed by selecting a thread (using the co-array extensions) and then an offset in the array [11]. This alternate view is useful, for example, in having threads that update only their local portions of the shared array. UPC provides limited support for writing such code with the `upc_forall` loop, with its affinity statement. We present two mapping functions, ThreadView and ReverseThreadView, that support addressing the UPC shared array by a thread identifier and offset pair.

2.1 ThreadView

ThreadView is a formula for accessing a shared array by addressing the offset on a particular thread. The result is a global shared array index. With a fixed block size and using the `THREADS` constant, we have an exact translation from co-array-style addressing by offset on a particular thread to a UPC-style shared array index.

function: index j = ThreadView(thread t, offset i, block size b, thread count P)
note: We use operations `div` and `mod` as described in 6.3.4 of [7], here and throughout.
formula: $j \leftarrow (i \ \texttt{div} \ b) \star P \star b + t \star b + (i \ \texttt{mod} \ b)$

2.2 ReverseThreadView

We provide the reverse map as well: given a global array index, we produce a thread identifier and offset on that thread for a given block size and number of threads.

function: (thread t, offset i) = ReverseThreadView(index j, block size b, thread count P)
formula: $t \leftarrow (j \ \texttt{div} \ b) \ \texttt{mod} \ P$
$i \leftarrow (j \ \texttt{mod} \ b) + ((j \ \texttt{div} \ b) \ \texttt{div} \ P) \star b$

3 Two-Dimensional Layouts: Tiles, Rows, and Blocks

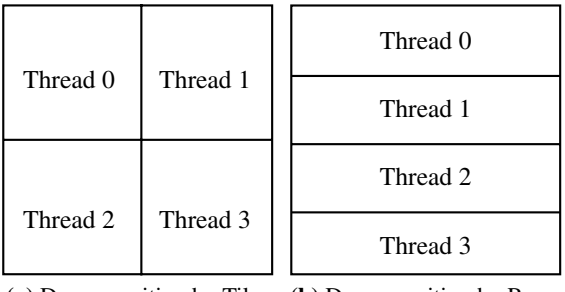
Like ANSI C, UPC supports rectangular, multidimensional arrays [7, 10]. Further, like ANSI C, UPC multidimensional arrays are in fact linear memory regions with automatic linearization of multidimensional coordinates [7, 10]. Thus, we adopt the approach of providing the multidimensional interface in our mapping functions, and assume an underlying one-dimensional data structure. This allows us to leverage our one-dimensional mapping functions and operations. We now present two application-driven layouts, and then a generalization that encompasses them both.

3.1 Motivation

The textbook example of a parallel algorithm (see, for example, [5]) is that of iteratively solving a differential or difference equation on a rectangular region. With fixed

boundary values, and assuming convergence, the problem can be solved numerically by iteratively updating from a template, where each location's value depends on its previous value and those of its neighbors. This update step can be parallelized, and the best static decomposition for minimizing the communication-to-computation ratio is to divide the array into square or nearly-square submatrices, assigning one to each processor. Similar block structures are observed in algorithms from linear algebra [3], and support for these has been a question in UPC discussion circles.

Consider next the problem of computing a vector $\mathbf{x}$ of N elements from some source M-element vector $\mathbf{y}$ and an $N \times M$ array A. If we use the "owner-writes" heuristic for decomposition, we would assign the computation of x_i, the i^{th} element of $\mathbf{x}$, to the thread with affinity to x_i. To minimize remote access, we would also assign affinity of row i of A to that same thread. This decomposition, shown in Fig. 1(b), leads to a natural decomposition of $\mathbf{x}$ into blocks of size N/P, and A into sets of N/P rows, where P is the number of threads.



(a) Decomposition by Tiles **(b)** Decomposition by Rows

Fig. 1. Example of two-dimensional decompositions for four threads of execution. Image is a logical view of the affinity regions for decomposition: (a) by tiles (b) by rows.

3.2 A Generalization: Decomposition by Rectangular Blocks

The above decompositions are special cases of a more general pattern: decomposition of a rectangular array into regular rectangular subarrays. Consider an $N \times M$ array A and $P = Q \times R$ threads of execution. We envision the threads assigned to processors in a mesh layout, so that the threads are arranged into Q rows of R threads each. We divide A into P submatrices, $A_{q,r}$, of size $N/Q \times M/R$, arranged in the same layout as the threads so that $A_{q,r}$ is assigned to the thread in row q, column r of our mesh layout. Requiring $M = N$ and $Q = R$ gives our mapping by tiles, and requiring $R = 1$ gives our mapping by rows. An example is shown in Fig. 2. We assume throughout that Q divides N and R divides M evenly.

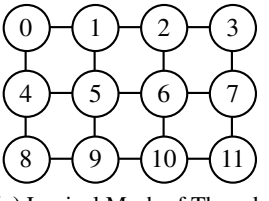
BlockView. We provide the forward mapping of a two-dimensional array index pair (i, j) mapped to a thread identifier t and an offset d.

function: (thread t, offset d) = BlockView(row i, column j, row count N, column count M, processor row count Q, processor column count R)
formula: $t \leftarrow (i \texttt{ div } (N/Q)) * R + j \texttt{ div } (M/R)$
$d \leftarrow (i \bmod N/Q) * M/R + j \bmod M/R$

ReverseBlockView. We now provide the reverse map, starting from a thread identifier t and offset d and expanding back to global two-dimensional array indices i and j.

function: row i, column j = ReverseBlockView(thread t, offset d, row count N, column count M, processor row count Q, processor column count R)
formula: $i \leftarrow (t \texttt{ div } R) * N/Q + d \texttt{ div } M/R$
$j \leftarrow (t \bmod R) * M/R + d \bmod M/R$

We observe that, as an added benefit of the ReverseBlockView map, we could provide an interface, **ThreadBlock**, that treats the subarray with affinity to a particular thread as a private two-dimensional array.



(a) Logical Mesh of Threads

Thread 0	Thread 1	Thread 2	Thread 3
Thread 4	Thread 5	Thread 6	Thread 7
Thread 8	Thread 9	Thread 10	Thread 11

(b) Decomposition by Blocks

Fig. 2. (a) We view the threads of execution arranged logically as a mesh of threads. In our figure, we assume twelve threads of execution, arranged in a 3×4 mesh. (b) We decompose the array into rectangular affinity regions, arranged in the same pattern.

4 An Application to Iterative Solvers

Returning to our example of an iterative equation solver, we consider how to better support this application and use the affinity regions of shared memory in UPC. At a high level, the serial algorithm is:

1. Compute an error term and check whether it is within tolerance limits.
2. If the solution is not within tolerance, update each entry using the previous values of the location and its neighbors. Go to the first step.

Considerations for any parallel solution are:

- Deterministic Solution: The first version may be developed in serial code, which is then parallelized. A deterministic parallel solution will not rearrange computation, and hence will agree with the serial code.
- Convergence Properties: There exist several patterns for iterating through the values in the array. Such choices affect the convergence properties of the solution.
- Communication of Boundary Values: Reduction operations can be used to determine whether the desired tolerance has been achieved. However, the decomposition of the array for parallel update operations can lead to boundary values that must be exchanged with neighboring threads.

– Simplicity: The algorithm is iterative, and thus a performance concern is to make the innermost part of the loop as simple as possible.

We will not address the first two directly, but will instead focus on the latter two concerns. We first ask whether our BlockView and related mapping functions are sufficient. The iterative update could be written as:

```
For each row r to update in my block
    For each column c to update in my block
        Update block using ThreadBlock and BlockView
```

Given that BlockView will properly map requests that fall off the edges of the thread's affinity region, we have a working solution. However, this solution has the undesirable properties of not being deterministic and using fine-grained accesses to neighboring affinity regions. We now present a solution that will make clear that the neighboring values come from the previous iteration and that uses a global "ghost cell exchange" collective communication operation that allows for both synchronization only as needed for the collective and the potential for aggregation of data requests. We will assume throughout that the template used for the update access neighboring positions that differ in at most one unit in each coordinate, allowing for an eight-point stencil. The generalization to wider stencils follows immediately.

4.1 Overview of a UPC Solution

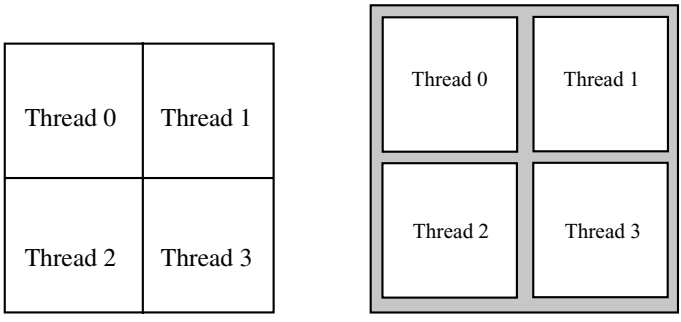
We propose the following for support of ghost cell communication in UPC. We provide the programmer with a logical view of an $N \times M$ array, partitioned among $P = Q \times R$ processors in regular-sized blocks so that the processors and affinity regions are arranged as a $Q \times R$ mesh. We then provide mapping functions that enforce this view of the data region, allow for access to a thread's block and ghost cells for the iteration, and a collective operation for exchanging ghost cells at the start of each iteration.

We first allocate a region of memory that holds $N \times M + (2N/Q + 2M/R + 4) \times P$ items. This adds $2N \times R + 2M \times Q + 4R \times Q$ items of storage overhead. We view this as a two-dimensional array divided into $(N/Q + 2) \times (M/R + 2)$-sized blocks arranged as for the BlockView. Each thread's affinity region now space for the array plus the ghost cell region. This is illustrated in Fig. 3.

We consider the following building blocks:

– GCBlockView: A mapping function that translates a global index pair (i, j) to mask the interleaved ghost cell region. This is a wrapper for BlockView.
– ReverseGCBlockView: The reverse map to GCBlockView that post-processes the result of ReverseBlockView to include the ghost cell region.
– ThreadGCBlock: A wrapper for ThreadBlock that provides a thread to iterate over its logical piece of the array in a straightforward manner.
– GCExchange: Collective communication to exchange boundary values.

The parallel algorithm first allocates a new array of size $N \times N + (2N/Q + 2M/R + 4) \times P$, places data into it with GCBlockView, then loops:



(a) Logical View of Blocked Array **(b)** Memory Allocation with Added Buffers

Fig. 3. Overview of Solution for Iterative Solver. (a) We continue to present a blocked view of the array to the application through the GCBlockView interface. (b) We allocate additional memory in each thread's affinity region to construct a boundary for storing ghost cell data and accessing these via the ThreadGCBlock mapping function.

1. Using proposed collective operations such as `upc_all_reduce` [14], collect the error estimates from each thread for a global error estimate and return this to the threads. If the error estimate is within tolerance, stop.
2. All threads execute GCExchange and prepare the array for the update step.
3. Each thread iterates over its affinity region (regions of the array for which a thread is responsible) using two nested `for` loops, using ThreadGCBlock to read and write its data elements, to compute the next iteration of updates.

The inner loop of the code is tight as the use of the mapping functions avoids the need for conditionals or special packing and unpacking – this is done only once at the start of the algorithm. GCExchange separates communication and computation steps, making the communication step explicit for potential optimization.

5 Conclusion

We presented here a first step toward an application support layer with support for various dense matrix decomposition strategies and a detailed discussion of support for a particular class of algorithms, iterative solvers on rectangular domains with fixed boundary conditions. Future work will include several additional useful support functions, as well as performance measures of the gain from using these functions. Based on work in [6], we developed a physically-based matrix decomposition scheme, allowing a more problem-directed decomposition of dense matrices. We have included these in [2]. Similar efforts are needed to expand support for other scientific and parallel algorithm building blocks in UPC, similar to the wide range of projects to provide application support and communication encapsulation for MPI. As we have seen here, the built-in features of UPC for data locality management are limited, but they can be used to construct more useful and elaborate data locality management tools. Our results are an example of these. UPC today is similar to MPI without its various support libraries and frameworks. A good support library will allow for broader acceptance of UPC.

References

1. Erik Bowman, Karen Devine, Robert Heaphy, Bruce Hendrickson, William F. Mitchell, Matthew St. John, and Courtenay Vaughan. *Zoltan: Data Management Services for Parallel Applications: User's Guide*. Sandia National Laboratories, 1.54 Edition.
2. Jonathan Leighton Brown and Zhaofang Wen. An Application Support Layer for Unified Parallel C. Technical report, Sandia National Laboratories, 2004.
3. Stoer R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 2002.
4. F. Cantonnet, T. El-Ghazawi, P. Lorenz, and J. Gaber. Fast Address Translation Techniques for Distributed Shared Memory Compilers. In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2005.
5. David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.
6. Carter Edwards, Po Geng, Abani Patra, and Robert van de Geijn. Parallel Matrix Distributions: Have we been doing it all wrong? Technical report, The University of Texas at Austin, 1995.
7. Tarek A. El-Ghazawi, William W. Carlson, and Jesse M. Draper. UPC Language Specification v1.1.1., 2003.
8. Michael Heroux and James Willenberg. Trilinos Users Guide. Technical report, Sandia National Laboratories, 2003.
9. George Karypis. Multilevel algorithms for multiconstraint hypergraph partitioning. Technical report, University of Minnesota, 1999.
10. Brian Kernighan and Dennis Ritchie. *C Programming Lanuage*. Prentice Hall, 2nd edition, 1988.
11. R. Numrich and J. Reid. Co-Array Fortran for Parallel Programming. *ACM Fortran Forum*, 17(2):1-31, 1998.
12. Robert C. Oehmke. *High Performance Dynamic Array Structures*. PhD thesis, University of Michigan, Ann Arbor, MI, September 2004.
13. Robert van de Geijn. *Using PLAPACK*. MIT Press, 1997.
14. Elizabeth Wiebel, David Greenberg, and Steve Seidel. UPC Collective Operations Specification v1.0, 2003.
15. Katherine Yelick, L. Semenzato, G. Pike, C. Myamoto, B. Liblit, A. Krishnamurthy, P. N. Hilfinger, S. L. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A High Performance Java Dialect. *Concurrency: Practice and Experience*, 10:11-13, 1998.

Simple, List-Based Parallel Programming with Transparent Load Balancing

Jorge Buenabad-Chávez[1], Miguel A. Castro-García[1,2,⋆], and Graciela Román-Alonso[2]

[1] Sección de Computación, Centro de Investigación y de Estudios Avanzados del IPN, Ap. Postal 14-740, D.F. 07360, México
jbuenabad@cs.cinvestav.mx

[2] Departamento de Ing. Eléctrica, Universidad Autónoma Metropolitana, Izt., Ap. Postal 55-534, D.F. 09340, México
{mcas, grac}@xanum.uam.mx

Abstract. We present a data-list management library that both simplifies parallel programming and balances the workload transparently to the programmer. We present its use with an application that dynamically generates data, such as those based on searching trees. Under these applications, processing data can unpredictably generate new data to process. Without load balancing, these applications are most likely to imbalance the workload across processing nodes resulting in poor performance. We present experimental results on the performance of our library using a Linux PC cluster.

1 Introduction

Today the ubiquity of PC clusters and the availability of a number of parallel programming libraries have made the use of parallelism ready to hand. Programming with *message passing* or *shared memory* is still widely used because of its general applicability, despite the need for the programmer both to divide the workload, to assign it to the processors and to specify communication for processors to share data or to synchronise their tasks [1, 6, 8, 11]. *Shared abstract data-types* (SADTs) are shared objects (such as queues) that appear local to each processor; concurrent access to them is transparent to the programmer through SADTs relevant functions (such as enqueue and dequeue) [7]. Hence the programmer does not need to divide and assign the workload, nor to specify communication. However, the programmer is responsible to represent the application data as one or more of the shared object(s) available.

Skeletons are pre-packaged parallel algorithmic forms, or parallel code pattern functions. The programmer has only to assemble the appropriate skeletons to solve the problem at hand. However, not all practical problems can be simply represented with the skeletons proposed so far, and some skeletons require a relatively large number of parameters complicating their use [4, 5, 10].

⋆ Thanks to CONACyT for the institutional support.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 920–927, 2006.

In this paper we present a data list management library (DLML) for parallel programming in clusters. DLML is useful to develop parallel applications whose data can be organised into a list, and the processing of each list item does not depend on other list items. The programming is almost sequential and, at run time, load balancing takes place transparently to the programmer. DLML lists are similar to SADTs queues [7] both in organisation and use (we describe DLML use in this paper). SADTs queues follow the FIFO policy on inserting/deleting items, DLML follows no order. The motivation of DLML was to provide transparent load balancing to parallel applications running on clusters. SADTs queues were designed first for parallel programming on shared memory platforms, and later for distributed memory platforms. For the latter, load balancing was also integrated.

An earlier version of DLML was designed to help to improve the execution time of a parallel evolutionary algorithm [3]. Another version was tested with a divide-and-conquer algorithm [9]. These two kinds of algorithms manage a total amount of data that is known in advance. The version presented in this paper is simpler to use, uses a more efficient load balancing policy based on distributed workpools, and can be used with applications that dynamically generate data in unpredictable ways.

With our library, the data list must be accessed using typical list functions, such as *get* and *insert*, provided by our environment. Although the use of these functions is typical, internally they operate on several distributed lists, one in each processing node. When a processor finds its local list empty, more data is asked from remote processing nodes, thus balancing the workload dynamically and transparently to the programmer.

In Section 2 we present the Non-Attacking Queens problem as an example of applications that dynamically generate data. In Section 3 we show this application using our data list-management library. In Section 4 we present the organisation of the load balancing policy of our library. In Section 5 we discuss empirical results on the performance advantage of using our programming environment. We offer some conclusions and describe future work in Section 6.

2 The Non-attacking Queens Problem

In this section we present the Non-Attacking Queens (NAQ) problem [2] as a sample application that dynamically generates data organised into a list. It consists of finding all the possibilities of placing N queens on an $N \times N$ chessboard, so that no queen attacks any other. The search space of NAQ can be modeled with an N degree searching tree. We implemented a looping version of NAQ managing such tree as a list, which, at any time, contains only leaves that represent possible solutions to explore. For $N = 4$, Figure 1 shows the only two solutions (left), the search space (middle) and part of its list management trace (right) by our list-management library.

A queen position can be specified with two coordinates in a two-dimensional array. However, in our implementation, a queen placed on row i ($1 <= i <= N$)

is represented by i itself, while the column it is placed in is given by the vector element $chessboard[i]$. A list item B is a structure with two information fields: B.queen is the number of queen to be placed, and B.chessboard is an N-size vector with the position of the queens placed so far.

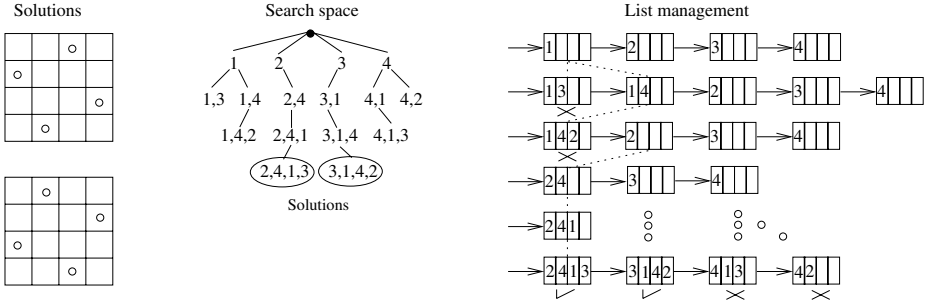


Fig. 1. NAQ: solutions, search space and our list representation for $N = 4$

In our sequential code below, functions `Create_element()`, `Insert()` and `Get()` are used to manage the list. Function `Attacked(B.queen, column, B.chessboard)` returns *True* if B.queen can be placed in column, taking into account previously placed queens in B.chessboard; otherwise returns *False*. An element taken from the list with `Get()` is discarded but possibly generates new elements to insert into the list. The algorithm finishes when the list becomes empty. (`Get()` function returns False).

```
function Non-attacking-queens(List *L) {
  queen = 1; chessboard[1<=i<=N] = 0; nr_solutions = 0;
  q = Create_Element(queen,chessboard);     //Initial element to
                                            //place the queen 1
  Insert(&L, q);
  while( Get(&L, &B) ) {
    for(column=1;column <= N; column++ ) {  // generate elements
      if (! Attacked(B.queen,column,B.chessboard) )
        if (B.queen < N) {                  // where B.queen
          B.chessboard[B.queen] = column;   // is not attacked
          q = Create_Element(B.queen+1,B.chessboard);
          Insert(&L,q);
        } else
          nr_solutions= nr_solutions +1;    //one more solution
    }                                       // N-queens have been placed
  }
  return nr_solutions;
}
```

3 Integrating DLML

In this section we show how to modify the sequential version of NAQ shown above to use our data list-management library, or DLML. The DLML version runs in parallel, under the SPMD (Simple Program Multiple Data) model, i.e., a data list is managed in each node running the application. The DLML version also runs with transparent dynamic load balancing.

The code below shows the DLML version, wherein the data list functions are those of DLML. Note that, under the SPMD model, the functions `DLML_Create_First_Element()` and `DLML_Insert_First()` will be called in each node; their implementation is, however, such that they will return immediately in all nodes except in the master node (0), where it is actually executed.

`DLML_Get()` tries to get a data element from the local list, or a remote list if that one is empty, into its second parameter. It returns *False* if no data element is found locally and remotely; otherwise return *True*.

`DLML_Reduce_Add()` will also be called in each node, but is only actually executed by the master processor. It is in charge of gathering and adding the partial results in each node.

```
function Non-attacking-queens(List *L) {
  queen = 1; chessboard[1<=i<=N] = 0; nr_solutions = 0;
  q = DLML_Create_First_Element(queen,chessboard);

  DLML_Insert_First(&L, q);
  while( DLML_Get(&L, &B) ) {
    for(... ) {
      ...
      q= DLML_Create_Element(B.queen+1,B.chessboard);
      DLML_Insert(&L,q);
      ...
    }
  }
  final_result= DLML_Reduce_Add( nr_solutions );
  return final_result;
}
```

4 DLML Architecture

At run time the DLML architecture consists of one *application process* and a load balancing *DLML* process on each node running a parallel application, see Figure 2. These processes communicate through message-passing; DLML processes in different nodes communicate likewise. They follow a protocol with the types of messages shown in that figure: *Local data, Insert data, Get data, Request node info, Node info, Request data, Remote data*, and *No items*.

A data list is managed in each node. When an application process tries to get a data element and its local list is not empty, an element therein is given to the application. When the list is empty, the peer DLML process is activated

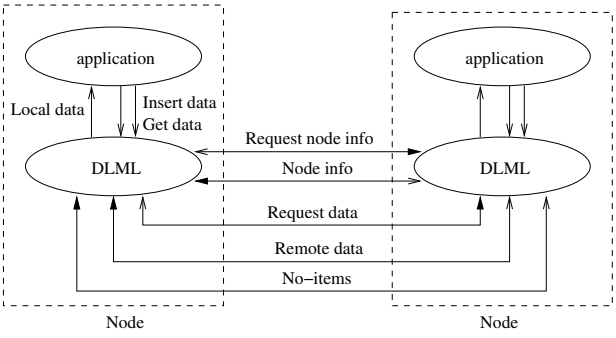


Fig. 2. Load balance protocol messages

to ask remote DLML processes more data, following a bidding policy [12]; i.e., asking all the length of their list, and selecting the node with the greatest list length to ask it more data.

All *request data* messages received by a node are stored in a queue. To satisfy these requests, the data list is evenly divided into the total number of requests plus one to count in that node, thus trying to balance the workload. If the number of requests exceeds the data list length L, only $L - 1$ requests are satisfied. If the list becomes empty before satisfying the *requests*, the response message *no items* is sent, and the DLML process asking for data starts again the bidding policy.

5 Performance Evaluation

To evaluate the performance of DLML, we ran the sequential version and the DLML version of the non-attacking queens (NAQ) problem (shown in Sections 2 and 3), for different problem sizes: chessboards of size $N \times N$, for $8 \leq N \leq 17$. The sequential version was run on an Intel Pentium IV 3.4 GHz processor with 2 GB RAM memory under Linux. The DLML version (implemented with MPI), was run on a dedicated, 17-node homogeneous PC cluster, each node configured with: 1 Pentium IV 2.8 GHz processor, 512 MB RAM memory and Linux. All nodes are interconnected through a Fast Ethernet 3COM switch.

5.1 Sequential vs DLML

Figure 3 shows the execution time of sequential NAQ (SEQ) and DLML-NAQ (DLML). The x axis shows the size of the problem N, and the y axis shows execution time in seconds in log_{10} scale. For small problem sizes (not shown in that figure), SEQ performs better than DLML. Partly, this is because the sequential version runs on a faster processor. But also because, in the parallel version, processors asks for data throughout the run, and some requests arrive after the master processor has finished processing all the data.

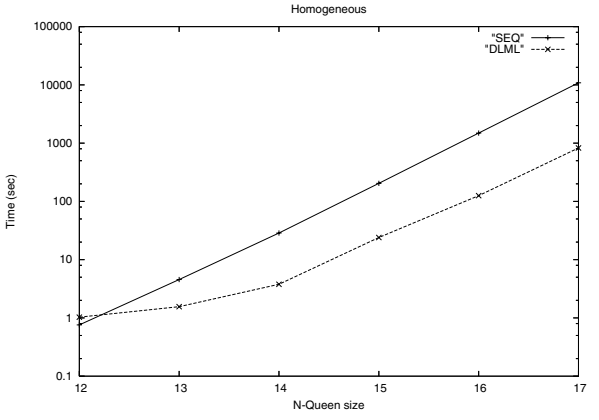


Fig. 3. NA-Queens SEQ vs DLML

DLML performs better than SEQ for problems of size $N \geq 13$ because the number of *possible* solutions explored was much greater, and parallelism did help in these runs. For $N = 12$, there are 14,200 solutions. For N equal to 13 and 17, there are 73,712 and 95,815,104 solutions, respectively. For $N = 17$, the execution time in minutes for SEQ was 180.6 and for DLML 13.7.

5.2 DLML vs DLML Without Load Balancing

To evaluate the performance of the load balancing strategy of DLML, we also run the DLML version but with DLML load balancing policy deactivated. We refer to this version as DLML-NLB (for no load balancing).

Before comparing DLML vs. DLML-NLB, we describe some aspects of their operation. Under DLML-NLB the master node starts processing data. When *all* data requests from other nodes arrive, the master partitions and distributes data evenly among all nodes. However, the processing of each data item *can*, or *cannot*, generate more data items to explore. Thus some processors may do more work than others. In DLML, there is no initial data partitioning; processors ask for work throughout the run when they find their list empty.

Figure 4 shows the execution time of DLML and DLML-NLB. The x axis shows the size of the problem N, and the y axis shows execution time in seconds in log_{10} scale. All times correspond to runs on 17 processors. DLML shows better performance in all runs because of its load balancing policy. For $N = 17$, the execution time in minutes for each version was 46.2 and 13.7, respectively.

Figure 5 shows the amount of data items processed by each processor under both DLML (shaded boxes) and DLML-NLB (on 17 processors), for $N = 17$. The x axis plots the identification number of each processor, and the y axis plots the number of data items processed by each processor in millions. In DLML, despite the individual dynamic data generation of each processor, work was distributed more evenly. In DLML-NLB the amount of work was not nearly evenly distributed among the processors.

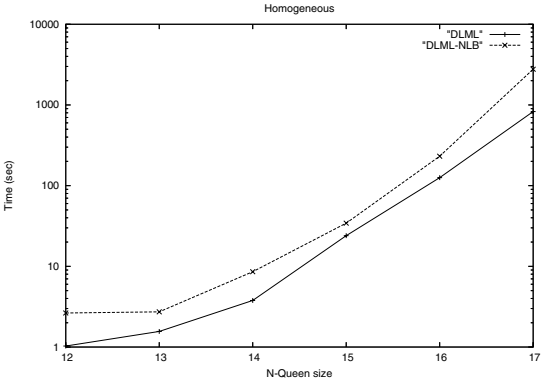


Fig. 4. Execution time under DLML vs. DLML-NLB, for different problem sizes, $12 \leq N \leq 17$, on homogeneous nodes

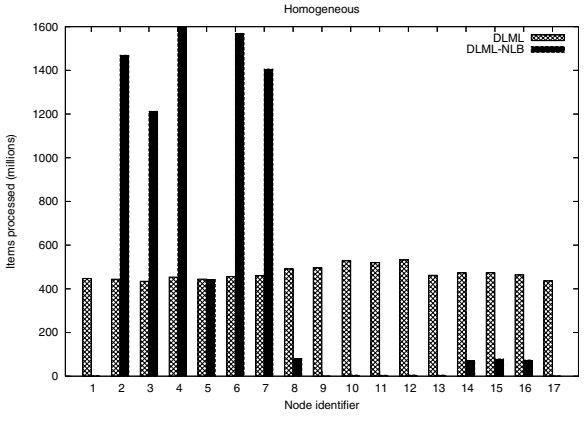


Fig. 5. Load distribution by processor for $N = 17$ queens, under DLML and DLML-NLB both on 17 homogeneous nodes

6 Conclusions and Future Work

We have presented DLML a list-management library that simplifies the programming of applications whose data can be organised into a list. Programming under DLML is almost sequential, but the execution is parallel and with transparent, dynamic load balancing. Using the non-attacking queens problem as sample application that dynamically generates data, we showed the ease of use of DLML and also its performance benefit on a homogeneous PC cluster.

We are currently improving the design of the DLML interface to include more list manipulation functions other than the ones presented here (insert, get and reduce). We are also improving both the communication between the application

process and the DLML process in each node through shared memory, and the load balancing policy to make it more scalable.

References

1. Christiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, and Willy Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.
2. Aiden Bruen and R. Dixon. The n-queens problem. *Discrete Mathematics*, 12(4):393–395, 1975.
3. Miguel A. Castro-García, Graciela Román-Alonso, Jorge Buenabad-Chávez, Alma E. Martínez-Licona, and John Goddard-Close. Integration of load balancing into a parallel evolutionary algorithm. In *Advanced Distributed Systems*, volume 3061 of *Lecture Notes in Computer Science*, pages 219–230. Springer-Verlag, January 2004.
4. Murray I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. MIT Press, 1989.
5. R. Di Cosmo, Z. Li, S. Pelagatti, and P. Weis. Skeletal parallel programming with ocamlp3l 2.0. In *Third International Workshop on High-level Parallel Programming and Applications (HLPP 2005)*, July 2005.
6. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*. MIT Press, first edition, December 1994.
7. D.M. Goodeve, S.A. Dobson, J.M. Nash, J.R. Davy, P.M. Dew, M. Kara, and C.P. Wadsworth. Towards a model for shared data abstraction with performance. *Journal of Parallel and Distributed Computing*, 49(1):156–167, February 1998.
8. Jean-Louis Roch, Thierry Gautier, and R'emi Revire. Athapascan: an api for asynchronous parallel programming. Technical report, INRIA RT-0276, 2003.
9. Graciela Román-Alonso, Miguel A. Castro-García, and Jorge Buenabad-Chávez. Easing message-passing parallel programming through a data balancing service. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 3241 of *Lecture Notes in Computer Science*, pages 295–302. Springer-Verlag, September 2004.
10. David B. Skillicorn and Domenico Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, June 1998.
11. Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference, The MPI Core*, volume 1. MIT Press, second edition, August 1998.
12. J. A. Stankovich and I.S Sidhu. An adaptive bidding algorithm for processes, clusters, and distributed ups. In *Inter. Conf. on Distributed Computing Systems*, pages 49–59. IEEE, 1984.

SILC: A Flexible and Environment-Independent Interface for Matrix Computation Libraries

Tamito Kajiyama[1,2], Akira Nukada[1,2], Hidehiko Hasegawa[1,3], Reiji Suda[1,2], and Akira Nishida[1,2]

[1] CREST, Japan Science and Technology Agency, Saitama 332–0012, Japan
[2] The University of Tokyo, Tokyo 113–8656, Japan
{kajiyama, nukada, reiji, nishida}@is.s.u-tokyo.ac.jp
[3] University of Tsukuba, Tsukuba 305–8550, Japan
hasegawa@slis.tsukuba.ac.jp

Abstract. We propose a new framework, named Simple Interface for Library Collections (SILC), that gives users access to matrix computation libraries in a flexible and environment-independent manner. SILC achieves source-level independence between user programs and libraries by (1) separating a function call into data transfer and a request for computation, (2) requesting the computation by means of mathematical expressions in the form of text, and (3) using a separate memory space to carry out library functions independently of the user programs. Using SILC, users can easily access various libraries without any modification of the user programs. This paper describes the design and implementation of SILC based on a client-server architecture, and presents some experimental results on the performance of the implemented system in different computing environments.

1 Introduction

Solutions of systems of linear equations and other matrix computations take a major proportion of execution time and memory resources in many large-scale scientific applications. As a result, a large number of matrix computation libraries have been developed [1, 2, 3] to facilitate the rapid development of user programs. Each library offers a different set of solvers and matrix storage formats and has its own application programming interface that is incompatible with other libraries.

The traditional way to use matrix computation libraries, i.e., through function calls, makes user programs dependent on the libraries. Users of a library have to prepare input matrices in a specific storage format and to make a function call using a library-specific function name together with a number of arguments in a prescribed order. Although such function calls are plain and intuitive, they result in source-level dependency on the library, making it difficult to replace one library with another.

There are various computing environments, such as personal computers, symmetric multiprocessor (SMP) systems, high-end supercomputers, and clusters,

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 928–935, 2006.

each of which has its own highly optimized libraries. Therefore, users must modify their user programs significantly in order to port them from one computing environment to another. Moreover, there are various solvers and matrix storage formats, the best combination of which varies according to the computing environment in use and the problem to be solved. However, because of the source-level dependency resulting from the use of a solver and matrix storage format in the form of conventional function calls, users who want to find the most efficient solver and matrix storage format must make considerable modifications in the user programs to change solvers and matrix storage formats.

To address those issues that arise from the source-level dependency on specific libraries, we propose a new framework that allows users to easily utilize matrix computation libraries in a flexible and computing environment-independent manner. The framework, named Simple Interface for Library Collections (SILC), is based on the following three design decisions.

- To separate a function call into data transfer (to and from a separate memory space) and a request for computation.
- To request the computation by means of mathematical expressions in the form of text.
- To use the separate memory space to carry out library functions independently of user programs.

In our framework, a function call occurs through three steps: sending input data (i.e., arguments), requesting computation by means of mathematical expressions, and receiving the results of the computation. The operators that comprise the expressions are translated into a series of function calls and are carried out in a separate memory space. The results of the computation are sent back only when they are required by user programs.

The main benefits of employing SILC are as follows.

- User programs will be free of source-level dependency on specific libraries, so that users won't need to modify their programs when changing libraries according to the computing environment and the problem to be solved.
- Users need to prepare only the smallest amount of data. Temporary memory space used for carrying out library functions is automatically allocated before the library functions are called.
- A variety of computing environments and programming languages can be used, since computation is requested by means of mathematical expressions in the form of text.

2 Design and Implementation

We have been developing a SILC system for shared-memory parallel computing environments. Figure 1 shows an architectural overview of the SILC system. It is based on a client-server architecture. The SILC server and user programs can run either on the same machine or on different machines on a network.

A user program connects to the SILC server and utilizes the features of matrix computation libraries by sending three types of requests described below.

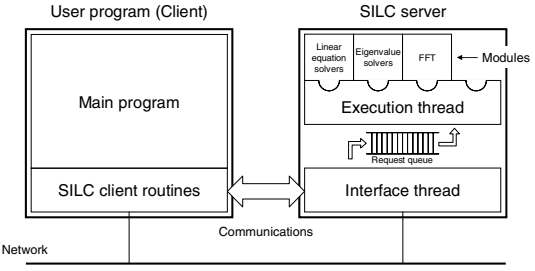


Fig. 1. Architectural overview of SILC

PUT deposits data such as matrices and vectors, together with names for later reference. The data are converted and transferred into the server's memory space, which is independent of the user program. The deposited data remain there until deleted explicitly.

EXEC requests computation by means of mathematical expressions in the form of text. The names defined by preceding PUT requests are used in the expressions to refer to the deposited data. The computation is carried out on the server asynchronously. The results of the computation and names defined by the expressions are retained in the server's memory space.

GET fetches data from the server. Names are used to specify the data to be fetched, and those data are sent back into the memory space of the user program. The data are kept undeleted on the server.

In case a user program is written in C, the following three client routines are used to issue the PUT, EXEC, and GET requests, respectively:

- `SILC_PUT(`$\langle name \rangle$`,` $\langle data \rangle$`)`
- `SILC_EXEC(`$\langle expr \rangle$`)`
- `SILC_GET(`$\langle data \rangle$`,` $\langle name \rangle$`)`

where $\langle name \rangle$ is a data name and $\langle expr \rangle$ is a mathematical expression (whose syntax is described later), each specified by a string. $\langle data \rangle$ is a pointer to the `silc_envelope_t` structure that is used for data communications between the user program and the SILC server.

The requests from the user program are received by the interface thread in the SILC server. PUT and GET requests are handled by the interface thread, while EXEC requests are stored in the request queue and processed by the execution thread one after another. The user program and the interface thread run synchronously, while the execution thread runs asynchronously.

Figure 2 (a) shows a user program that calls a library function `ssi_cg` [3] to solve a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ with the CG method [4]. The input data of the library function are matrix A and vector $\boldsymbol{b}$ as well as some solver-specific parameters, while the output is the solution $\boldsymbol{x}$. These data are represented by library-specific data structures and are passed to the library function through its arguments in a prescribed order. On the other hand, in the

```
SSI_MATRIX A;
SSI_SCALAR *b, *x, work[N*6], params[2];
int options[6], status;

/* Create matrix A and vectors b and x */
status = ssi_cg(b, x, work,
                params, options, &A, NULL);
```

(a)

```
silc_envelope_t A, b, x;

/* Create matrix A and vectors b and x */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b"); /* Call a solver */
SILC_GET(&x, "x");
```

(b)

Fig. 2. Comparison between the two ways of using a library function. (a) is a user program that makes use of a library function in the traditional manner. (b) is another user program written in the framework of SILC.

framework of SILC, the same computation can be achieved as shown in Fig. 2 (b). In this framework, the input data are deposited by two separate calls of the client routine `SILC_PUT`. After the input data are deposited, computation is requested by the `SILC_EXEC` routine. This routine's argument is a mathematical expression in the form of text that instructs the solution of the system of linear equations using an appropriate library function (e.g., `ssi_cg`). Finally, the output data are fetched by `SILC_GET`. The source code shown in Fig. 2 (b) does not contain any code that is specific to the actual library to be used for the computation.

The SILC system is equipped with a simple command language to represent a request for computation in the form of mathematical expressions. The unit of computation that is carried out at once is a statement, which is either an assignment or a procedure call. The left-hand side of an assignment statement is a variable name, which can be used without declaring a data type. The right-hand side of the assignment statement is an expression, which consists of variable names, operators, and function calls. Some of the operators are binary arithmetic operators (`+`, `-`, `*`, `/`, `%`), solutions of systems of linear equations (e.g., `A \ b` obtains the solution $\boldsymbol{x}$ as in $A\boldsymbol{x} = \boldsymbol{b}$), complex conjugates (`A~`), and conjugate transposes (`A'`). There is no control statement in the command language; loops and conditional branching are achieved by the programming languages in which user programs are written.

Every operator, function, and procedure[1] that appears in a mathematical expression is translated into a call of a library function with the help of a wrapper that actually calls the library function. The wrapper provides a unified interface to the library function so that the SILC server can invoke all library functions in the same manner. Related wrappers are grouped into an arithmetic module, and all modules are dynamically loaded into the SILC server at startup. When loading arithmetic modules, the server constructs a mapping table that relates operators, functions, and procedures to certain wrappers. The server then handles each of the operators, functions, and procedures used in a given mathematical expression by invoking a corresponding wrapper, which results in a call

[1] Functions return values, while procedures do not. Moreover, procedures can change the values of arguments, whereas functions cannot. The distinction between functions and procedures is introduced to eliminate ambiguities from mathematical expressions.

Table 1. Four server configurations used for preliminary experiments

Environment	Specifications	Thread(s)
(a) A notebook PC	CPU: Intel Pentium M 733 1.1 GHz, Memory: 768 MB, OS: Fedora Core 3	–
(b) SGI Altix 3700	CPU: Intel Itanium 2 1.3 GHz × 32, Memory: 32 GB, OS: Red Hat Linux Advanced Server 2.1	1
(c) IBM eServer OpenPower 710	CPU: IBM Power5 1.65 GHz × 2 (4 logical CPUs), Memory: 1 GB, OS: SuSE Linux Enterprise Server 9	4
(d) SGI Altix 3700	Same as (b)	16

of a particular library function. Matrix storage formats are implemented in a similar way, and their wrappers are grouped into a storage format module.

Support for a new matrix storage format can be incorporated into the SILC system by providing a storage format module coupled with an arithmetic module. Both arithmetic and storage format modules are parallelized with OpenMP in shared-memory parallel computing environments.

To assess the performance of SILC servers in different computing environments, we conducted preliminary experiments using the four configurations of SILC servers summarized in Table 1. The three computing environments of the notebook PC, Altix 3700, and OpenPower 710 were interconnected via a 100 Base–TX network. We used the user program shown in Fig. 2 (b), which solves a system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$ using the CG method, where A is an $N \times N$ tridiagonal matrix of double precision in the Compressed Row Storage (CRS) format [4]. An arithmetic module that includes a wrapper for the library function `ssi_cg` was used to solve the system of linear equations. The user program was carried out on the notebook PC for all cases. Since the server to which the user program connects is specified in a configuration file, the same user program was used without any modification during the experiments. Figure 3 shows the results of the experiments, with dimension N on the horizontal axis and execution time in seconds (including communication time for establishing connection and transferring data) on the vertical axis. The experimental results proved that

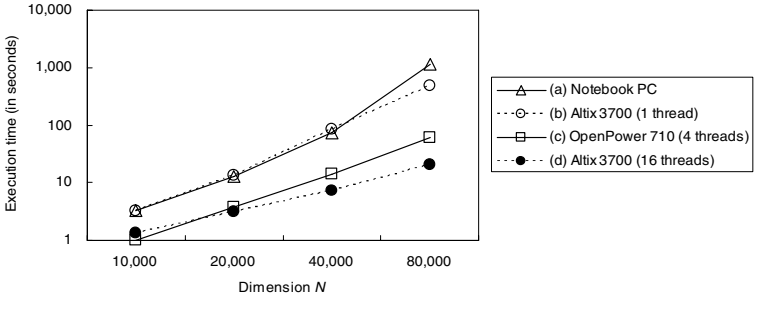


Fig. 3. Experimental results with the four server configurations shown in Table 1

(1) the computing environment that achieves the best performance varies according to dimension N, and (2) the execution times in the cases of (c) and (d) are much shorter than those in the case of (a) even at the cost of depositing matrix A and vector $\boldsymbol{b}$ and fetching the solution $\boldsymbol{x}$ through the relatively slow network. Since communication time is roughly on the order of $O(N)$ while computation time is on the order of $O(N^2)$, the observation (2) agrees with the expectation that the communication time will take a smaller fraction of the whole execution time as dimension N increases.

3 Usability and Benefits

3.1 User Programs

User programs in the framework of SILC are independent of the computing environment in which a SILC server runs. This allows users to develop their programs without being concerned about the details of various computing environments. Users can port their programs from one computing environment to another without any modification to the programs.

SILC separates data transfer from a request for computation, permitting PUT, EXEC, and GET requests to be sent from different user programs separately. Moreover, a SILC server accepts connections from multiple user programs, allowing those programs to use the server as an in-memory database through which they exchange data. Therefore, in the framework of SILC, separate user programs can be easily combined in such a manner that a mesh generation program issues PUT requests, a solution program sends EXEC requests, and a visualization program uses GET requests.

Since data transfer and requests for computation in the form of text are relatively lightweight tasks, less-powerful computing environments, such as personal computers and mobile environments, will suffice to run user programs for SILC. This characteristic allows combinations in which, for example, a personal computer is used to control computations on high-performance parallel computers.

3.2 Libraries

An exchange of libraries is required mainly in either of two situations: (1) users wish to change computing environments or (2) users wish to use different solvers and matrix storage formats provided in other libraries. There are a variety of optimized matrix computation libraries that are only available in a particular computing environment. Moreover, the most efficient solver and matrix storage format depend strongly on the computing environment to be used as well as on the problem to be solved. Aside from the considerable costs of modifying user programs in order to switch libraries, it is burdensome for users to maintain multiple versions of the same program, each of which is written for a specific computing environment and a problem based on a specific combination of a solver and matrix storage format. In the framework of SILC, on the other hand, users can easily change libraries either by using different SILC servers running

in other computing environments or by modifying the mapping table in a SILC server so that different library functions are used.

In addition, a SILC server has a separate memory space, which enables the server to convert deposited matrices into different storage formats independently of user programs. For example, it can be easily achieved that a user program uses the CRS format to deposit and receive matrices, while the server uses the Jogged Diagonal Storage (JDS) format [4] to carry out computations on them. Although conversion of storage formats takes some time and space according to the sizes of the matrices, there are cases where better performance is achieved if the matrices are converted into an appropriate storage format before the computations on them. Moreover, the conversion of storage formats allows users to choose solvers and storage formats from among a wide range of matrix computation libraries.

3.3 Programming Languages

In SILC, computation is requested by means of mathematical expressions in the form of text. In addition, the client routines of SILC to be linked with user programs are small and easy to implement. Therefore, various programming languages can be used to develop user programs. At the moment, three sets of client routines for C, Fortran, and Python are available. The main requirements for a programming language to implement the client routines are the capabilities of numerical computation, text processing, and socket-based interprocess communications; the major programming languages meet these requirements.

SILC permits various combinations of matrix computation libraries and programming languages. For example, user programs written in Python can easily make use of libraries written in C or Fortran in the same way.

4 Related Work

To improve the usability of matrix computation libraries, various approaches have been proposed based on Remote Procedure Call (RPC) [5, 6] and code generation techniques [7, 8, 9]. NetSolve [5] and Ninf–G [6] are middleware for realizing RPC in Grids. In these systems, RPCs are requested in a manner similar to traditional function calls. In contrast, SILC employs simple mathematical expressions to request matrix computations, allowing users to ignore complicated matters between user programs and matrix computation libraries. CMC [8] is a compiler that translates a user-defined function in MATLAB [10] into a subroutine in Fortran 90. CMC provides support for several sparse matrix storage formats and carries out a variety of source-level optimizations. Both CMC and SILC pursue the same goal of enhancing the utility of matrix computations. Whereas CMC focuses on the generation of optimized Fortran subroutines from MATLAB functions, SILC focuses on the use of various matrix computation libraries in a language-independent manner.

5 Concluding Remarks

The traditional manner of using matrix computation libraries through function calls usually results in source-level dependency on the libraries, making it impossible to switch libraries without modifying user programs. To address this issue, we have proposed a new framework for using matrix computation libraries in a flexible and environment-independent manner. In this paper, we presented the design and implementation of the proposed framework for shared-memory parallel computing environments based on a client-server architecture. We also reported the results of preliminary experiments assessing the performance of the implemented system, and discussed the usability and benefits of our proposal.

We plan to provide modules for major matrix computation libraries, allowing users to easily switch libraries and compare the performance of user programs with respect to different solvers and matrix storage formats. Implementation of a scripting language for SILC, run-time optimization of mathematical expressions, and MPI-based parallelization of the SILC system are also in our future plans.

Acknowledgments

This research was supported by a grant from the Core Research for Evolutional Science and Technology (CREST) of the Japan Science and Technology Agency [11].

References

1. Dongarra, J.: Freely available software for linear algebra on the Web (2004) http://www.netlib.org/utk/people/JackDongarra/la-sw.html.
2. Wu, K., Milne, B.: A survey of packages for large linear systems. Technical Report LBNL–45446, Lawrence Berkeley National Laboratory (2000)
3. Kotakemori, H., Hasegawa, H., Kajiyama, T., Nukada, A., Suda, R., Nishida, A.: Performance evaluation of parallel sparse matrix–vector products on SGI Altix3700. In: First International Workshop on OpenMP (IWOMP 2005). (to appear)
4. Barrett, R., *et al.*: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM (1994)
5. NetSolve: http://icl.cs.utk.edu/netsolve/ (2005)
6. Ninf Project: http://ninf.apgrid.org/ (2005)
7. De Rose, L., Padua, D.: Techniques for the translation of MATLAB programs into Fortran 90. ACM TOPLAS **21** (1999) 286–323
8. Kawabata, H., Suzuki, M., Kitamura, T.: A MATLAB-based code generator for sparse matrix computations. In: APLAS 2004, LNCS 3302. (2004) 280–295
9. Kennedy, K., *et al.*: Telescoping languages: A system for automatic generation of domain languages. Proceedings of the IEEE **93** (2005) 387–408
10. The MathWorks, Inc.: http://www.mathworks.com/ (2005)
11. Nishida, A.: SSI: Overview of simulation software infrastructure for large scale scientific applications. In: IPSJ SIG Notes, 2004–HPC–098. (2004) 25–30
12. Luszczek, P., Dongarra, J.: Design of interactive environment for numerically intensive parallel linear algebra calculations. In: ICCS 2004, LNCS 3039. (2004) 270–277

A Fortran Evolution of mpC Parallel Programming Language

Alexey Kalinov, Ilya Ledovskikh, Mikhail Posypkin,
Zakhar Levchenko, and Vladimir Chizhov

Institute for System Programming of the Russian Academy of Sciences,
109004, 25 B.Kommunisticheskaya str. Moscow, Russia

Abstract. mpF is a new parallel extension of Fortran 90. It was developed on base of experience of development and use of the mpC parallel programming language. The paper compares programming models of mpC and mpF.

1 Introduction

Network of computers is the most common architecture for parallel computing. In general it is a set of diverse computers interconnected via diverse communication equipment. For programming high performance computations on such heterogeneous networks the mpC parallel programming language and its programming environment was developed in the Institute for System Programming, Russian Academy of Sciences in the end of 1990s under the direction of A. Lastovetsky [1, 2]. mpC is a parallel extension of C with explicit parallelism. The main idea underlying mpC is that an mpC application explicitly defines an abstract network and distributes data, computations and communications over the network. The mpC programming environment uses in run time this information as well as information about any real executing network in order to map the application to the real network in such a way that ensures efficient execution of the application on this real network. A proper choice of the level of constructs and the use of the explicit approach to parallel programming enables mpC to achieve a high expressive power and efficiency, which are characteristic of low-level tools like MPI [3], with the convenience, which is characteristic of high-level languages like HPF [4]. The experience in using mpC [5] showed that the efficiency of mpC programs is comparable with that of such high-quality specialized parallel libraries as ScaLAPACK [6] on homogeneous systems and is greater on heterogeneous computer systems.

Fortran still remains the main programming language for scientific and engineering computations. This encouraged us to develop an mpC like extension of the Fortran called mpF [7]. Computational experiments showed that efficiency of mpF applications is practically the same as of its MPI counterpart [8]. While design the language we took into account experience of using mpC and made some modifications in its concepts. Those modifications are in the focus of this paper. We will describe main concepts of the mpF and show their relations to the concepts of the mpC.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 936–943, 2006.

The rest of the paper is organized as follows. In Section 2, we outline principles of the computing space management. In Section 3, we discuss the distribution of data and computations among regions of the computing space. Section 4 is devoted to the extension of assignment statement. In Section 5, we discuss structure of the program. Section 6 concludes the paper.

2 The Computing Space Management

In mpF, the concept of the computing space is introduced. It is defined as the set of virtual processors (nodes), which can have different performance. It is assumed that all nodes of the computing space are connected with each other. Subsets of the computing space (called regions of the computing space) are used to distribute data, evaluate expressions, and execute statements. The computing space is managed using network objects or simply networks in the way similar to that used for memory management. The only conceptual difference is that a node of an existing network, which is called its parent, must initiate the creation of a new network.

Two intrinsic networks are always available:

- `SPACE` represents the entire computing space;
- `HOST` represents the network consisting of the single node associated with the console.

Network objects are declared in the way similar to Fortran arrays; the user can treat network as array of nodes. The network declaration introduces the name of network and its attributes. The rank or the rank and shape of the network are specified by DIMENSION attribute. The network with the ALLOCATABLE attribute is one for which the computing space is allocated by an ALLOCATE statement. Also two new attributes are introduced in mpF - the POWER attribute specifies performances of the nodes, and the PARENT attribute specifies the parent node. The simplest network declaration is

```
NETWORK SIMPLE(N)
```

It describes network `SIMPLE` consisting of `N` nodes. By default, all nodes have the same performance, and the parent node of this network is `HOST`.

The intrinsic network `SPACE` is treated as if it were declared in a explicit network declaration

```
NETWORK SPACE(TOTAL_NODES())
```

where `TOTAL_NODES()` is the intrinsic function that returns the number of nodes in the computing space. The `SPACE`'s parent is `HOST`.

A more complicated network declaration is

```
NETWORK, DIMENSION(N,M), POWER(POW), &
         PARENT(SIMPLE(K)) :: HETERONET
```

It describes two-dimensional networks consisting of `N`×`M` nodes. Parent of this network is the node of network `SIMPLE` with the coordinate `K`. The relative performance of the node `HETERONET(I,J)` is `POW(I,J)`, where `POW` should be an integer array with attribute `DIMENSION(N,M)`.

As we mentioned above networks are considered as arrays of nodes. One may define sections or elements of the array of nodes called subnetworks:

```
SUBNET(SIMPLE) S1(2:4), S2(1::2)
```

Here, two subnetworks of the one-dimensional network `SIMPLE` are declared: subnetwork `S1` comprising nodes 2-4 and subnetwork `S2` consisting of the nodes with odd indexes.

The statement

```
SUBNET(HETERONET) S3(I,:)
```

declares the subnetwork of the two-dimensional network HETERONET that consists of the `I`-th row of `HETERONET`.

2.1 The Differences from mpC

In mpC computing space is not considered as network. This keeps logical consistency of the program model - we do not allocate network inside another network; but complicates it. We will point below to the simplification followed from considering the computing space as a network.

In mpC the network is not considered as array of nodes. The user has to describe the network type before to declare a network object. The network type declares coordinate variables for node specification. For example the following declarations are needed to declare the `SIMPLE` network in mpC:

```
nettype SimpleNet(N) { coord i=N; };
net SimpleNet(n) SIMPLE;
```

Here `i` is an integer coordinate variable ranging from `0` to `N-1`. The declarations of the subnets `S1` and `S2` in mpC could be declared as follows:

```
subnet [SIMPLE:i>0 && i<4) S1, [SIMPLE:I%2==0] S2;
```

(Note, that in mpC coordinates are ranging from 0 instead 1 in mpF).

Though mpC approach to defining computing space regions is more general than in mpF, but for Fortran users the familiar array syntax is more convenient.

3 The Distribution of Data and the Evaluation of Expressions

The distribution of the data and computations in mpF is conceptually the same as in mpC. A variable that is distributed over a region of the computing space consists of components of the same type: one per node of this region. Data objects can be distributed over a network, a subnetwork, the entire computing space, or can be assigned to a particular node of the computing space. A distributed object is called replicated if the values of all its components are identical.

In mpF, the `DISTRIBUTION` and `REPLICATED` attributes are added to the declaration statements, and the `DISTRIBUTION` and `REPLICATED` specification statements are added.

The statement

```
INTEGER, REPLICATED, DISTRIBUTION(SPACE) :: N
```

declares the integer replicated variable `N` distributed over the entire computing space.

The statements

```
REAL MAT_ROW
DIMENSION(N) :: MAT_ROW
DISTRIBUTION(W) :: MAT_ROW
```

declare the real array `MAT_ROW` of the shape `(1:N)` distributed over the network `W`.

An expression can be evaluated by a single node (including the host processor) or by a region of the computing space. In the latter case, the expression is called distributed, and the region on which the expression is evaluated is called the *expression distribution region*. The value of a distributed expression also can be distributed over a network or subnetwork. This network (or subnetwork) must be a subregion of the expression distribution region. The region over which the value is distributed is called the *value distribution region*.

There are two important properties of distributed expressions. First, the expression may be asynchronous or synchronous. An asynchronous expression does not require any communication between the nodes of the computing space for its evaluation. A synchronous expression requires the communication between the nodes of the computing space for its evaluation. Second, value of expression can be either replicated or not. All components of replicated expression value are equal.

A new primary expression called the cutting expression is introduced. The cutting expression is determined by the specifier of a network or subnetwork that must be a subregion of the region of distribution of the operand value. The result is the corresponding segment of the distributed value of the operand. The cutting expression is evaluated asynchronously.

To get node coordinates in region of the computing space, the binary operator `.COORDOF.` is introduced. The right-hand operand of `.COORDOF.` specifies the region of the computing space, and the left-hand operand specifies the dimension of the right-hand operand. The result of `.COORDOF.` is the distributed integer value with the components equal to the corresponding coordinate of the node that stores these components. `.COORDOF.` is an asynchronous operator. This operator is slightly different from its mpC analog. In mpC the left-hand operand specifies the name of the coordinate variable. The set of intrinsic Fortran functions is extended by the distributed reduction functions by analogy with the intrinsic functions in Fortran 90 designed for operating on arrays.

4 Extension of the Assignment Statement

There are three types of distributed assignment statement: asynchronous, broadcast, and the parallel-send assignment. In the case of the broadcast and the parallel assignment the types of the variable that is assigned a value and the value of the expression to be assigned must be identical; i.e., no type conversion is allowed.

If the region of distribution of a variable coincides with the region of distribution of the value to be assigned, the assignment is executed asynchronously and component-by-component; i.e., each component of the distributed

variable is assigned the value of the distributed expression belonging to the same node.

If the region of distribution of the variable in the assignment operator includes more than one node and the region of distribution of the expression value consists of a single node, the value of the expression is broadcasted over the region of distribution of the variable and is assigned to its every component. We call such assignment *broadcast*.

If the variable and the expression value are distributed over different subregions of the same network or subnetwork and both subregions consist of the same number of nodes, a one-to-one mapping between the nodes is established, which is specified by a linear order. The components of the distributed expression value are passed to the corresponding nodes of the variable distribution region and assigned to the components of this variable. We call such assignment *parallel-send assignment*.

The following program fragment includes assignment statements of all three types.

```
NETWORK NET(N)
DISTRIBUTION (NET) K
INTEGER, REPLICATED, DISTRIBUTION (NET) :: I,J
...
K = 1.COORDOF.NET      ! asynchronous assignment
K = HOST(K)            ! broadcast assignment
NET(I)(K) = NET(J)(K)  ! parallel-send assignment
```

Here, the network `NET` consisting of `N` nodes and the variable `K` distributed over this network are described. The first assignment is asynchronous; all nodes belonging to `NET` execute it independently and concurrently. The value of the component of the variable `K` at a node of the network `NET` is equal to the coordinate of this node in the network. Because of cutting expression `HOST(K)` the value of expression in the second assignment belongs to single node `HOST`. The statement assigns to all components of the variable `K` the value of its component on `HOST`. The last statement assigns to the components of `K` on the node with the coordinate `I` in `NET` the value of its component on the node with the coordinate `J`. It is important that expressions in subnetwork specifiers in the broadcast and parallel-send assignment statements (in the example above, these are the variables `I` and `J`) must be replicated. For example, this makes it possible to avoid a frequent error (when programming in MPI) when processes incorrectly (differently) compute the coordinates of other processes involved in the assignment.

4.1 The Differences from mpC

There are two main differences in extension of assignment statement in mpC and mpF. The first one follows from considering the computing space as a network in mpF. In mpC the only distributed assignment allowed on computing space is broadcast. In mpF the parallel-send assignment is allowed as well.

In mpF there are three types of distributed assignment statement: asynchronous, broadcast, and the parallel-send assignment. In mpC there are two

additional types: gather and scatter. Let us consider the following example of mpC code:

```
net SimpleNet(N) w;
int [host]a[N], [w]b;
b   = a[];  // scatter
a[] = b;    // gather
```

Scatter. The left operand is distributed over some region of the computing space (network `w` in our example) and the right operand belongs to some node (`host` in our example). The value of the right operand belongs to a processor node of the network. The value of the right operand is a vector, whose elements may be assigned without a type conversion to components of the left operand, and the number of elements of the vector is equal to the number of components of the left operand. The execution of the operator consists in sending i-th element of the vector to i-th (in the natural numeration) node of `w`, where the element is assigned to i-th component of the left operand for all i from 0 to N-1.

Gather. The value of the right operand is distributed over some region of the computing space (network `w` in our example), and the left operand belongs to some node (`host` in our example). The left operand is an lvector whose length is equal to the number `N` of components of the value of the right operand, and the type of members of the lvector is compatible in relation to assignment with the type of components of the value of the right operand. The execution of the operator consists in sending i-th (in natural numeration) component of the value of the right operand to host, where it is assigned to i-th member of the left operand for all i from 0 to N-1.

The main disadvantage of those scatter and gather is rather complex semantics. Moreover if number of nodes in the computing space region and length of the vector are computed in run time those scatter and gather may be the source of hard to detect errors.

We did not include those types of distributed assignment statement because they can be easily replaced with loop and parallel-send.

5 Structure of the Program

There are two classes of procedures in mpF: nodal and distributed.

All subroutines and functions of Fortran 90 (including the intrinsic ones) are nodal procedures in mpF. Nodal procedures are executed by individual nodes of the region of the computing space where the values of the actual arguments are located. If the actual arguments are distributed expressions, then the nodal procedure is independently executed by the nodes of the region of distribution of these expressions' values; no communications within the nodal procedure are allowed, and the creation of nets or subnets is prohibited. If all the actual arguments are asynchronous, then the call to the nodal procedure is an asynchronous expression.

Distributed subroutines and functions are executed by nodes of a certain region of the computing space. Syntactically, distributed procedures do not differ from nodal ones. The difference is that the distribution of the name of such a procedure must be explicitly specified; the other difference is in the list of formal parameters.

In addition to the parameters with the names interpreted in the conventional (for Fortran) way, any distributed subroutine or function must have exactly one network parameter. The name of this parameter must be the name of a user-defined (in the scope of the procedure or outside it) network or an intrinsic network.

The procedure is distributed over the region of the computing space determined by the network parameter. The formal parameters of such a procedure must be described (explicitly or implicitly) as distributed over the network parameter or as belonging to the parent node of the network parameter. If the procedure has no explicit interface, its formal parameters by default are distributed over the region of the computing space determined by the network parameter.

If the specification section of the distributed procedure contains the DISTRIBUTION statement that contains the name of the function in its list, then this statement defines the region of distribution of the value returned by the function. The following program fragment defines the distributed procedure `INPUT_DATA` that is rigidly connected to the network `HOST` and the function `PAR_MXM` distributed over the network `LINE`. Some formal parameters of this function and function value are distributed over the parent of the network `LINE`.

```
SUBROUTINE INPUT_DATA (HOST,N,X,Y)
...
END
FUNCTION PAR_MXM (LINE,N,MDIM,A,B)
   NETWORK LINE(N)
   REPLICATED N
   DISTRIBUTION(PARENT(LINE)) :: PAR_MXM,A,B,MDIM
   ...
END FUNCTION PAR_MXM
```

5.1 The Differences from mpC

In mpC there are three classes of functions: nodal, network and basic. The network and basic functions are analog of distributed procedures in mpF. The difference between network and basic functions is that the basic function is rigidly connected to the entire computing space. Because entire computing space is not considered as a network in mpC basic functions forms a separate class of functions.

6 Conclusions

Because mpF is the parallel extension of Fortran and mpC is the parallel extension of C those two languages are different. But the main concepts of the parallel extension are similar for the both mpC and mpF.

We started the mpF project with about 8 years of development and using mpC in mind. While keeping the mpC parallel programming model in general, we have slightly reduced and simplified it. As a consequence the distinction from Fortran 90 is minimal thereby decreasing efforts needed for mastering mpF by Fortran users.

mpF is aimed at parallel programming on a variety of platforms ranging from shared memory machines and specialized clusters to commodity heterogeneous local networks. It provides a good balance between ease of use for a programmer and parallel efficiency.

Acknowledgements

This research is supported by Computational and Information Aspects of Solving Large Problems program of the Division of Mathematical Sciences of the Russian Academy of Sciences.

References

1. Lastovetsky A., Arapov D., Kalinov A., Ledovskikh I., A parallel language and its programming system for heterogeneous networks, *Concurrency: practice and experience*, 2000, 12, pp. 1317-1343
2. Lastovetsky A., Parallel Computing on Heterogeneous Networks. John Wiley & Sons, 2003
3. Message Passing Interface Forum. MPI: A Message Passing Interface Standard, Version 1.1, June, 1995, *www.mpi-forum.org/docs/docs.html*.
4. High Performance Fortran Forum. High Performance Fortran Language Specification, Version 2.0 (Rice University, Houston, 1997), *http://dacnet.rice.edu/Depts/CRPC/HPFF/versions/hpf2/hpf-v20/index.html*.
5. Kalinov, A. and Lastovetsky, A., Heterogeneous Distribution of Computations while Solving Linear Algebra Problems on Networks of Heterogeneous Computers, *J. Parallel Distrib. Comput.*, 2001, vol. 61, no. 4, pp. 520-535.
6. Blackford, L.S., Choi, J., Cleary, A., d'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hanmmarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R.C., ScaLAPACK User's Guide, Philadelphia: SIAM, 1997.
7. A. Kalinov and I. Ledovskikh, An Extension of Fortran for High Performance Parallel Computing, *Programming and Computer Software*, Vol. 30, No. 4 (2004) 209-217. *Translated from Programmirovanie*, Vol. 30, No. 4, 2004.
8. Kalinov A., Ledovskikh I., Posypkin M., Levchenko Z., and Chizhov V., An Implementation of the Matrix Multiplication Algorithm SUMMA in mpF. *Proceedings of 8th International Conference PaCT'05*, Krasnoyarsk, Russia, September 2005, 420-432

Java Programs Optimization Based on the Most–Often–Used–Paths Approach

Eryk Laskowski[1], Marek Tudruj[1], Richard Olejnik[2], and Bernard Toursel[2]

[1] Institute of Computer Science PAS, 01-237 Warsaw, Ordona 21, Poland
{laskowsk, tudruj}@ipipan.waw.pl
[2] Université des Sciences et Technologies de Lille,
Laboratoire d'Informatique Fondamentale de Lille (LIFL UMR CNRS 8022)
{olejnik, toursel}@lifl.fr

Abstract. The paper presents a Java byte–code optimization algorithm, which determines an initial distribution of objects among virtual machines (JVMs) so as to decrease direct inter–object communication and balance loads of the virtual machines. The proposed optimization algorithm is based on a graph representation of control and data dependencies between methods in Java programs. These dependencies, expressed in the form of conditional macro–dataflow graphs, are discovered by a static analysis of program byte–code.

Object placement optimization is performed before a Java program is executed in a parallel system. The optimization methods are based on the Dominant Sequence Clustering (DSC) approach. First, macro nodes are clustered on an unlimited number of processors (logical JVMs) to reduce the total program execution time. Next, clusters are merged and scheduled to adjust the number of logical JVMs to the number of real processors. The presented approach is supported by branch optimization techniques, which include detection of mutually–exclusive paths and scheduling of most–often–used–paths based on branch probabilities.

1 Introduction

The paper presents a new optimization technique for multithreaded Java programs, executed on a set of distributed JVMs. The motivation of this work comes from current practice of object oriented computing, where objects are usually placed using arbitrary assignment to available system JVMs. Such a strategy can result in an unbalanced execution of programs at run time. A solution for the problem can be an optimization algorithm, performed before a Java program is executed in a parallel system, which will determine the best distribution of program elements on virtual machines.

An optimization algorithm, presented in this paper, is based on a static pre–optimization method applied to the byte–code generated by the Java compiler. First, it constructs a data flow graph of the Java program based on data and control dependencies in the byte–code. Next, a fine grain analysis of the data flow graph is done, which results in the design of a macro data flow graph (MDFG). The graph depicts data and control dependencies between the sequential code

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 944–951, 2006.

macro data flow nodes identified inside threads and methods of a sequential program. In particular, it includes branch nodes, which transfer control in the program in a data dependent way. The branch nodes are annotated with the probabilities of taking the constituent output paths. This graph is transformed into a parallel version distributed on a set of processors (JVMs) in a way, which reduces program execution time. Clustering and scheduling heuristics are applied to the macro nodes of the generated graph to decrease inter–processor data communication and balance processor computation loads. The heuristics are based on the use of the *most–often–used–path* approach and exploitation of *mutually–exclusive paths* in program graphs [6, 7].

The paper is composed of 4 parts. The first part explains how macro data flow graphs are generated from program byte–code. In the second part, we discuss the node clustering and scheduling technique applied to the macro data flow graphs. The third part explains management and execution of distributed threads corresponding to nodes of scheduled Java program graphs on JVMs. The fourth part presents an example of optimization of a Java parallel program for matrix multiplication based on recursive data decomposition.

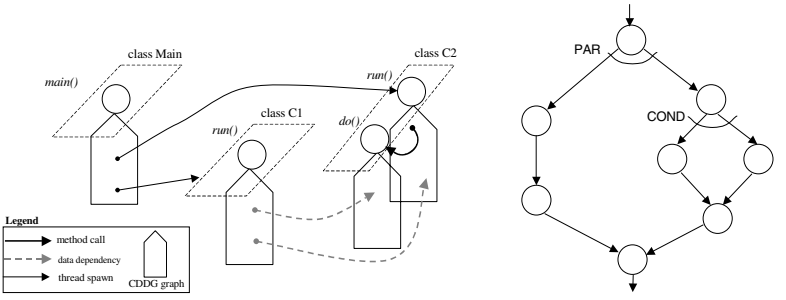
2 Parallel Program Representation

Presented optimization algorithms use graph representations of Java programs [1]. These dependence graphs are the result of the analysis of the program byte–code. The analysis identifies control and data dependencies between byte–code instructions. There are several program tools [1], which are able to generate such graph representations of programs in an automated way.

We introduce two intermediate program graph representations: MTCG and CDDG graphs. The nodes in a Method/Thread Call Graph (MTCG) represent methods of program objects, the edges represent method calls. Each called method has its own Control/Data Dependence Graph (CDDG). The nodes in the CDDG graph correspond to sequences of byte–code instructions that appear inside a method byte–code (basic blocks of byte–code). The edges correspond to control and data flow between byte–code instructions of a method. Data dependence edges connect byte–code instructions, which result in data exchange between methods. An example of MTCG and CDDG graphs, with data and control dependence edges between CDDG of methods, is shown in Fig. 1(a).

We annotate MTCG and CDDG graphs with statistics gathered during program execution for sets of representative data (profiling). During profiling, the number of mutual method calls and thread spawns is measured. Also the way in which branch and loop instructions are executed is registered. The profiling results for all created objects in each class, all called methods, all spawned threads, including statistical parameters of all branch and loop instructions, are stored in trace files. We use a profiling tool, which is similar to that used in ADAJ environment [2].

The final result of the analysis of program byte–code and trace files is a Macro Data Flow Graph (MDFG) of a program. The MDFGs are obtained by the



(a) A MTCG graph

(b) A MDFG for MTCG from Fig. 1(a)

Fig. 1. Example graph representations of a Java program

transformation of the control/data dependence graphs. It is done by agglomerating in macro nodes all sequential byte–code instructions, which are separated by method calls, thread spawning, accesses to remote object data fields and branch (loop) instructions. Control and data dependence edges between boundary byte–code instructions in adjacent macro nodes are replaced by data dependence edges between macro nodes. The nodes have weights equal to their execution time. The weights of edges correspond to the serialization and transmission time of data, which are necessary to execute the target macro nodes on a remote JVM.

The obtained MDFG graphs can include conditional branch macro nodes. Branch nodes correspond to sequential byte–code instructions ending with a conditional jump instruction (i.e. **if_icmple**, **if_icmpne** or other opcodes) or a switch instruction (i.e **lookupswitch** opcode). A branch macro node can have several outgoing paths: two – for a branch instruction or several – for a switch instruction. Loops in the MDFG graph are subject to unrolling. If the number of iterations in a loop cannot be estimated, we form a macro node encapsulating the whole loop. We assign execution probability to each outgoing path of a conditional branch macro node based on an analysis of the program execution traces. Fig. 1(b) shows the exemplary MDFG graph of a part of a program.

3 Program Optimization Algorithm

The presented algorithm is based on static program graph optimization methods that lead to a reduction of the total program execution time in a set of parallel JVMs. These optimization methods are applied to the described macro data flow graph (MDFG) of Java programs. The first step of optimization consists of clustering of macro nodes on unlimited number of processors (logical JVMs) to reduce the execution time of the clustered nodes. Next, the clusters are merged and scheduled with simultaneous load balancing to reduce the number of logical JVMs to the number of real processors in the system.

We use modified clustering algorithms based on a dominant sequence approach (DSC heuristics) extended by *trace scheduling* approach [5] and

most–often–used–path optimization method [7]. The priority of nodes determines the selection order for clustering. In our modified algorithm, the priority of a node n is the sum of *tlevel*(n) and *blevel*(n), where the *tlevel*(n) is the length of the longest path from the entry graph node to n (excluding n) and the *blevel*(n) is the length of the longest path from node n to an exit node. At each clustering step, the algorithm selects a node with the highest priority from the set of *ready nodes* (B_R in Fig. 2), i.e. such nodes, whose parents have all been analyzed (visited) and which belong to the currently analyzed subgraph – *CS*. If there is no *ready node* in *CS*, the algorithm selects another outgoing path of a recently visited branch macro node to determine the new current subgraph. These paths are selected in descending order of their probabilities, i.e. *most–often–used–path* optimization is used. When the algorithm meets a branch macro node in the current subgraph, a outgoing path with the highest probability becomes the new current subgraph and the node is put into the stack of visited branch nodes *BS*. The branch node is removed from the top of *BS* when all nodes in its outgoing trees have been visited. Then, a recently considered tree of a branch node from the top of *BS* becomes the current subgraph.

```
Input: B - set of all macro nodes of the program graph
Assign each b ∈ B to a separate cluster
For each b ∈ B
     Find blevel(b)
tlevel(b0)= 0
BU = B    /BU - set of unvisited nodes of the program graph/
BS = ∅
Set outgoing sub-graph of b0 as current sub-graph - CS
While BU ≠ ∅
     Find set of "ready" nodes BR ⊂ BU in CS
     Find bR ∈ BR with the biggest priority PRIO(bR)
     If bR is not first node in CS then merge bR with:
          1) the cluster of the preceding node for which tlevel(bR) after
             merging decreases to the maximal degree
          2) all clusters of preceding nodes, which have bR as the only
             child and produce a decrease of tlevel(bR) after merging
          (if there are no such nodes as in 1 or 2, leave bR in its cluster)
     BU = BU - {bR}    /mark bR as visited/
     Update PRIO values for nodes in BU
     If bR is branch node then
          Store bR on top of branch stack BS with the state of its CS
          Select sub-graph of outgoing path of bR with the highest
              probability as the new CS
     If all nodes in all outgoing CSs of the branch node on top of BS
                                              have been visited then
          Remove the node on top of BS
          If the top of BS is not empty then
              Set the unvisited CS of the node on top of BS as the new CS
EndWhile
```

Fig. 2. Outline of the clustering algorithm for MDFGs with branches

```
Assign priorities to nodes   /priority of node n: blevel(n)/
Detect mutually exclusive branch sub-graphs
While not all nodes have been scheduled
      While there is ready node and free processor list is not empty
            For each free processor p and each ready node n
                  Compute est(n, p) - earliest starting time of node n on
                        processor p,
            Schedule node n to a processor p, which minimizes est(n, p)
      EndWhile
      Move time to the point in which a processor becomes free
      Update the list of free processors
      Update ready node list
EndWhile
```

Fig. 3. Outline of the macro nodes scheduling algorithm

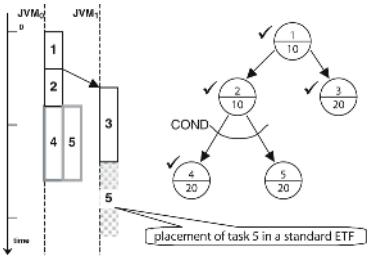


Fig. 4. Scheduling of mutually–exclusive paths

The scheduling algorithm of clusters for MDFG with branches, Fig. 3, is based on modified earliest task first (ETF) heuristics [8]. The algorithm employs *detection of mutually–exclusive paths* optimization method [6]. The algorithm selects for scheduling a cluster n and a processor p, which assure the lowest value of earliest starting (execution) time $est(n, p)$. When node n is scheduled on processor p, it can be inserted in parallel with nodes from mutually exclusive paths of branches if they are assigned to this processor, otherwise the node n is appended as the last node on processor p.

An example of scheduling of mutually–exclusive paths of a program graph is shown in Fig. 4. Nodes 1, 2, 3, 4 of MDFG on the left side of Fig. 4 have already been scheduled. The scheduling algorithm has to decide where to place task 5. In a standard ETF approach, this task would be placed on JVM1 (gray rectangle in Fig. 4). The proposed algorithm detects mutually–exclusive paths, thus task 5 can share the JVM0 with task 4, since these tasks will never be executed together. The total length of the schedule has been reduced.

4 Runtime Support for Execution of Schedules

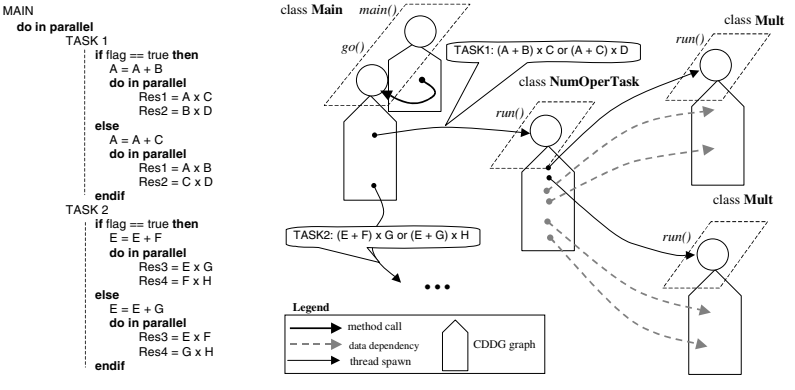
Macro nodes of the scheduled macro data flow graph (MDFG) of the program will be executed in general on different JVMs. Some nodes of the macro data

flow graph will be executed on a different JVM than the initial JVM of the spawning thread. Such nodes have to be transformed into distributed threads. We construct distributed threads out of portions of method byte–code. For this, we can follow the technique applied in JavaParty [3] or in the Brakes tool [4], which enables dynamic thread distribution between JVMs. Such technique is new, comparing standard strategy for object assignment to processors, where all methods of an object are allocated to the same virtual (or real) processor.

The scheduled macro data flow graph (MDFG) of the program is executed on standard (unmodified) Java Virtual Machines. A distributed thread manager daemon process is installed on each virtual machine used. This process is provided with special supervisor methods, which update the application program object context concerned with the thread execution and load the code of the created thread and a static copy of the object that the thread cooperates with. The transmission of serialized objects is used for this purpose. After remote execution of the distributed thread corresponding to a macro data flow graph node, the return to the parent node (thread) can take place by sending back to the parent virtual machine the serialized context of the next thread, which is to be executed. The applied technique is similar to that explained in [3, 4].

5 Example

In this chapter, we present an example of the scheduling process of a simple parallel program containing conditional branches. The program consists of two parallel tasks (TASK1 and TASK2 in Fig. 5(a)), which are spawned from the main method of the program (MAIN in Fig. 5(a)). Each task performs matrix addition and then spawns two threads for parallel execution of matrix multiplication. Depending of the value of a conditional variable (**flag** in Fig. 5(a)), matrix addition and multiplication operate on different input matrices.



(a) Pseudocode (b) The MTCG graph

Fig. 5. An examplary parallel Java program

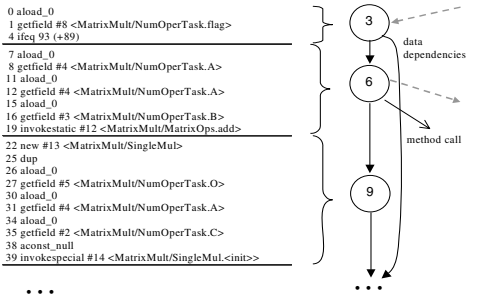
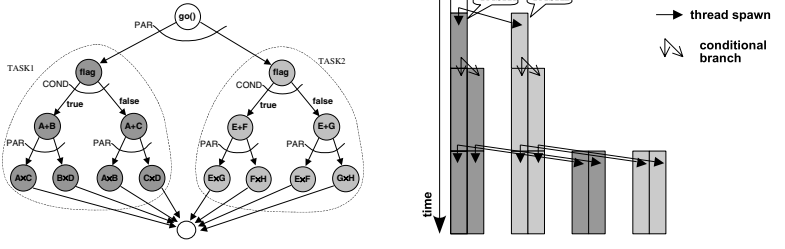


Fig. 6. Part of CDDG graph of TASK1 main method



(a) After the clustering phase (b) After the mapping phase

Fig. 7. The results of the optimization

The presented program optimization algorithm assumes that a system library of arithmetic operations performs matrix addition and multiplication methods. These library methods are treated as single, atomic operations, thus their byte–code is not subject to an analysis and scheduling. The distinction between the program and the system objects is based on recognition of package names.

After the analysis of the program byte–code and the trace file, generated during program profiling, a MTCG graph is constructed (Fig. 5(b)). A part of the CDDG graph of one of program methods is shown in Fig. 6.

The MTCG, along with the attached CDDGs for each node, is the basis for the macro–dataflow graph of the program (MDFG). Our exemplary MDFG graph contains conditional branch macro nodes, since the flow of control depends on the value of **flag** variable. During the clustering phase, clustered macro–nodes are constructed from macro–nodes of the MDFG. In the clustered MDFG (Fig. 7(a)), conditional nodes are shown as circles with outgoing edges marked with "COND" label (edges marked with "PAR" represent thread spawning).

The result of the mapping phase is shown in Fig. 7(b). Macro nodes of the MDFG have been scheduled onto a set of JVMs. Macro nodes implied by mutually exclusive conditional branches are mapped in parallel on the same JVM, which provides reduction of program execution time.

6 Conclusions

A Java program optimization algorithm has been proposed in the paper. The proposed algorithm assumes representation of Java programs as macro data flow graphs, generated on the basis of the program byte–code. The algorithm contains several optimization steps, which can deal with conditional nodes and loops in program graphs. These steps include the most–often–used path scheduling and the trace–based approach during the clustering phase. A modified ETF list scheduling heuristics, combined with detection of mutually exclusive paths optimization method is used in the mapping phase. The scheduled Java programs distributed on a set of JVMs, provide reduction of program execution time.

References

1. J. Zhao, *Dependency Analysis of the Java Byte code.* Proc. of the 24th IEEE Annual Int. Computer Software and Application Conference, Taipei, Oct. 2000.
2. A. Bouchi, R. Olejnik and B. Toursel, *A New Estimation Method for Distributed Java Object Activity*, IPDPS – Workshop on Java for Parallel and Distributed Computing, Fort Lauderdale, USA, 2002.
3. M. Philippsen, M. Zenger, *JavaParty – Transparent Remote Objects in Java.* Concurrency: Practice & Experience, Vol. 9, No. 11, pp. 1225-1242, November 1997.
4. D. Weyns, E. Truyen, P. Verbaeten, *Distributed Threads in Java*, Proceedings of the NATO Adanced Research Workshop on Advanced Environments, Tools and Applications for Cluster Computing, Mangalia, Romania, September 2001.
5. M. Agarawal et al, *Speculative Trace Scheduling in VLIW Processors*, IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors, Germany, 2002.
6. Yuan Xie, Wayne Wolf, *Allocation and scheduling of conditional task graph in hardware/software co–synthesis.* DATE 2001, pp. 620-625.
7. Ciaran Murphy, Xiaojun Wang, *Most Often Used Path Scheduling Algorithm,* Proceedings of the 5th World Multi–Conference on Systemics, Cybernetics and Informatics (SCI 2001), July 22-25, 2001, Orlando, Florida, USA. Vol. XII, pp. 289-295.
8. Jing–Jang Hwang et al, *Scheduling Precedence Graphs in Systems with Interprocessor Communication Times*, Siam J. Comput., Vol. 18, No. 2, 1989.

Vertex-Magic Total Labeling of a Graph by Distributed Constraint Solving in the Mozart System

Adam Meissner and Krzysztof Zwierzyński

Institute of Control and Information Engineering,
Poznań University of Technology,
pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, Poland
{Adam.Meissner, Krzysztof.Zwierzynski}@put.poznan.pl

Abstract. In this paper we present how a problem of a vertex-magic total labeling of a graph may be expressed in terms of constraint programming over finite domains (CP(FD)) in the Mozart system. A program representing the problem is easily transformable into a parallel version, which can be executed on distributed machines. We describe the results of experiments for estimating a speedup, a work granularity and an overhead of a parallel version in comparison with sequential computations.

1 Introduction

Graph labeling is a well-known and intensively investigated problem in graph theory (e.g. [2]). In general, it consists in assigning numbers to vertices or edges of a given graph in the way it meets some detailed conditions. A particular case of graph labeling is called a *vertex-magic total labeling* [3]. It allocates subsequent numbers (beginning from 1) to vertices and edges of a graph, so a sum of the vertex label and the labels of all edges incident to it is equal for every vertex. It should be noted that the considered labeling does not exist for every graph.

A general problem of constructing a vertex-magic total labeling can be solved by algorithmic methods. As one of them we consider a *constraint programming over finite domains* (e.g. [1], [4], [6]). In this approach a program is a set of formulas (i.e. constraints) with variables ranging over some initial finite domains, which are usually represented by sets of non-negative integers. A *constraint solving* consists in finding a variable assignment which satisfies all considered formulas. This effect is achieved by performing two operations on the constraint set, namely a *propagation* and a *distribution*. In general, a propagation successively narrows the range of the variables according to the semantics of particular constrains. This activity aims to *determine* the variables, that is to restrict their domains to one-element sets. The propagation may also reach a fixed point although not all variables are determined. In such a case, a distribution is performed which consists in creating two new sets of constraints, to wit $C \cup \{x = l\}$ and $C \cup \{x \neq l\}$, where C represents a current set of constraints, x is one of undetermined variables and l is a value from the actual domain of x. Rules of selection x and l form a *distribution strategy*. A propagation alternating with a distribution are

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 952–959, 2006.

performed until all variables are determined or until the constraint set becomes contradictory (e.g. it contains both $x = l$ and $x \neq l$). This process may be modeled by a *search tree* which vertices are the constraint sets created during the distribution.

A great advantage of the constraint programming methodology is that a program may be executed in sequence as well as in parallel practically without any modifications of the code. This is caused by the fact, that the way in which the computational process is performed depends only on a strategy deciding how the search tree is explored, which has no impact on the program. In particular, the search tree may be decomposed into subtrees, which are explored independently. This approach may significantly reduce the computation time if the search tree consists of an appropriate number of subtrees of the similar height.

The concept of a parallel constraint solving on distributed machines was implemented in the Mozart system [7], which is a computational environment for the Oz language [5]. The system contains special objects, the so-called *search engines*, intended for exploring a search tree in various ways including the distributed parallel search.

The paper is organized as follows. In section 2 we show that the problem of a vertex-magic total labeling of a graph may be easily denoted as a program in the Oz language with constraints, which are predefined in the Mozart system. In section 3 we describe the results of our experiments aimed at comparing the efficiency of a program execution concerning the sequential and, respectively, the parallel exploration of the search tree. Section 4 contains some final remarks.

2 Formulating the Problem

Let $G = \langle V, E \rangle$ be a simple graph consisting of a set of vertices V and a set of edges E, let $E_v \subseteq E$ denote a set of all edges, incident to a vertex $v \in V$ and let "$|\cdot|$" symbolize the cardinality of a set. A *vertex-magic total labeling* of a graph G ([3]) is an injective mapping f from $V \cup E$ to the set of integers ranging from 1 to $|V \cup E|$, such that for every two vertices u and v, $f(u) + f(d_1) + \ldots + f(d_m) = f(v) + f(e_1) + \ldots + f(e_n)$, where $m = |E_u|$, $n = |E_v|$, $d_i \in E_u$ for $i = 1, \ldots, m$ and $e_i \in E_v$ for $i = 1, \ldots, n$.

The problem of searching for a vertex-magic total labeling of a graph may be reduced to solving the set P of diophantine equations and inequalities, which is described below. Let X be a set of all unknowns appearing in P. Every unknown represents a label assigned to an element of the set $V \cup E$; it should be observed that $|X| = |V \cup E|$. The elements of the set X are denoted by x_i for $i = 1, \ldots, |X|$. Moreover, we assume that the unknown which symbolizes a label of a vertex v may be also indicated by x_v while the symbols $x_{v,i}$ for $i = 1, \ldots, |E_v|$ are alternative denotations for unknowns representing labels of the edges incident to a vertex v. The set P contains equations and inequalities of the following three types.

$$x_v + x_{v,1} + \ldots + x_{v,n} = s \qquad \text{where } n = |E_v| \tag{1}$$

$$1 \leq x_i \leq |X| \qquad \text{for } 1 \leq i \leq |X| \tag{2}$$

$$x_i \leq x_j \qquad \text{for } 1 \leq i < j \leq |X| \tag{3}$$

Every equation of the form (1) corresponds to one vertex v from the set V. A left-hand side of this equation is called a *sum of labels at a vertex v*. A subset of P, containing all equations (1), represents the condition stating that the sum of labels must be the same at every vertex and equal to some constant s. Every inequality of the form (2) restricts the range of one unknown from the set X, which stands for the assumption that every label belongs to the interval $1, \ldots, |V \cup E|$. The inequalities of the form (3) express the demand that the function f is an injective mapping.

Below, we give an example of constructing a vertex-magic total labeling of a simple graph depicted in Fig. 1. The symbols assigned to vertices and edges are

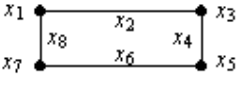


Fig. 1. A graph with unknowns representing labels

unknowns from the set P, which in this case consists of the following equations: $x_8 + x_1 + x_2 = s$, $x_2 + x_3 + x_4 = s$, $x_4 + x_5 + x_6 = s$, $x_6 + x_7 + x_8 = s$ and the following inequalities: $1 \leq x_i \leq 8$ for $1 \leq i \leq 8$ and $x_i \neq x_j$ for $1 \leq i < j \leq 8$. A vertex-magic total labeling of the graph form Fig.1 is any solution of the set P; in one of them, the vector $\langle x_1, \ldots, x_8 \rangle$ equals $\langle 4, 2, 7, 3, 8, 1, 5, 6 \rangle$ and $s = 12$.

As said before, the set P may be transformed into a program in the Oz language using the constraint programming paradigm. The program has to create a set of variables representing unknowns from the set X and impose appropriate constraints on them. For this purpose the function `MagicColor` is defined. It takes `ETab` – the adjacency matrix of an input graph as an argument and it returns a procedure which creates a set of variables with constraints imposed on them. The symbol `$` standing for a procedure name in line 2 and in line 6 causes that the expression denoting the procedure becomes its identifier.

```
fun {MagicColor ETab}                                              % 1
  proc {$ Sol}                                                     % 2
    Constr                                                         % 3
  in                                                               % 4
    {MakeConstr ETab Constr Sol}                                   % 5
    {ForAll Constr proc {$ L} {FD.sum L ':' S} end}                % 6
    Z = {List.length Sol}                                          % 7
    Sol ::: 1#Z                                                    % 8
    {FD.distinct Sol}                                              % 9
    {FD.int ({MinDeg ETab}+1)#({MaxDeg ETab}+1)*Z S}               % 10
    {FD.distribute ff Sol}                                         % 11
  end                                                              % 12
end                                                                % 13
```

In the first step (line 5) the procedure `MakeConstr` is called; it takes the adjacency matrix `ETab` and returns two lists called `Constr` and `Sol`. The list `Constr` consists of lists L_i for $i = 1, \ldots, |V|$ which are used for the creation of constraints corresponding to equalities of the form (1). Every list L_i contains variables representing unknowns which appear on the left-hand side of one equation. In other words, an element of the list `Constr` which corresponds to a vertex $v \in V$ has the structure $[x'_v\ x'_{v,1} \ldots\ x'_{v,n}]$, where $n = |E_v|$ and the symbols $x'_v, x'_{v,1}, \ldots, x'_{v,n}$ stand for Oz variables representing unknowns from the set X. On every list L_i for $i = 1, \ldots, |V|$,a constraint is imposed stating that the sum of variables contained in the list must be equal to a variable `S` denoting a constant s from (1). This is done by the predefined procedure `FD.sum` called in line 6, which is applied to every element of the list `Constr`.

As said before, the list `Sol` consists of variables representing unknowns from the set X. The length of the list (counted in line 7) equals to the cardinality of this set. In lines 8 and 9, two kinds of constraints are imposed on every element of `Sol` corresponding to inequalities of the form (2) and (3), respectively. The predefined procedure `FD.distinct`, called in line 9, expresses the condition according to which all elements of the list `Sol` have to get distinct values.

Additionally in line 10, the domain of the variable `S` is narrowed in order to reduce the computational time, since the sum of labels at every vertex from the set V satisfies the condition $MinD + 1 \leq s \leq |V| \cdot (MaxD + 1)$, where the symbols $MinD$ and $MaxD$ denote the minimal and, subsequently, the maximal vertex degree in the graph V.

However, a basic impact on the efficiency of the computational process has a distribution strategy. The procedure constructed by the function `MagicColor` uses (line 11) the strategy called *First Fail* (in abbreviation: *ff*), which is predefined in the Mozart system. We have found experimentally that for most examples of the input data, it generates smaller search trees than any other predefined strategy. The *ff* strategy works as follows: it selects a leftmost undetermined element from the given list of variables (in this case, the list `Sol`) for which the current domain has the minimal size. Hence, the order of elements of the list `Sol` has an influence on the efficiency of the computational process. Regarding this, we have observed that the best results are obtained when the variables corresponding to edge labels precede the variables which represent vertex labels. It follows from the fact that every unknown standing for a vertex label in the set P appears only in one equation (1), while every unknown denoting an edge label is a member of two equations of this form. For this reason, the distribution applied to edge labels generally leads to stronger effects of the propagation in comparison to the vertex label distribution.

As we have already said in section 1, the solution of a given problem in the constraint programming paradigm is obtained by the exploration of a search tree. In the Mozart system, this process is performed by search engines which are instances of various subclasses of the class `Search`. Every subclass implements a particular strategy of the search tree exploration. In the experiments described in section 4, a sequential exploration of the tree is accomplished by an object from the class `Search.object`. Below, we quote a fragment of the program, where

the first line creates an appropriate object and the second line sends a message to it demanding the next solution of the given problem, which is represented by the list of variables `Y`.

```
E = {New Search.object script({MakeMagic IncTab})}
{E next(Y)}
```

The objects from the class `Search.parallel` are used for the parallel exploration of the search tree. The creation of this kind of object initiates the process called a *manager*, which controls all the computations. The manager designates so-called *workers*, that is, processes intended for exploring the tree. It also determines the number of workers and indicates the computers where particular workers are to be run. For example, the following command

```
E1 = {New Search.parallel init(a:1#ssh b:1#ssh)}
```

states, that the search tree will be explored in parallel by two workers, one of them runs on the computer called `a` and the other one on the machine `b`. The manager communicates with workers using the remote command interpreter `ssh`. The demand for the solution is expressed in a similar way as in the sequential exploration. Additionally, one can specify whether all the solutions have to be found or just one of them.

3 Evaluation

As said before, the experiments presented in this section were aimed at evaluating the efficiency of the parallel execution of the program described above in comparison with the sequential processing. We considered the following criteria, similar to those suggested in [4]:

- a *time overhead*, introduced by the method of distributing the task on individual computers,
- a *speedup*, caused by parallelization of the computational process,
- a *work granularity*, i.e., a degree of the dispersion of the search tree on particular machines.

It should be underlined that all measurements given in this paper are approximations due to the heterogeneity of the computational environment, which is described in the sequel.

All tests consisted in finding one example of a vertex-magic total labeling of graphs shown in Fig. 2. The graphs were chosen under the basic condition that for each of them the time of sequential computations was restricted to the range of 20 to 200 sec. for every machine. The symbols We_8, $K_{4,3}$, Pet and $T_{3,3}$, which appear in Fig. 2, denote individual graphs; in particular, the symbol Pet indicates the Petersen graph, well-known in graph theory. The numbers assigned to vertices and edges are an example of a vertex-magic total labeling. It should be noted that a labeling of this type does not exist for the graph $T_{3,3}$.

The computational environment used for the tests consisted of six machines powered by the Mozart system 1.3.1; their parameters are given in Table 1.

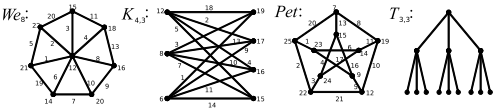


Fig. 2. Test graphs with a vertex-magic total labeling

Table 1. The parameters of machines which form the computational environment

Name	Processor	RAM	Ethernet	Operating system
W1	Celeron, 433 MHz	192 MB	100 MBit	Win. 2000 Prof.
W2	Pentium III, 433 MHz	256 MB	100 MBit	Win. NT Workstation 4.0
W3	Pentium III, 433 MHz	256 MB	10 MBit	Win. NT Workstation 4.0
W4, W5	Pentium III, 866 MHz	256 MB	100 MBit	Win. NT Workstation 4.0
M	Pentium IV, 1.8 GHz	256 MB	100 MBit	Win. 2000 Prof.

In the tests performed for the speedup and for the work granularity, we considered various configurations of the computational environment, containing from one to five computers on which the workers were run (one worker on one computer). The symbol M denotes a machine on which only the manager was executed. The machines processing the workers (W1, ...,W5) were introduced to the environment according to their order given in Table 1. For example, a variant of the configuration with two workers was compounded of the machines called M, W1 and W2, while a variant with three workers consisted of M, W1, W2 and W3. For the sake of simplicity, when it does not lead to misunderstanding, we identify the computer M with the manager and the computers W1, ...,W5 with the workers in the sequel. For every test, a computational time is a wall time (i.e. a system clock time) taken as an arithmetic mean of seven runs.

Time overhead. A test concerning the time overhead for every computer W1, ..., W5 and every graph consists of two steps. In the first step of our experiments, the time of sequential computations T was taken. Then, the parallel computations were performed for the same input data in which both the manager and one worker were run on the given computer. The time overhead is regarded as an increment of the computational time ΔT, which appeared when the test was run in parallel. The results of the tests are given in Table 2.

Table 2. The time overhead ΔT [sec.] for computers W1,..., W5 in comparison with the time of sequential computations T [sec.]

	W1		W2		W3		W4		W5	
Graph	T	ΔT	T	ΔT	T	ΔT	T	ΔT	T	ΔT
Pet	52.06	6.96	46.98	5.33	47.19	3.96	23.3	4.85	23.35	4.84
$T_{3,3}$	87.96	7.17	79.48	1.44	79.73	0.79	39.46	2.58	39.52	1.28
$K_{4,3}$	135.94	17.25	122.04	6.41	122.62	6.48	60.73	4.28	60.82	3.61
We_8	181.40	19.66	163.32	4.33	163.96	1.19	81.05	4.18	81.13	4.25

It should be noted that the time overhead for W1 differs significantly from the results obtained for the other computers (particularly for the graphs $K_{4,3}$ and We_8). This effect might have been caused by some factors independent from the method of parallelization. Hence, we neglect these results in further considerations. In the remaining cases, one may observe a general regularity that the time overhead ΔT depends mainly on the tested graph and, to a much smaller extent, on the time T. This indicates that the time overhead is determined not as much by the computational power of a machine as by the communication efficiency of the computational environment. In addition, a relatively small spread of the ΔT values may suggest that a significant part of ΔT for the tested graphs is the time necessary to establish the connection using the remote shell `ssh`.

Speedup and work granularity. A speedup of computations is defined as a quotient of the time of computations performed in the environment consisting of the manager and one worker, and in the environment with subsequently growing number of workers. The results of tests for the speedup are given in the bar chart located on the left side of Fig. 3. The bars depict the speedup achieved for individual graphs. A pattern of the bar denotes the number of workers in the computational environment according to the legend which is on the right side of the chart.

For the graph $T_{3,3}$ one can observe a practically linear increment of the speedup while the subsequent workers are embodied into the environment. In other cases, some irregularities can be noted depending on the one hand on super-linear speedup (as in case of We_8) and on the other hand on lengthening the computational time (e.g. for the graphs $K_{4,3}$ and *Pet* when the number of workers changes from four to five). These anomalies generally follow from the nature of the problem as well as from the nondeterministic algorithm of solving it. It should be remarked that partitioning the search tree into fragments, which are to be explored in parallel, is an example of a so-called *embarrassingly parallel problem*. This means that no particular effort is needed to segment the tree and it can be done in many ways. The program considered in this paper stops after finding one solution, in other words, one vertex in the search tree. Hence, the computational time remarkably depends on the way the tree is fragmented and on the localization of the vertex containing a solution in the particular fragment of the tree. In consequence, the computational time may seriously decrease if an appropriate vertex appears in an initial part of one of the currently created

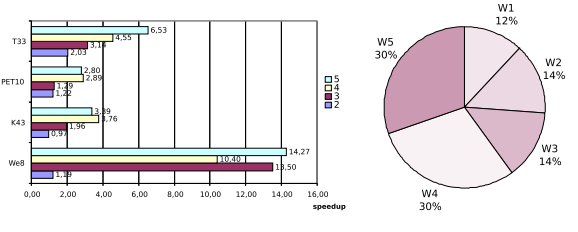


Fig. 3. Speedup and work granularity

subtrees. On the other hand, the time may increase if the tree is divided in a less advantageous manner. These effects do not take place in case of the graph $T_{3,3}$, which cannot be labeled in the way considered in this article, therefore it demands the exploration of the whole search tree. A practically linear speedup of computations, which has been observed for this graph, proves the efficiency of the task decomposition strategy implemented in the Mozart system. This observation is confirmed by the results of tests shown in the pie chart on the right side of Fig. 3. The chart considers the computational environment consisting of five workers. The work granularity is measured by the number of vertices of the search tree explored by one worker in relation to the total size of the tree. The result for every test is taken as an arithmetic mean of seven runs.

It turns out, that in the given environment, the work granularity almost exclusively depends on the computational power of workers and it is nearly not affected by the input data. For every test graph, the work granularity taken for each worker does not differ from the results given in the chart by more than 2%. This indicates that the task decomposition strategy, that is, the strategy of the search tree exploration implemented in the Mozart system may efficiently adapt a load of the individual worker to its computational capabilities.

4 Final Remarks

We show that the problem of a vertex-magic total labeling of a graph is easily expressible by constraint programming paradigm in the Mozart system. The program may be executed both in sequence and in parallel. The transition from the sequential to the parallel version practically does not need any modifications of the code and it noticeably decreases the computational time. This work was supported by Poznań University of Technology, the grant DS 45-083/2005.

References

1. Apt, K.R.: Principles of Constraint Programming. Cambridge Univ. Press (2003)
2. Gallian, J. A.: A dynamic survey of graph labeling. El. J. Combin. **5(1)** (1998) 1-43
3. MacDougall, J.A., Miller, M., Slamin, Wallis, W.D.: Vertex-magic total labelings of graphs. Utilitas Mathematica **61** (2002) 3-21
4. Schulte, Ch.: Programming Constraint Services, PhD thesis. University of Saarlandes (2000)
5. Smolka, G.: The Oz Programming Model. Computer Science Today. LNCS **1000** (1995) 324–343
6. Van Hentenryck, P.: Constraint Satisfaction in Logic Programming. Programming Logic Series. The MIT Press (1989)
7. The Mozart Programming System, http://www.mozart-oz.org (2004)

A Parallel Numerical Library for Co-array Fortran

Robert W. Numrich

Minnesota Supercomputing Institute, University of Minnesota,
Minneapolis, MN 55455 USA

Abstract. This paper describes a parallel numerical library based on Co-array Fortran syntax in combination with the object-oriented features of Fortran 95. It defines distributed data structures based on an abstract object called a vector map. It uses co-array syntax, embedded in methods associated with distributed objects, for communication between those objects based on information in the vector map. It applies a finite difference operator to the shallow water equations to illustrate how to use the library to calculate solutions for partial differential equations.

1 Introduction

This paper describes how to combine the parallel extensions of Co-array Fortran [15, 16, 18] with the object-oriented features of Fortran 95 [1] to implement a class library of distributed objects with a set of associated parallel methods. Although Fortran 95 is not a true object-oriented language, it nevertheless includes many object-oriented features that allow the programmer to emulate object-oriented techniques. Many programmers have used these techniques for serial codes [1, 4, 7, 6, 11, 12, 13]. When compilers become available for Fortran 2003, the latest standard for the language [5, 17], more programmers will use them because Fortran 2003 contains additional object-oriented features. Furthermore, co-arrays have been added to the next standard, beyond Fortran 2003 [16], which will make them available to everyone on every platform.

This paper extends the object-oriented techniques used for serial codes to parallel codes. It defines distributed objects based on a new object called a vector map, which keeps track of the relationship between the global view of a distributed object and the local view of the same object. Co-array syntax provides communication between distributed objects based on the information contained in the vector map. Distributed objects are derived types that are organized into modules, which also contain related procedures, so that they resemble classes. The programmer declares an object, by name and type, and creates a specific instance of that object by calling a constructor. Constructors have alternative interfaces that provide different data distributions depending on the actual arguments presented to them.

Figure 1 shows vector maps at the center of the library. Vector maps together with vector and matrix objects combine to build distributed objects called block

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 960–969, 2006.

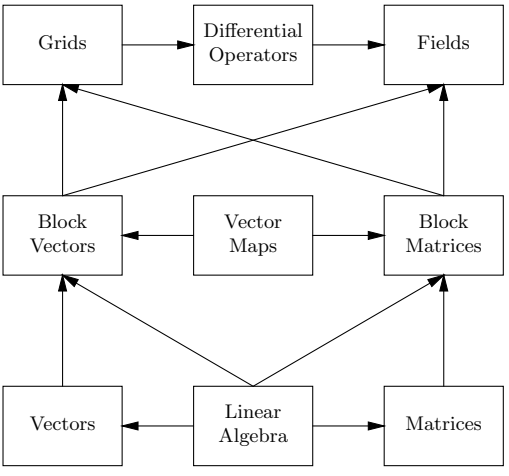


Fig. 1. The Library's Structure. At the base are vector and matrix objects, fundamental objects for building more complicated structures like block-vector objects and block-matrix objects. Vector maps, the central objects of the library, contain information describing the relationship between the global view and the local view of block-vector objects and, by extension, of block-matrix objects. Grid and field objects extend block-vector and block-matrix objects. Parallel procedures for linear algebra and finite difference operators use co-array syntax for communication.

vectors and block matrices. These objects, in turn, combine together to build grid and field objects suitable for solving differential equations. Procedures provided in the library, such as linear algebra operations for vector and matrix objects or halo exchange operations for field objects, use information in the vector map and co-array syntax to perform communication operations between objects.

The library encapsulates the details of data decomposition inside constructors with procedure interfaces based on named derived types. Although the Fortran language is not case-sensitive, the library adopts a naming convention that roughly follows the convention used in some object-oriented languages, such as Java [19, Appendix A]. The names of derived types begin with uppercase letters. For long names containing more than one noun, each noun starts with an uppercase letter. For example, vector maps are objects based on a derived type named `VectorMap`; a block vector, with eight-byte precision, is an object based on a derived type named `BlockR8Vector`. A procedure name begins with a lowercase letter, usually with a suggestion of its function, followed by the name of the object to which the procedure applies. For example, the constructor for a block vector has the name `newBlockVector`.

Fortran 2003 includes a way to associate procedures directly with objects [5, 17], but Fortran 95 does not. The library uses the first argument of each procedure to make the association, a convention adopted almost universally by Fortran programmers [1]. The library enforces the association through generic interfaces.

2 Vector Maps

The most important object in the library, as implied by its central position in Figure 1, is the vector map [14]. A vector map is an abstract description of the relationship between the global view of a vector and the local view of the same vector. It holds none of the data associated with the vector nor any information about how work should be performed on the data.

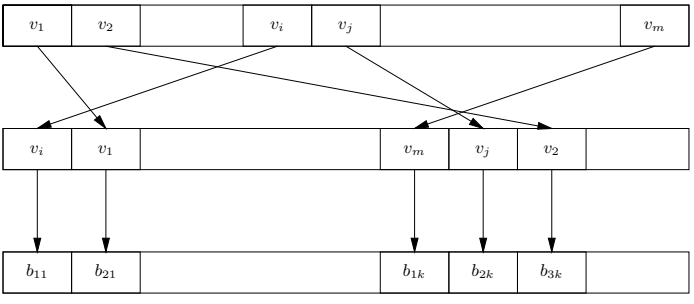


Fig. 2. A Vector Map. The top horizontal strip represents a global vector v cut into blocks labeled by a global block index, $v_i, i = 1, m$. The second horizontal strip represents the global blocks permuted into some other order. The third horizontal strip represents the assignment of these permuted global blocks in groups to images (processors), $p_k, k = 1, P$. Each image contains its own collection of local blocks labeled by local block indices. In this example, image p_1 holds two global blocks, v_i, v_1, relabeled as local blocks b_{11}, b_{21}. Image p_k holds three global blocks, v_m, v_j, v_2, relabeled as local blocks b_{1k}, b_{2k}, b_{3k}.

Figure 2 shows the general form of a vector map. The top horizontal strip represents a vector $\mathbf{v}(:)$ of length n, the global view of the vector. A vector map cuts the global vector into a set of m blocks, $\mathbf{v}_i$, labeled with a global block index, $i = 1, m$. These blocks are vectors, $\mathbf{v}_i(:)$, of length n_i such that $\sum_i n_i = n$.

The vector map reorders the global blocks according to some permutation, provided by the programmer, as represented by the middle horizontal strip. It assigns a fixed number of these blocks to each program image, as specified by the programmer, taking them in order from the permuted set. Each program image p_k owns a set of local blocks, $\{\mathbf{b}_{1k}, \mathbf{b}_{2k}, \ldots\}$, relabeled with two indices, the local block index first and the image number second. Each local block is a vector $\mathbf{b}_{ik}(:)$ with length n_{ik} such that $\sum_{ik} n_{ik} = n$. The programmer retrieves information about the relationship between the global view at the top of Figure 2 and the local view at the bottom through a set of inquiry functions associated with the map.

Vector maps are general enough to describe any distribution the programmer might like to use. For example, the number of global blocks may equal the number of images and the permutation may be the identity. Then each image owns a single block with the global block number equal to the image number. Or the

programmer may want to use a block-cyclic distribution with a round-robin distribution of the blocks among the images. In that case, the number of blocks equals the number of images, and the global block size equals the vector length divided by the number of images. If the global block size is one, the map assigns the elements of the global vector one by one to the images in a round-robin fashion. Or the permutation might reverse the global indices so that the map distributes the elements back to front. The global blocks need not be the same size, and each image can own a different number of local blocks. In fact, some images may own none or one image may own them all.

All the components of a vector map are private to the data structure. The programmer creates a vector map by calling a constructor that builds a map for a particular decomposition. For example, one form of the constructor has the interface,

```
call newVectorMap(map,n,k,p)
```

It generates a map corresponding to the important special case of block-cyclic distribution [9]. The global vector length is `n`, the global blocks are of size `k`, and the number of images is `p`.

Vector maps induce maps for higher objects. For example, the library builds distributed matrix objects using one vector map for the row dimension and another vector map for the column dimension [8, 9, 10, 14].

3 Vector and Matrix Objects

Although the Fortran language represents vectors and matrices as intrinsic types, there are good reasons to define derived types for these objects. For example, the library includes the precision of the data in the name of the object. The class `R8Vector` contains eight-byte real data compared with the class `C4Vector` with four-byte complex data. The object contains information about the upper and lower bounds of the vector and information about halo cells added to each end. The data structure can also hold information about the nature of the object, its color or spin, for example, related to particular field properties.

The library defines objects as named derived types and provides a set of well-defined procedures for the programmer to manipulate the objects. The Vector Class, for example, has the following data structure,

```
type XXVector
  real(kind=XX),allocatable,target :: vector(:)
  type(VectorBounds)               :: bounds
end type XXVector
```

Data is contained in the array component `vector(:)` with precision represented by the symbol `XX` in the set {`R4,R8,C4,C8,I4,I8`}. The data structure

```
type VectorBounds
  private
```

```
  integer :: lowerBound,upperBound,haloWidth
end type VectorBounds
```

has private components to prevent the programmer from corrupting the object.

To create a vector object, the programmer must use a constructor, for example,

```
call newXXVector(v,n1,n2,w)
```

The constructor sets `bounds%lowerBound` to `n1`, `bounds%upperBound` to `n2`, and `bounds%haloWidth` to `w`. It allocates the vector array component with dimensions `v%vector(n1-w:n2+w)` and sets the array to zero, including the halos.

To manipulate a vector object, the programmer uses procedures associated with the object to obtain information about it. For example, to obtain a copy of the internal data in vector object `v`, the programmer might write

```
n1 = getLowerBound(v)
n2 = getUpperBound(v)
vCopy(n1:n2) = v%vector(n1:n2)
```

Or the programmer may request a pointer into the data without making a copy,

```
vPtr => getPointerToData(v)
```

which allows the programmer to manipulate the data directly without the overhead of an extra copy.

The Matrix Class has an obvious structure that mimics the Vector Class,

```
type XXMatrix
  real(kind=XX),allocatable,target :: matrix(:,:)
  type(MatrixBounds)               :: bounds
end type XXMatrix
```

A call to the constructor,

```
call newXXMatrix(a,m1,m2,n1,n2,w1,w2)
```

creates a matrix object with bounds `m1,m2` for the row dimension, `n1,n2` for the column dimension, and two halo widths, `w1,w2`, for the rows and columns. The constructor allocates, and sets to zero, the two-dimensional array component `a%matrix(m1-w2:m2+w2,n1-w1:n2+w1)`, and sets the appropriate values in the `MatrixBounds` component.

Vector and matrix objects may be co-arrays, for example,

```
type(R8Vector) :: v[*]
type(R8Matrix) :: a[*]
```

Each image calls its own constructor to build its own local vector or matrix object independent of the other images. The programmer supplies appropriate synchronization before reading or writing remote data. To read data from another image, the programmer can write something like,

```
x(:,:) = a[q]%matrix(:,:)
```

which makes a local copy `x(:,:)` of the data contained in `a%matrix(:,:)` from image `[q]`.

4 Block Vectors and Block Matrices

The library's power resides in objects called Block Vectors and Block Matrices, which combine vector and matrix objects with vector maps. They are distributed data structures that contain within themselves not only a description of the data distribution but also a collection of associated procedures, which allow the programmer to manipulate the objects and to handle communication between objects on different images.

The Block Vector object contains a VectorMap to describe the data distribution and a Vector to hold the data itself,

```
type BlockXXVector
  type(VectorMap)            :: map
  type(XXVector),allocatable :: block(:)
  !-other components-!
end type BlockXXVector
```

A call to the constructor,

```
call newBlockXXVector(v,n,k,p,w)
```

creates a block vector of global length `n` across `p` images with block size `k`. The lower global index is one, by default, and the upper global index is `n`. The constructor builds an internal vector map using a constructor that, for this case, corresponds to a block-cyclic distribution. Each image holds some number, perhaps zero, of local blocks. The constructor allocates an array of vector objects `v%block(:)`, one for each local block owned by an image, and allocates space for each local block`block(k)%vector(:)` according to the block size. The lower index for each local block is one and each block has its own upper index. Each local block has extra halo cells of width `w` on both sides. The constructor sets all local blocks to zero, including the halos.

The internal vector map contains all the information necessary to describe the relationship between the global view and the local view of a vector. A vector is an abstract tensor object, $\mathbf{v}$, with global elements, v^i, whose numerical values are determined by the choice of global basis vectors [14]. A block vector converts a global vector with just one index to a collection of local vector with three indices [8, 9],

$$\mathtt{v(i)} \leftrightarrow \mathtt{v(j,b,q)}$$

representing the assignment of element `i` of the vector to element `j` in block `b` on image `q`. In co-array syntax, this mapping has the transparent representation,

$$\mathtt{v(i)} \leftrightarrow \mathtt{v[q]\%block(b)\%vector(j)}$$

In the same way, the Block Matrix object,

```
type BlockXXMatrix
private
  type(VectorMap)            :: rowMap
  type(VectorMap)            :: colMap
  type(XXMatrix),allocatable :: block(:,:)
  !-other components-!
end type BlockXXMatrix
```

contains two internal vector maps, one for the row indices and one for the column indices. A call to the constructor,

```
call newBlockXXMatrix1(a,m,n,k,l,p,q,w1,w2)
```

creates a block-cyclic distribution in both the row and column directions in the same way as for block vectors. The constructor builds the internal vector maps to represent the row and the column decompositions. The argument `m` is the global row dimension, which is cut into blocks of size `k` and distributed across `p` images. The argument `n` is the global column dimension, which is cut into blocks of size `l` and distributed across `q` images. The lower bounds are one by default for both global and local indices. Argument `w1` is the halo width added to both ends of the column dimension for each local block. Argument `w2` is the halo width added to both ends of the row dimension for each local block.

After invocation of the constructor, each image holds its own collection of local blocks, as an array of matrix objects `a%block(:,:)`, with space for local block `(b,c)` allocated in `a%block(b,c)%matrix(:,:)`. The programmer can determine how many local blocks each image holds and how they are related to each other by invoking a set of procedures, which provide access to the components of the data structure. If the block matrix `a` is a co-array, all images must call the constructor before any remote data references are made. The programmer supplies appropriate synchronization before trying to move data from one image to another.

5 Field Objects

The library supports distributed Field Objects, at the top of Figure 1, based on block-vector and block-matrix objects at lower levels. The application of a finite difference scheme to solve the one-dimensional shallow water equations illustrates how to use the library. The partial differential equations, defined by Cahn [2] and by Arakawa and Lamb [3], for the surface height $h(x,t)$ and the two velocity components $u(x,t)$ and $v(x,t)$, as functions of the space variable x and the time variable t, are the equations [3, eqs. 23-25, p. 183]

$$\frac{\partial u}{\partial t} - Fv + G\frac{\partial h}{\partial x} = 0$$

$$\frac{\partial v}{\partial t} + Fu = 0$$

$$\frac{\partial h}{\partial t} + H\frac{\partial u}{\partial x} = 0.$$

In these equations, F is the Coriolis frequency, G is the acceleration of gravity, and H is the mean height of the surface, which is assumed to be small relative to the width of the space interval.

The fields `u,v,h` are block vectors declared as co-arrays:

```
type(BlockR8Vector),dimension[*] :: h,u,v
call newBlockVector(h,n,k,p,w)
call newBlockVector(u,getVectorMap(h))
call newBlockVector(v,getVectorMap(h))
call setBlockVector(h,h0)
```

The first call to the constructor creates the field `h` with the number of grid points `n` needed to represent the finite difference operator, with a block size `k`, and with the number of images `p` over which to distribute the field. The halo width `w` equals one, wide enough for a two-point stencil for the first-order difference operator. A call to an alternative form of the constructor, one that accepts a predefined vector map as a second argument, creates the fields `u` and `v`. The procedure `getVectorMap(h)` returns the map created by the constructor for field `h`. Use of the second form of the constructor avoids the overhead of building the map more than once and guarantees that all three fields have the same distribution. The procedure `setBlockVector(h,h0)` sets the field `h` from the input array `h0(:)`, which contains its initial values.

Having created field objects, the programmer decides to let each image perform work on the local blocks that it owns. For each of its local blocks, an image obtains the length of the block and a pointer into the block, with or without halos depending on how it is used in the difference formula. Each image performs the appropriate finite difference operation independently of the others. Synchronization among images occurs within the halo exchange operation, which uses co-array syntax internally to update overlapping halo regions. With a loop over some predetermined number of time steps, `tMax`, the code might look like the following:

```
do t=1,tMax
  do b=1,getNumLocalBlocks(u)
    m = getLocalBlockLength(u,b)
    hPtr => pointerToLocalBlock(h,b)
    uPtr => pointerToLocalBlockwithHalo(u,b)
    hPtr(1:m) = hPtr(1:m) - 0.5*H*(dt/dx)*(uPtr(2:m+1)-uPtr(0:m))
  enddo
  call cyclicHaloExchange(h)
  do b=1,getNumLocalBlocks(u)
    m = getLocalBlockLength(u,b)
    hPtr => pointerToLocalBlockwithHalo(h,b)
    uPtr => pointerToLocalBlock(u,b)
    vPtr => pointerToLocalBlock(v,b)
```

```
    uPtr(1:m) = uPtr(1:m) + F*dt*vPtr(1:m) &
                          - 0.5*G*(dt/dx)*(hPtr(2:m+1)-hPtr(0:m))
    vPtr(1:m) = vPtr(1:m) - F*dt*uPtr(1:m)
  enddo
  call cyclicHaloExchange(u)
  call cyclicHaloExchange(v)
enddo
```

The library hides from the programmer all the details of data distribution and all the details of how to exchange data between objects. The block vector objects themselves contain all the necessary information, and the procedures associated with them know how to perform the required operations.

6 Summary

This paper has described a parallel numerical library that combines the object-oriented features of Fortran 95 with the parallel syntax of Co-array Fortran. The library contains many other procedures associated with block-vector and block-matrix objects, which cannot be described in detail here. For example, it includes parallel matrix transpose, matrix multiplication and LU decomposition procedures for block matrices. The performance of these procedures is discussed in a separate paper [14]. The library also includes reduction procedures to compute the global minimum, global maximum, and global sum of distributed objects. It also contains various norms for distributed objects, for example, the L-1 and L-2 norms. The library currently does not support sparse vector or matrix objects, which are candidates for future extensions.

Acknowledgements

The author acknowledges support for this research through grant DE-FC02-01ER25505 from the U.S. Department of Energy as part of the Center for Programming Models for Scalable Parallel Computing, sponsored by the Office of Science, and through an appointment as a Goddard Visiting Fellow from the NASA Goddard Earth Sciences and Technology Center.

References

1. J. E. Akin. *Object-oriented programming via Fortran 90/95*. Cambridge University Press, 2003.
2. Albert Cahn, Jr. An investigation of the free oscillations of a simple current system. *Journal of Meteorology*, 2(2):113–119, June 1945.
3. Akio Arakawa and Vivian R. Lamb. Computational design of the basic dynamical processes of the UCLA general circulation model. *Methods in Computational Physics*, 17:173–265, 1977.

4. Malcolm Cohen. Information technology-programming languages-Fortran-enhanced data type facilities. Technical Report ISO/IEC TR 15581(E), ISO/IEC, Geneva, 2001.
5. Malcolm Cohen. Fortran 2003: Into the Future. *Dr. Dobb's Journal*, pages 50–56, July 2004.
6. V. K. Decyk, C. D. Norton, and B. K. Szymanski. How to express C++ concepts in Fortran 90. *Scientific Programming*, 6(4):363, 1998.
7. Viktor K. Decyk, Charles D. Norton, and Boleslaw K. Szymanski. How to support inheritance and run-time polymorphism in Fortran 90. *Computer Physics Communications*, 115:9–17, 1998.
8. Jack J. Dongarra, Robert A. van de Geijn, and David W. Walker. Scalability issues affecting the design of a dense linear algebra library. *Journal of Parallel and Distributed Computing*, 22:523–537, 1994.
9. Jack J. Dongarra and David W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2):151–180, June 1995.
10. Carter Edwards, Po Geng, Abani Patra, and Robert van de Geijn. Parallel matrix distributions: Have we been doing it all wrong? Technical Report PLAPACK Working Note No. 1, TR-95-39, Department of Computer Sciences, University of Texas at Austin, October 1995.
11. Mark G. Gray and Randy M. Roberts. Object-based programming in Fortran 90. *Computers in Physics*, 11(4):355–361, July-August 1997.
12. L. Machiels and M. O. Deville. Fortran 90: An entry to object-oriented programming for the solution of partial differential equations. *ACM Transactions on Mathematical Software*, 23(1):32–49, March 1997.
13. C.D. Norton, V.K. Decyk, and B.K. Szymanski. High performance object-oriented scientific programming in Fortran 90. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM Activity Group on Supercomputing, Society for Industrial and Applied Mathematics, March 1997. CD ROM format.
14. Robert W. Numrich. Parallel numerical algorithms based on tensor notation and Co-Array Fortran syntax. *Parallel Computing*, 31:588–607, 2005.
15. Robert W. Numrich and John K. Reid. Co-Array Fortran for parallel programming. *ACM Fortran Forum*, 17(2):1–31, 1998.
16. Robert W. Numrich and John K. Reid. Co-arrays in the next Fortran standard. *ACM Fortran Forum*, 2005.
17. John Reid. The future of Fortran. *Computing in Science and Engineering*, 5(3): 59–67, July/August 2003.
18. John K. Reid and Robert W. Numrich. Co-Array Fortran for parallel programming. Technical Report JTC1/SC22/WG5 N1592 (http://www.nag.co.uk/sc22wg5/ftp.html), ISO/IEC, Geneva, May, 2004.
19. Brett Spell. *Professional Java Programming*. Wrox Press Ltd., Birmingham, UK, 2000.

A Hybrid MPI/OpenMP Implementation of a Parallel 3-D FFT on SMP Clusters

Daisuke Takahashi

Graduate School of Systems and Information Engineering,
University of Tsukuba, 1-1-1 Tennodai,
Tsukuba, Ibaraki 305-8573, Japan
daisuke@cs.tsukuba.ac.jp

Abstract. In the present paper, we propose a hybrid MPI/OpenMP implementation of a parallel three-dimensional fast Fourier transform (FFT) algorithm on SMP clusters. The three-dimensional FFT algorithm can be altered to create a block three-dimensional FFT algorithm in order to reduce the number of cache misses. We then use the obtained block three-dimensional FFT algorithm to implement the parallel three-dimensional FFT. We succeeded in obtaining a performance of over 14 GFLOPS on the AIST Super Cluster M-64 (using 32 nodes out of 132 available, Itanium2 1.3 GHz, 4-way SMP).

1 Introduction

The fast Fourier transform (FFT) [1] is an algorithm that is widely used today in science and engineering. Parallel three-dimensional FFT algorithms on distributed-memory parallel computers have been investigated extensively [2, 3, 4, 5]. The hybrid MPI/OpenMP model is a natural parallel programming paradigm for emerging parallel architectures that are based on clusters of SMPs.

In the present paper, we propose a hybrid MPI/OpenMP implementation of a parallel three-dimensional FFT on clusters of SMPs. Our hybrid implementation of the parallel three-dimensional FFT is based on a blocking algorithm for a parallel three-dimensional FFT [5]. The block three-dimensional FFT algorithm combines multicolumn FFTs [6, 7] and transpositions to reduce the number of cache misses.

We have implemented the parallel block three-dimensional FFT algorithm on a 32-node quad Itanium2 SMP cluster, and the obtained performance results are reported herein.

Section 2 describes the conventional three-dimensional FFT algorithm. In Section 3, we propose a block three-dimensional FFT algorithm, which is used for problems that exceed the cache size. In Section 4, we propose a parallel block three-dimensional FFT algorithm. Section 5 describes the in-cache FFT algorithm used for problems that fit into the data cache. Section 6 presents the performance results. Finally, in Section 7, we provide concluding remarks.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 970–977, 2006.

2 Three-Dimensional FFT

The three-dimensional discrete Fourier transform (DFT) is given by

$$y(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2}^{j_2 k_2} \omega_{n_1}^{j_1 k_1}, \tag{1}$$

where $\omega_{n_r} = e^{-2\pi i/n_r}$ $(1 \le r \le 3)$ and $i = \sqrt{-1}$.

The three-dimensional FFT based on the multicolumn FFT algorithm is as follows:

Step 1: Transpose

$$x_1(j_3, j_1, j_2) = x(j_1, j_2, j_3).$$

Step 2: $n_1 n_2$ individual n_3-point multicolumn FFTs

$$x_2(k_3, j_1, j_2) = \sum_{j_3=0}^{n_3-1} x_1(j_3, j_1, j_2) \omega_{n_3}^{j_3 k_3}.$$

Step 3: Transpose

$$x_3(j_2, j_1, k_3) = x_2(k_3, j_1, j_2).$$

Step 4: $n_1 n_3$ individual n_2-point multicolumn FFTs

$$x_4(k_2, j_1, k_3) = \sum_{j_2=0}^{n_2-1} x_3(j_2, j_1, k_3) \omega_{n_2}^{j_2 k_2}.$$

Step 5: Transpose

$$x_5(j_1, k_2, k_3) = x_4(k_2, j_1, k_3).$$

Step 6: $n_2 n_3$ individual n_1-point multicolumn FFTs

$$y(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} x_5(j_1, k_2, k_3) \omega_{n_1}^{j_1 k_1}.$$

The distinctive features of the three-dimensional FFT can be summarized as follows:

- Three multicolumn FFTs are performed in Steps 2, 4 and 6. Each column FFT is small enough to fit into the data cache.
- The three-dimensional FFT has three transpose steps, which typically are the chief bottlenecks in cache-based processors.

3 A Block Three-Dimensional FFT Algorithm

We combine the multicolumn FFTs and transpositions to reduce the number of cache misses, and we modify the conventional three-dimensional FFT algorithm to reuse the data in the cache memory. As in the conventional three-dimensional FFT described above, it is assumed in the following that $n = n_1 n_2 n_3$ and that

```
      COMPLEX*16 X(N1,N2,N3)
      COMPLEX*16 YWORK(N2+NP,NB),ZWORK(N3+NP,NB)
!$OMP PARALLEL
!$OMP DO
      DO J=1,N2
        DO II=1,N1,NB
          DO KK=1,N3,NB
            DO I=II,MIN0(II+NB-1,N1)
              DO K=KK,MIN0(KK+NB-1,N3)
                ZWORK(K,I-II+1)=X(I,J,K)
              END DO
            END DO
          END DO
          DO I=II,MIN0(II+NB-1,N1)
            CALL IN_CACHE_FFT(ZWORK(1,I-II+1),N3)
          END DO
          DO K=1,N3
            DO I=II,MIN0(II+NB-1,N1)
              X(I,J,K)=ZWORK(K,I-II+1)
            END DO
          END DO
        END DO
      END DO
!$OMP DO
      DO K=1,N3
       DO II=1,N1,NB
         DO JJ=1,N2,NB
           DO I=II,MIN0(II+NB-1,N1)
             DO J=JJ,MIN0(JJ+NB-1,N2)
               YWORK(J,I-II+1)=X(I,J,K)
             END DO
           END DO
         END DO
         DO I=II,MIN0(II+NB-1,N1)
           CALL IN_CACHE_FFT(YWORK(1,I-II+1),N2)
         END DO
         DO J=1,N2
           DO I=II,MIN0(II+NB-1,N1)
             X(I,J,K)=YWORK(J,I-II+1)
           END DO
         END DO
       END DO
       DO J=1,N2
         CALL IN_CACHE_FFT(X(1,J,K),N1)
       END DO
      END DO
!$OMP END PARALLEL
```

Fig. 1. A block three-dimensional FFT algorithm

n_b is the block size. In addition, each processor is assumed to have a multi-level cache memory. A block three-dimensional FFT algorithm can be written as follows:

1. Consider the data in main memory as an $n_1 \times n_2 \times n_3$ complex matrix. Fetch and transpose the data n_b rows at a time into an $n_3 \times n_b$ matrix. The $n_3 \times n_b$ array fits into the L2/L3 cache.
2. For each group of n_b columns, perform n_b individual n_3-point multicolumn FFTs on the $n_3 \times n_b$ array in the L2/L3 cache. Each column FFT also fits into the L1 data cache.
3. Transpose each of the resulting $n_3 \times n_b$ matrices, and return the resulting n_b rows to the same locations in the main memory from which they were fetched.
4. Fetch and transpose the data n_b rows at a time into an $n_2 \times n_b$ matrix.
5. For each group of n_b columns, perform n_b individual n_2-point multicolumn FFTs on the $n_2 \times n_b$ array in the L2/L3 cache.
6. Transpose each of the resulting $n_2 \times n_b$ matrices, and return the resulting n_b rows to the same locations in the main memory from which they were fetched.
7. Perform $n_2 n_3$ individual n_1-point multicolumn FFTs on the $n_1 \times n_2 \times n_3$ array.

We note here that this algorithm is a *three-pass* algorithm. Figure 1 gives the pseudo-code for this block three-dimensional FFT algorithm. Here, the arrays `YWORK` and `ZWORK` are the work arrays. The parameters `NB` and `NP` are the blocking parameter and the padding parameter, respectively.

4 Parallel Block Three-Dimensional FFT Algorithm

Let $N = N_1 \times N_2 \times N_3$. On a distributed-memory parallel computer having P processors, a three-dimensional array $x(N_1, N_2, N_3)$ is distributed along the third dimension N_3. If N_3 is divisible by P, then each processor has distributed data of size N/P. Next, we introduce the notation $\hat{N}_r \equiv N_r/P$ and denote the corresponding index as $\hat{J}_r$, which indicates that the data along J_r are distributed across all P processors. Here, we use the subscript r to indicate that this index belongs to dimension r. The distributed array is represented as $\hat{x}(N_1, N_2, \hat{N}_3)$. At processor m, the local index $\hat{J}_r(m)$ corresponds to the global index as the *block* distribution:

$$J_r = m \times N_r + \hat{J}_r(m), \quad 0 \le m \le P-1, \quad 1 \le r \le 3. \tag{2}$$

In order to illustrate the all-to-all communication, it is convenient to decompose N_i into two dimensions, $\tilde{N}_i$ and P_i, where $\tilde{N}_i \equiv N_i/P_i$. Although P_i is the same as P, we use the subscript i to indicate that this index belongs to dimension i.

Starting with the initial data $\hat{x}(N_1, N_2, \hat{N}_3)$, the parallel three-dimensional FFT can be performed according to the following steps:

Step 1: $N_2 \cdot (N_3/P)$ individual N_1-point multicolumn FFTs

$$\hat{x_1}(K_1, J_2, \hat{J}_3) = \sum_{J_1=0}^{N_1-1} \hat{x}(J_1, J_2, \hat{J}_3)\omega_{N_1}^{J_1 K_1}.$$

Step 2: Transpose

$$\hat{x_2}(J_2, K_1, \hat{J}_3) = \hat{x_1}(K_1, J_2, \hat{J}_3).$$

Step 3: $N_1 \cdot (N_3/P)$ individual N_2-point multicolumn FFTs

$$\hat{x_3}(K_2, K_1, \hat{J}_3) = \sum_{J_2=0}^{N_2-1} \hat{x_2}(J_2, K_1, \hat{J}_3)\omega_{N_2}^{J_2 K_2}.$$

Step 4: Rearrangement

$$\begin{aligned}\hat{x_4}(\tilde{K_1}, K_2, \hat{J}_3, P_1) &= \hat{x_3}(K_2, \tilde{K_1}, P_1, \hat{J}_3) \\ &\equiv \hat{x_3}(K_2, K_1, \hat{J}_3).\end{aligned}$$

Step 5: All-to-all communication

$$\hat{x_5}(\hat{K_1}, K_2, \tilde{J}_3, P_3) = \hat{x_4}(\tilde{K_1}, K_2, \hat{J}_3, P_1).$$

Step 6: Rearrangement

$$\begin{aligned}\hat{x_6}(J_3, \hat{K_1}, K_2) &\equiv \hat{x_6}(\tilde{J}_3, P_3, \hat{K_1}, K_2) \\ &= \hat{x_5}(\hat{K_1}, K_2, \tilde{J}_3, P_3).\end{aligned}$$

Step 7: $(N_1/P) \cdot N_2$ individual N_3-point multicolumn FFTs

$$\hat{x_7}(K_3, \hat{K_1}, K_2) = \sum_{J_3=0}^{N_3-1} \hat{x_6}(J_3, \hat{K_1}, K_2)\omega_{N_3}^{J_3 K_3}.$$

Step 8: Transpose

$$\hat{y}(\hat{K_1}, K_2, K_3) = \hat{x_7}(K_3, \hat{K_1}, K_2).$$

The distinctive features of the parallel three-dimensional FFT algorithm can be summarized as follows:

- The parallel three-dimensional FFT is accompanied by a local transpose (data rearrangement).
- $N^{2/3}/P$ individual $N^{1/3}$-point multicolumn FFTs are performed in Steps 1, 3 and 7 for the case of $N_1 = N_2 = N_3 = N^{1/3}$.
- The all-to-all communication occurs just once.

If both N_1 and N_3 are divisible by P, then the workload on each processor is also uniform.

Although the input data $\hat{x}(N_1, N_2, \hat{N_3})$ is distributed along the third dimension N_3, the output data $\hat{y}(\hat{N_1}, N_2, N_3)$ is distributed along the first dimension N_1. If we assume that the input data and output data have the same distribution, then an additional all-to-all communication step is needed.

5 In-Cache FFT Algorithm

We use the radix-2, 4 and 8 Stockham autosort algorithm [8] for in-cache FFT.

Although the Stockham autosort algorithm requires a scratch array of the same size as the input data array, digit-reverse permutation is unnecessary. If the Stockham autosort algorithm is used for the individual FFTs, then the additional scratch requirement for performing the individual FFTs is $O(N^{1/3})$ (where $N = N_1 \times N_2 \times N_3$) at most.

We inserted OpenMP directives to the proposed block three-dimensional FFT. The outermost loop of each FFT step shown in Fig. 1 is distributed across the processors. Each directive of OpenMP may cause an overhead. In order to reduce the fork/join overhead, three parallel regions can be fused, as shown in Fig. 1.

6 Performance Results

In order to evaluate the proposed parallel three-dimensional FFT, called FFTE (version 4.0), we compared its performance to that of the FFTW library (version 2.1.5) [9], which is known to be one of the fastest FFT libraries for many processors. Although the latest version of FFTW is version 3.0.1, MPI parallel transforms are only available in version 2.1.5.

We averaged the elapsed times obtained from 10 executions of complex forward FFTs. The parallel FFTs were performed on double-precision complex data, and the table for twiddle factors was prepared in advance. We used transposed order output to reduce the all-to-all communication step for the FFTE and the FFTW.

The AIST Super Cluster M-64 (132 node, quad Itanium2 1.3 GHz, 16 KB L1 instruction cache, 16 KB L1 data cache, 256 KB L2 cache, 3 MB L3 cache, 16 GB main memory per node, Linux 2.4.21) was used as clusters of SMPs. In the

Table 1. Performance of parallel three-dimensional FFTs on the AIST Super Cluster M-64 (Itanium2 1.3 GHz, 4-way SMP)

P (Nodes × CPUs)	$N_1 \times N_2 \times N_3$	FFTE 4.0 (Hybrid)		FFTE 4.0 (MPI)		FFTW 2.1.5	
		Time	MFLOPS	Time	MFLOPS	Time	MFLOPS
1×1	$2^8 \times 2^9 \times 2^9$	16.84255	517.98	16.58638	525.98	33.59739	259.67
1×2	$2^8 \times 2^9 \times 2^9$	9.57885	910.77	8.92122	977.91	21.98175	396.88
1×4	$2^8 \times 2^9 \times 2^9$	6.44173	1354.32	5.54898	1572.21	17.03849	512.03
2×1	$2^9 \times 2^9 \times 2^9$	19.67159	921.09	19.28121	939.74	45.80216	395.60
2×2	$2^9 \times 2^9 \times 2^9$	12.83536	1411.68	10.60596	1708.42	29.34613	617.44
2×4	$2^9 \times 2^9 \times 2^9$	9.77419	1853.80	6.69942	2704.62	21.22772	853.57
4×1	$2^9 \times 2^9 \times 2^{10}$	20.90372	1797.81	20.95441	1793.46	56.67690	663.07
4×2	$2^9 \times 2^9 \times 2^{10}$	14.27743	2632.19	13.53452	2776.67	37.50561	1002.01
4×4	$2^9 \times 2^9 \times 2^{10}$	11.34989	3311.13	8.46259	4440.84	26.54643	1415.67
8×1	$2^9 \times 2^{10} \times 2^{10}$	22.97201	3388.74	25.54573	3047.33	58.68224	1326.57
8×2	$2^9 \times 2^{10} \times 2^{10}$	16.20815	4802.91	14.38517	5411.56	43.43497	1792.25
8×4	$2^9 \times 2^{10} \times 2^{10}$	13.03677	5971.29	9.51725	8179.50	27.83126	2797.08
16×1	$2^{10} \times 2^{10} \times 2^{10}$	30.33136	5310.06	34.05250	4729.79	65.79830	2447.80
16×2	$2^{10} \times 2^{10} \times 2^{10}$	22.75091	7079.33	21.49411	7493.27	51.58395	3122.31
16×4	$2^{10} \times 2^{10} \times 2^{10}$	19.33317	8330.83	16.04758	10036.48	34.95780	4607.31
32×1	$2^{10} \times 2^{10} \times 2^{11}$	36.71174	9066.85	40.87128	8144.10	71.18536	4675.96
32×2	$2^{10} \times 2^{10} \times 2^{11}$	28.60080	11638.14	28.58210	11645.75	56.51478	5889.79
32×4	$2^{10} \times 2^{10} \times 2^{11}$	25.10995	13256.10	22.46464	14817.06	40.04432	8312.29

experiment, we used 1 node $\sim$ 32 nodes on the AIST Super Cluster M-64. The nodes on the Itanium SMP cluster are interconnected through a Myrinet-XP switch. MPICH-SCore [10] was used as a communication library.

The Intel C Compiler (`icc`, version 8.1) and the Intel Fortran Compiler (`ifort`, version 8.1) were used on the AIST Super Cluster M-64. For the FFTE (Hybrid), the compiler options used were specified as "`ifort -O3 -openmp`." For the FFTE (MPI), the compiler options used were specified as "`ifort -O3`." For the FFTW, the compiler options used were specified as "`icc -O3`" and "`ifort -O3`."

Table 1 compares the FFTE (Hybrid and MPI) and the FFTW in terms of their run times and MFLOPS. The first column of the table indicates the number of processors. The second column gives the problem size. The next six columns show the average elapsed time in seconds and the average execution performance in MFLOPS. The MFLOPS values are each based on $5N \log_2 N$ for a transform of size $N = 2^m$.

For $N = 2^{10} \times 2^{10} \times 2^{11}$ and $P = 32 \times 4$, the FFTE (MPI) runs approximately 12% faster than the FFTE (Hybrid), as shown in Table 1. Table 2 shows the results of the all-to-all communication timings (pure MPI) on the AIST Super Cluster M-64. The first column of the table indicates the number of processors. The second column gives the problem size. The next two columns show the average elapsed time in seconds and the average bandwidth in MB/sec.

Tables 1 and 2 show that all-to-all communication overhead contributes significantly to the execution time. Table 2 shows that the all-to-all communication performances for 4 CPUs per node are better than those for either 1 CPU per

Table 2. All-to-all communication performance on the AIST Super Cluster M-64 (Itanium2 1.3 GHz, 4-way SMP)

P (Nodes × CPUs)	$N_1 \times N_2 \times N_3$	Time	MB/sec
1×2	$2^8 \times 2^9 \times 2^9$	1.15737	231.94
1×4	$2^8 \times 2^9 \times 2^9$	1.47452	136.54
2×1	$2^9 \times 2^9 \times 2^9$	3.68277	145.78
2×2	$2^9 \times 2^9 \times 2^9$	2.77443	145.13
2×4	$2^9 \times 2^9 \times 2^9$	2.74168	85.67
4×1	$2^9 \times 2^9 \times 2^{10}$	5.76482	139.69
4×2	$2^9 \times 2^9 \times 2^{10}$	4.30683	109.07
4×4	$2^9 \times 2^9 \times 2^{10}$	4.21265	59.74
8×1	$2^9 \times 2^{10} \times 2^{10}$	6.22999	150.81
8×2	$2^9 \times 2^{10} \times 2^{10}$	5.73692	87.73
8×4	$2^9 \times 2^{10} \times 2^{10}$	5.40266	48.13
16×1	$2^{10} \times 2^{10} \times 2^{10}$	12.44766	80.87
16×2	$2^{10} \times 2^{10} \times 2^{10}$	12.28788	42.33
16×4	$2^{10} \times 2^{10} \times 2^{10}$	11.84227	22.31
32×1	$2^{10} \times 2^{10} \times 2^{11}$	17.92775	58.02
32×2	$2^{10} \times 2^{10} \times 2^{11}$	17.73715	29.80
32×4	$2^{10} \times 2^{10} \times 2^{11}$	17.07957	15.59

node or 2 CPUs per node. For the FFTE (Hybrid), all threads except one are idle during MPI all-to-all communication.

Thus, the all-to-all communication time for hybrid implementation is greater than that for pure MPI implementation. Moreover, the hybrid implementation has a high cache miss ratio and a high thread overhead. These are the reasons why using the FFTE (MPI) is more advantageous than using the FFTE (Hybrid). As for related works, the pure MPI model outperformed the hybrid model [11].

On the other hand, compared to the FFTW, the FFTE (MPI) runs approximately 1.8 times faster for $N = 2^{10} \times 2^{10} \times 2^{11}$ and $P = 32 \times 4$. The performance of the FFTE remains at a high level even for the larger problem size, because of cache blocking. This is the reason why the FFTE is more advantageous than the FFTW. These results clearly indicate that the FFTE is superior to the FFTW.

We note that on the AIST Super Cluster M-64, a performance of over 14 GFLOPS was realized with $N = 2^{10} \times 2^{10} \times 2^{11}$ in the FFTE (MPI), as shown in Table 1.

7 Conclusion

We have proposed the hybrid MPI/OpenMP implementation of the parallel three-dimensional FFT algorithm on SMP clusters. We reduced the number of cache misses for the conventional three-dimensional FFT algorithm.

Performance results indicate that the pure MPI model outperformed the hybrid model. The performance of the FFTE is better than that of the FFTW.

We succeeded in obtaining a performance of over 14 GFLOPS on the AIST Super Cluster M-64 (using 32 nodes out of 132 available, Itanium2 1.3 GHz, 4-way SMP).

Acknowledgments

This work was done under the contract with the National Institute of Advanced Industrial Science and Technology (AIST).

The author wishes to thank Dr. Mitsuo Yokokawa, AIST, for his many helpful suggestions for using the AIST Super Cluster.

References

1. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19** (1965) 297–301
2. Agarwal, R.C., Gustavson, F.G., Zubair, M.: An efficient parallel algorithm for the 3-D FFT NAS parallel benchmark. In: Proceedings of the Scalable High-Performance Computing Conference. (1994) 129–133
3. Hegland, M.: Real and complex fast Fourier transforms on the Fujitsu VPP 500. Parallel Computing **22** (1996) 539–553
4. Calvin, C.: Implementation of parallel FFT algorithms on distributed memory machines with a minimum overhead of communication. Parallel Computing **22** (1996) 1255–1279
5. Takahashi, D.: Efficient implementation of parallel three-dimensional FFT on clusters of PCs. Computer Physics Communications **152** (2003) 144–150
6. Bailey, D.H.: FFTs in external or hierarchical memory. The Journal of Supercomputing **4** (1990) 23–35
7. Van Loan, C.: Computational Frameworks for the Fast Fourier Transform. SIAM Press, Philadelphia, PA (1992)
8. Swarztrauber, P.N.: FFT algorithms for vector computers. Parallel Computing **1** (1984) 45–63
9. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for the FFT. In: Proc. 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 98). (1998) 1381–1384
10. Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T., Ishikawa, Y.: High performance communication using a commodity network for cluster systems. In: Proc. Ninth International Symposium on High Performance Distributed Computing (HPDC-9). (2000) 139–146
11. Cappello, F.R., Richard, O., Etiemble, D.: MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks. In: Proc. 2000 ACM/IEEE Conference on Supercompuring (SC2000). (2000)

Safety of an Object-Based Version Vector Consistency Protocol of Session Guarantees*

Jerzy Brzeziński and Cezary Sobaniec

Institute of Computing Science,
Poznań University of Technology, Poland
{Jerzy.Brzezinski, Cezary.Sobaniec}@cs.put.poznan.pl

Abstract. Replication provides high performance and availability but introduces the problem of data consistency that arises when replicas are modified. Session guarantees may be used to manage replica consistency of a distributed system from a migrating client's perspective. This paper presents and proves safety of a new consistency protocol of session guarantees using object-based version vectors.

1 Introduction

Required properties of a distributed system with respect to consistency depend, in general, on the application and are formally specified by *consistency models*. There are numerous consistency models developed for *Distributed Shared Memory* systems. These models, called data-centric consistency models, assume that servers replicating data also access the data for processing purposes. In a mobile environment, however, clients are separated from servers; they can switch from one server to another. This switching adds a new dimension of complexity to the problem of consistency. *Session guarantees* [2], called also client-centric consistency models, have been proposed to define required properties of the system regarding consistency from the client's point of view. Intuitively: the client wants to continue processing after a switch to another server so that new operations will remain consistent with previously issued operations within a *session*.

Consistency protocols of session guarantees must efficiently represent sets of operations performed in the system. Version vectors based on vector clocks [3, 4] may be used for this purpose. Bayou system [5], implementing session guarantees for the first time, uses server-based version vectors. Safety of the consistency protocol of session guarantees, called VsSG, that uses server-based version vectors has been proved in [6]. Other approaches have been analyzed in [7].

In this paper a new consistency protocol of session guarantees, called VoSG, that uses object-based version vectors is proposed, and its safety is formally proved. Object-based version vectors, due to their structure, represent sets of writes defined by session guarantees more accurately comparing to server-based

* This work was supported in part by the State Committee for Scientific Research (KBN), Poland, under grant KBN 3 T11C 073 28.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 978–985, 2006.

version vectors. The messages exchanged by the VoSG protocol in case of writes are also smaller than messages of the VsSG protocol. However, depending on the characteristics of given application or system, the structure of object-based version vectors is usually less stable comparing to server-based version vectors. In a frequently changing environment some dynamic version vector maintenance mechanism can be used to alleviate the problem [8].

2 Session Guarantees

In this paper we consider a replicated distributed storage system. The system consists of a number of *servers* holding a full copy of a set of *shared objects*, and *clients* running applications that access the objects. A client may access a shared object after selecting a single server and sending a direct request to the server. Clients are mobile, so they can switch from one server to another during a session. Session guarantees are expected to take care of data consistency observed by a migrating client.

The set of shared objects replicated by the servers does not imply any particular data model or organization. The only assumption is that the objects are *isolated*, which means that their methods can access only internal states of the objects, and not states of other objects. In practice, it means that isolated objects do not reference each other. Operations performed on shared objects are divided into *reads* and *writes*. A read does not change the state of a shared object, while a write does. A write may cause an update of an object, it may create a new object, or delete an existing one. The requests for operations are handled by clients synchronously, and operations are performed by servers sequentially.

Operations on shared objects issued by a client C_i are ordered by a relation $\overset{C_i}{\rightarrow}$, called *client issue order*. A server S_j performs operations in an order represented by a relation $\overset{S_j}{\rightarrowtail}$. All operations performed locally by a server S_j are recorded in a totally ordered set $(\mathcal{O}_{S_j}, \overset{S_j}{\rightarrowtail})$, called *history*. Writes and reads on objects will be denoted by w and r, respectively, and operations for which the type is irrelevant will be denoted by o. An operation performed by a server S_j will be denoted by $o|_{S_j}$. An operation performed on an object x will be denoted by $o|_x$. The identifier of an object being accessed by an operation o will be denoted by $\mathtt{id}(o)$.

Definitions of session guarantees depend on the definition of *relevant writes* representing writes that have influenced results of a given read. In case of isolated objects relevant writes are defined in the following manner:

Definition 1. *The set of relevant writes of a read operation r performed on an isolated object at server S_j consists of all previous writes on that object performed at the server:*

$$RW(r|_{S_j}) = \left\{ w \in \mathcal{O}_{S_j} : \mathtt{id}(w) = \mathtt{id}(r) \wedge w \overset{S_j}{\rightarrowtail} r \right\}$$

Session guarantees have been introduced in [2]. The following formal definitions are based on those concepts.

Definition 2. *Read Your Writes (RYW) session guarantee is a system property meaning that* $\forall C_i \forall S_j \left[w \xrightarrow{C_i} r|_{S_j} \Rightarrow w \overset{S_j}{\rightarrowtail} r \right]$.

Definition 3. *Monotonic Writes (MW) session guarantee is a system property meaning that* $\forall C_i \forall S_j \left[w_1 \xrightarrow{C_i} w_2|_{S_j} \Rightarrow w_1 \overset{S_j}{\rightarrowtail} w_2 \right]$.

Definition 4. *Monotonic Reads (MR) session guarantee is a system property meaning that* $\forall C_i \forall S_j \left[r_1 \xrightarrow{C_i} r_2|_{S_j} \Rightarrow \forall w_k \in RW(r_1) : w_k \overset{S_j}{\rightarrowtail} r_2 \right]$.

Definition 5. *Writes Follow Reads (WFR) session guarantee is a system property meaning that* $\forall C_i \forall S_j \left[r \xrightarrow{C_i} w|_{S_j} \Rightarrow \forall w_k \in RW(r) : w_k \overset{S_j}{\rightarrowtail} w \right]$.

To provide the above defined properties required by migrating clients, an appropriate mechanism, called consistency protocol of session guarantees, must be implemented in the distributed system.

3 The VoSG Protocol of Session Guarantees

The proposed VoSG protocol of session guarantees intercepts communication between clients and servers: at the client side before sending a request, at the server side after receiving the request and before sending a reply, and at the client side after receiving the reply. These interceptions are used to exchange and maintain additional data structures necessary to preserve appropriate session guarantees. Clients express their requirements regarding consistency by assigning a set of session guarantees to their sessions. The set of session guarantees is then passed to the protocol along with every operation request. After receipt of a new request, a server checks whether its state is sufficiently up to date to satisfy client's requirements. If the server's state is outdated then the request is postponed and will be resumed after updating the server.

Servers periodically exchange information about writes performed in the past in order to synchronize states of replicas. This synchronization procedure eventually causes total propagation of all writes directly submitted by clients. It does not influence safety of the VoSG protocol but rather its liveness, and therefore it will not be discussed in this paper (example procedure is presented in [5]). As opposed to [2] we do not assume total ordering of non-commutative writes. The reason is that the problem of total ordering of non-commutative writes is in fact orthogonal to the problem considered in this paper.

Session guarantees define implicitly sets of writes that must be performed by a server before proceeding to the current operation. The sets of writes are monotonically increasing as the clients issue new operations. For this reason it

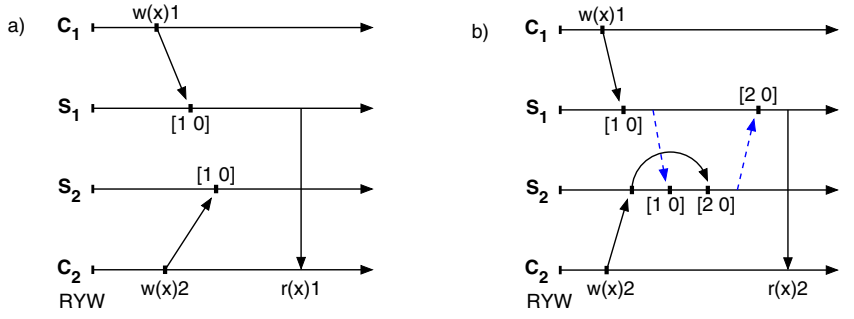


Fig. 1. Object-based version vectors and cache consistency

is not realistic to exchange explicit sets of writes between clients and servers for the purpose of checking appropriate session guarantees. Therefore, for efficient representation of sets of writes, we propose version vectors of the form $[\, v_1 \; v_2 \; \ldots \; v_{N_O} \,]$, where N_O is the total number of objects in the system, and a single position v_i denotes the number of writes performed on the i-th object by the selected server. Every server S_j maintains its local version vector V_{S_j}, and updates appropriate positions upon every write. Every write in the VoSG protocol is labeled with a vector timestamp set to the current value of the version vector V_{S_j} of the server S_j performing the write for the first time.

Fig. 1(a) shows a time-space diagram of a hypothetical execution in a system using object-based version vectors. Let us assume that there are two shared objects in the system, and initial values of server version vectors $V_{S_1} = V_{S_2} = [\, 0 \; 0 \,]$. The first client C_1 issues an operation $w(x)1$, which is a write of value 1 on object x. Let object x be represented by the first position in the version vector. After the write, the server's version vector V_{S_1} is set to $[\, 1 \; 0 \,]$. At the same time another client C_2 issues a write operation on the same object x at server S_2. After the second write the server version vector at S_2 is also set to $[\, 1 \; 0 \,]$, because S_2 has not contacted S_1 yet, and has not learned about the first write. Server version vectors V_{S_1} and V_{S_2} become equal, despite the fact that histories at the servers contain different write operations. As a result, the version vector representations of the sets of writes of histories at servers cannot reflect the differences. Therefore, the version vector management must be enhanced, so that every write will be uniquely timestamped. This can be achieved by global ordering of writes on respective objects, which effectively means that the servers must preserve *cache consistency* [9]. Example execution is presented in Fig. 1(b).

The VoSG protocol (presented in Fig. 2) maintains some data structures at clients and servers for managing session guarantees. Every client C_i maintains two version vectors: W_{C_i} and R_{C_i}. W_{C_i} represents all writes issued by the client, while R_{C_i} represents all writes that have influenced the results of reads issued by the client. Every server S_j maintains a version vector V_{S_j} representing all writes performed by the server. The version vector is updated whenever a new write is requested by a client, or when a synchronization between servers is performed.

Upon sending a request $\langle op, SG\rangle$ **to server** S_j **at client** C_i
1: $W \leftarrow \mathbf{0}$
2: **if** (`iswrite`(op) **and** MW $\in SG$) **or** (**not** `iswrite`(op) **and** RYW $\in SG$) **then**
3: $\quad W \leftarrow \max(W, W_{C_i})$
4: **end if**
5: **if** (`iswrite`(op) **and** WFR $\in SG$) **or** (**not** `iswrite`(op) **and** MR $\in SG$) **then**
6: $\quad W \leftarrow \max(W, R_{C_i})$
7: **end if**
8: send $\langle op, W\rangle$ to S_j

Upon receiving a request $\langle op, W\rangle$ **from client** C_i **at server** S_j
9: $seq \leftarrow 0$
10: **if** `iswrite`(op) **then**
11: $\quad seq \leftarrow$ `getSeqNumber`(`id`(op))
12: **end if**
13: **while** $\big(V_{S_j} \not\geq W \vee \big(seq > V_{S_j}[\texttt{id}(op)] + 1\big)\big)$ **do**
14: $\quad$ wait
15: **end while**
16: perform op and store results in res
17: **if** `iswrite`(op) **then**
18: $\quad V_{S_j}[\texttt{id}(op)] \leftarrow V_{S_j}[\texttt{id}(op)] + 1$
19: $\quad$ timestamp op with V_{S_j}
20: $\quad \mathcal{O}_{S_j} \leftarrow \mathcal{O}_{S_j} \cup \{op\}$
21: $\quad$ signal
22: **end if**
23: send $\big\langle op, res, V_{S_j}[\texttt{id}(op)]\big\rangle$ to C_i

Upon receiving a reply $\langle op, res, seq\rangle$ **from server** S_j **at client** C_i
24: **if** `iswrite`(op) **then**
25: $\quad W_{C_i}[\texttt{id}(op)] \leftarrow seq$
26: **else**
27: $\quad R_{C_i}[\texttt{id}(op)] \leftarrow seq$
28: **end if**

Every Δt **at server** S_j
29: **foreach** $S_k \neq S_j$ **do**
30: $\quad$ send $\big\langle S_j, \mathcal{O}_{S_j}\big\rangle$ to S_k
31: **end for**

Upon receiving an update $\langle S_k, \mathcal{O}\rangle$ **at server** S_j
32: **foreach** $w_i \in \mathcal{O}$ **do**
33: $\quad$ **if** $V_{S_j} \not\geq T(w_i)$ **then**
34: $\quad\quad$ perform w_i
35: $\quad\quad V_{S_j} \leftarrow \max\big(V_{S_j}, T(w_i)\big)$
36: $\quad\quad \mathcal{O}_{S_j} \leftarrow \mathcal{O}_{S_j} \cup \{w_i\}$
37: $\quad$ **end if**
38: **end for**
39: signal

Fig. 2. VoSG consistency protocol of session guarantees

The VoSG protocol interacts with requests sent from clients to servers and with replies sent from servers to clients. A request message is a couple $\langle op, SG\rangle$, where *op* is an operation to be performed, and SG is a set of session guarantees required by a given client. A reply is a triple $\langle op, res, W\rangle$, where *op* is the operation just performed, res represents the results of the operation (delivered to the application), and W is a vector representing the state of the server just after having performed the operation.

Before sending to the server, the request $\langle op, SG\rangle$ is substituted by a message $\langle op, W\rangle$, where W is a version vector representing writes required by the client. The vector W is calculated based on the type of operation (checked by the `iswrite`(*op*) function), and the set SG of session guarantees required by the client. The vector W is set to either $\mathbf{0}$, or W_{C_i}, or R_{C_i}, or to the maximum of these two vector (lines 1, 3 and 6). The maximum of two vectors V_1 and V_2 is a vector $V = \max(V_1, V_2)$, such that $V[i] = \max(V_1[i], V_2[i])$.

Upon receiving a new write request the server first generates a sequence number for the object being modified, which is accomplished by a call to function `getSeqNumber`() (line 11). It is assumed that sequence numbers are generated in a distributed manner by a separate algorithm. Before an operation is performed, two conditions are checked (line 13). The first condition ($V_{S_j} \geq W$) checks whether the server's version vector *dominates* the vector W sent by the client, which ensures that all writes required by the client have already been performed by the server. A vector V_1 dominates another vector V_2, which is denoted by $V_1 \geq V_2$, when $\forall i : V_1[i] \geq V_2[i]$. The second condition $seq \leq V_{S_j}[\mathtt{id}(op)] + 1$ ensures that all previous writes on a given object have already been performed, and the write represented by *op* is exactly the next write. The second condition is always true for reads, because *seq* is then set to 0, which allows reads to proceed. If the state of the server is not sufficiently up to date, the request is postponed (line 14), and will be resumed after synchronization with another server (line 39), or after another write (line 21).

After performing a write, the server version vector is incremented at position `id`(*op*), and a timestamped operation *op* is attached to the history $\mathcal{O}_{S_j}$. Next, a signal is generated (line 21) in order to resume threads that might be waiting because of inappropriate sequence number. Finally, the server sends back a reply with a single position of its version vector $V_{S_j}[\mathtt{id}(op)]$. The position represents the number of writes performed on the object just accessed.

At the client side, the number of writes sent by the server is stored at appropriate position of W_{C_i} in case of writes, or R_{C_i} in case of reads. The replies in the VoSG protocol contain only a single position of a version vector. The requests, however, must contain the whole version vectors.

4 Safety of the VoSG Protocol

Theorem 1. *The VoSG protocol preserves cache consistency.*

Proof. Before performing a new write, every server requests a sequence number for a given object. The sequence number must be exactly 1 greater than the value

stored in V_{S_j} version vector at appropriate position (see line 13 of the algorithm in Fig. 2). It means that all previous writes on that object must be performed before the current one. After performing a new write w, the V_{S_j} version vector is incremented at position $\mathtt{id}(w)$, and local history $\mathcal{O}_{S_j}$ is updated by attaching the write. Due to this procedure the histories contain continuous sequences of writes on respective objects. Writes that must be suspended, wait for updates from other servers. The missing writes are applied during synchronization (line 34). Histories exchanged by servers contain all operations on objects from the very beginning up to some time, therefore the missing parts can be applied without checking sequence numbers. □

Theorem 2. *RYW session guarantee is preserved by the VoSG protocol for clients requesting it.*

Proof. Let us consider two operations w and r, issued by a client C_i requiring RYW session guarantee. Let the read follow the write in the client's issue order, and let the read be performed by a server S_j, i.e. $w \overset{C_i}{\rightarrow} r|_{S_j}$. After performing w by some server S_k, $V_{S_k}[\mathtt{id}(w)]$ represents the last version number of the object identified by $\mathtt{id}(w)$. The client updates its vector W_{C_i}, so that $W_{C_i}[\mathtt{id}(w)] = V_{S_k}[\mathtt{id}(w)]$ (lines 23 and 25). The server S_j checks whether $V_{S_j} \geq W_{C_i}$ is fulfilled before performing r (lines 3 and 13), and hence ensures that $V_{S_j}[\mathtt{id}(w)] \geq W_{C_i}[\mathtt{id}(w)]$, which means that S_j has performed all previous writes on object $\mathtt{id}(w)$ up to and including the write w because writes on respective objects are totally ordered (Theorem 1). As local operations at servers are totally ordered, we get $w \overset{S_j}{\rightarrowtail} r$. This will happen for any client C_i requiring RYW and any server S_j, so $\forall C_i \forall S_j \left[w \overset{C_i}{\rightarrow} r|_{S_j} \Rightarrow w \overset{S_j}{\rightarrowtail} r \right]$, which means that RYW session guarantee is preserved. □

Theorem 3. *MR session guarantee is preserved by the VoSG protocol for clients requesting it.*

Proof. Let us consider two reads r_1 and r_2, issued by a client C_i requiring MR session guarantee. Let the second read follow the first read in the client's issue order, and let the second read be performed by a server S_j, i.e. $r_1 \overset{C_i}{\rightarrow} r_2|_{S_j}$. During the first read r_1, performed by some server S_k, the client updates its vector R_{C_i}, so that $R_{C_i}[\mathtt{id}(r_1)] = V_{S_k}[\mathtt{id}(r_1)]$ (lines 23 and 27). The server S_j checks whether $V_{S_j} \geq R_{C_i}$ is fulfilled before performing r_2 (lines 6 and 13), and hence ensures that $V_{S_j}[\mathtt{id}(r_1)] \geq R_{C_i}[\mathtt{id}(r_1)]$. This means that S_j has performed all previous writes on object $\mathtt{id}(r_1)$, because writes on respective objects are totally ordered (Theorem 1), and thus, according to Definition 1, S_j has performed all writes relevant to the read r_1. As local operations at servers are totally ordered, we get $\forall w_l \in RW(r_1) : w_l \overset{S_j}{\rightarrowtail} r_2$. This will happen for any client C_i and any server S_j, so $\forall C_i \forall S_j \left[r_1 \overset{C_i}{\rightarrow} r_2|_{S_j} \Rightarrow \forall w_l \in RW(r_1) : w_l \overset{S_j}{\rightarrowtail} r_2 \right]$, which means that MR session guarantee is preserved. □

A theorem and a proof for MW and WFR are analogous to RYW and MR respectively. Full versions of the theorems and proofs can be found in [10].

5 Conclusions

This paper has presented the VoSG consistency protocol of session guarantees, and a correctness proof showing that the protocol is safe, i.e. appropriate guarantees are provided when required. It is worth mentioning, however, that though the object-based version vectors used in the VoSG protocol are sufficient for fulfilling session guarantees, they are not necessary. Thus, other approaches are also possible, and they have been discussed in [7]. The sets of writes represented by version vectors are supersets of the exact sets resulting from appropriate definitions. The accuracy of the write-set representation is therefore an important factor of a consistency protocol of session guarantees influencing its performance. This problem is currently being considered, and appropriate simulation experiments are being carried out.

References

1. Tanenbaum, A.S., van Steen, M.: Distributed Systems — Principles and Paradigms. Prentice Hall, New Jersey (2002)
2. Terry, D.B., Demers, A.J., Petersen, K., Spreitzer, M., Theimer, M., Welch, B.W.: Session guarantees for weakly consistent replicated data. In: Proc. of the Third Int. Conf. on Parallel and Distributed Information Systems (PDIS 94), Austin, USA, IEEE Computer Society (1994) 140–149
3. Mattern, F.: Virtual time and global states of distributed systems. In Cosnard, Quinton, Raynal, Robert, eds.: Proc. of the Int'l. Conf. on Parallel and Distributed Algorithms, Elsevier Science Publishers B. V. (1988) 215–226
4. Fidge, C.: Logical time in distributed computing systems. Computer **24** (1991) 28–33
5. Petersen, K., Spreitzer, M.J., Terry, D.B., Theimer, M.M., Demers, A.J.: Flexible update propagation for weakly consistent replication. In: Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP-16), Saint Malo, France (1997) 288–301
6. Brzeziński, J., Sobaniec, C., Wawrzyniak, D.: Safety of a server-based version vector protocol implementing session guarantees. In: Proc. of Int. Conf. on Computational Science (ICCS2005), LNCS 3516, Atlanta, USA (2005) 423–430
7. Kobusińska, A., Libuda, M., Sobaniec, C., Wawrzyniak, D.: Version vector protocols implementing session guarantees. In: Proc. of Int. Symp. on Cluster Computing and the Grid (CCGrid 2005), Cardiff, UK (2005)
8. Ratner, D., Reiher, P., Popek, G.: Dynamic version vector maintenance. Technical Report CSD-970022, Univ. of California, Los Angeles (1997)
9. Goodman, J.R.: Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group (1989)
10. Brzeziński, J., Sobaniec, C.: Safety of VoSG consistency protocol of session guarantees. Technical Report RA-009/05, Institute of Computing Science, Poznań University of Technology (2005)

Minimizing Cost and Minimizing Schedule Length in Synthesis of Fault Tolerant Multiprocessors Systems

Mieczyslaw Drabowski and Krzysztof Czajkowski

Cracow University of Technology,
Warszawska 24, 31-155 Krakow, Poland
drabowski@pk.edu.pl, kczajkowski@wp.pl

Abstract. The paper includes a proposal of a new approach of synthesis of fault tolerant multiprocessors systems. Optimal task scheduling and optimal partition at resources are basic problems in high-level synthesis of computer systems. Coherent synthesis may have a practical application in developing tools for computers aided design of such systems.

1 Introduction

The goal of high-level synthesis of computer systems is to find an optimum solution satisfying the requirements and constraints enforced by the given specification of the system. The following criteria of optimality are usually considered: costs of system implementation, its operating speed and fault tolerant. A specification describing a computer system may be provided as a set of interactive tasks. In any computer system certain tasks are implemented by hardware.

The basic problem of system synthesis is partitioning system functions due to their hardware and software implementation. The goal of the resources assignment is to specify what hardware and software resources are needed for the implementation and to assign them to specific tasks of the system, even before designing execution details. In the synthesis methods used so far, software and hardware parts are developed separately [3], [10] and then composed, what results in cost increasing and decreasing quality of the final product.

Task scheduling is one of the most important issues occurring in the synthesis of operating systems responsible for controlling allocation of tasks and resources in computer systems. Another important issue that occurs in designing computer systems is assuring their fault-free operation. Such synthesis concentrates on developing fault-tolerant architectures and constructing dedicated operating systems for them. In this system an appropriate strategy of self-testing during regular exploitation must be provided. In general, fault tolerant architectures of computer systems are multiprocessor ones. The objective of operating systems in multiprocessor systems is scheduling tasks and their allocation to system resources. For fault tolerant operating system, this means scheduling usable and testing tasks - multiprocessors tasks - that should detect errors of executive modules, in particular processors [2].

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 986–993, 2006.

Modeling fault tolerant systems consists of resource identification and task scheduling problems that are both NP-complete. Algorithms for solving such problems are usually based on heuristic approaches [4]. The objective of this paper is to present the concept of combined (synergic) approach to the problem of fault tolerant system synthesis, i.e. a coherent solution to task scheduling and resource assignment problems. The solution includes also the system testing strategies based on multiprocessors tasks.

2 Coherent Process of Fault Tolerant System Synthesis

Modeling the joint search for the optimum task schedule and resource partition of the designed system into hardware and software parts is fully justified [1]. Simultaneous consideration of these problems may be useful in implementing optimum solutions, e.g. the cheapest hardware structures. With such approach, the optimum task distribution is possible on the universal and specialized hardware and defining resources with maximum efficiency. We propose the following schematic diagram of a coherent process of fault tolerant systems synthesis (Figure 1).

The suggested coherent analysis consists of the following steps [5], [6]:

1. Specification for the system. 2. Specification of tasks. 3. Assuming the initial values of resource set. 4. Defining testing tasks and the structure of system - testing strategy selection, 5. Task scheduling, 6. Evaluating the speed and system cost 7. The evaluation should be followed by a modification of the resource set, a new system partitioning into hardware and software parts and step 4.

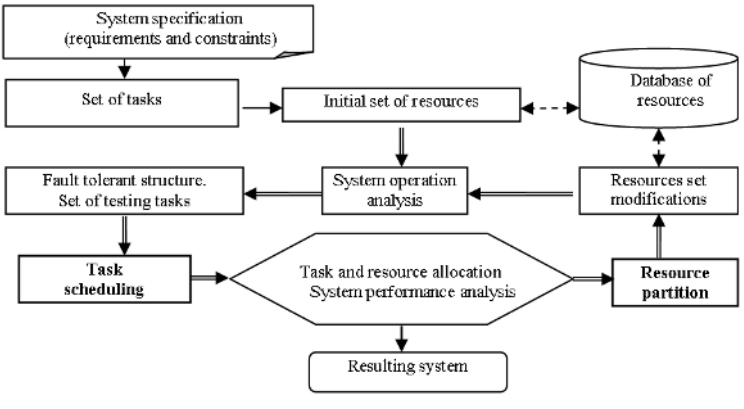


Fig. 1. The process coherent synthesis of dependable computer system

In this approach a combined search for optimal resources partition and optimal tasks scheduling occur. Iterative calculations are executed till satisfactory design results are obtained - i.e. optimal system structure, fault tolerant level and schedule.

3 Example of Fault Tolerant System Synthesis

Let us discuses a simple example of parallel multiprocessors system synthesis assuming that functional description of the system is given by the tasks digraph. A system will be optimized as regards cost (cost minimization) and execution speed (schedule length minimization). This system will be implemented in the common structure (without fault tolerant) and in the fault-tolerant structure [8].

3.1 System Specification

Testing system should realize tasks, which can be presented by a digraph (Fig. 2).

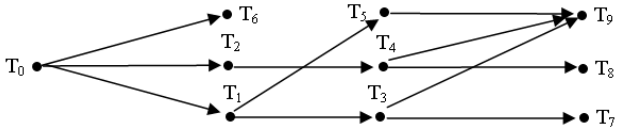


Fig. 2. Tasks digraph

We will assume executing times of tasks the following: $t_0 = 3, t_1 = 2, t_2 = 2, t_3 = 1, t_4 = 3, t_5 = 3, t_6 = 4, t_7 = 3, t_8 = 1, t_9 = 1$. As an optimal criterion for projecting system we will assume minimization of executing time for all testing tasks - minimization of testing tasks schedule length. The second considered optimal criterion is system cost which takes into consideration cost of all hardware resources. Cost of system is:

$$C_S = C_P + i * C_M \tag{1}$$

where: C_P - processor cost, C_M - memory cost, i - number of memory modules.

The additional requirements of designing system, we will assume as the following: RI - There is necessary a deadline of all tasks without delays executing which equals (or less) 13 time units. RII - There is necessary execute task T6 nonpreemptable (in time). RIII - There is necessary a critical line for T6 task which equals 9 time units. RIV - There is desirable a deadline, executing all processes without delay, equal (or less) 10 time units.

3.2 Project of Structure and Schedule Without Fault-Tolerant Technique

If we consider two identical and parallel processors (Figure 3), then this optimal schedule fulfills requirement RI that will realize Muntza-Coffmana algorithm. Cost of this system - with assumption that every processor needs one memory unit, equals $C_S = 2*C_P+10*C_M$. Taking into consideration requirement RII it is necessary to correct tasks schedule. System cost doesn't change. For requirement RIII realization there is necessary further correction of tasks schedule. System cost doesn't change, too. At last it turns out; that all requirements (include RIV)

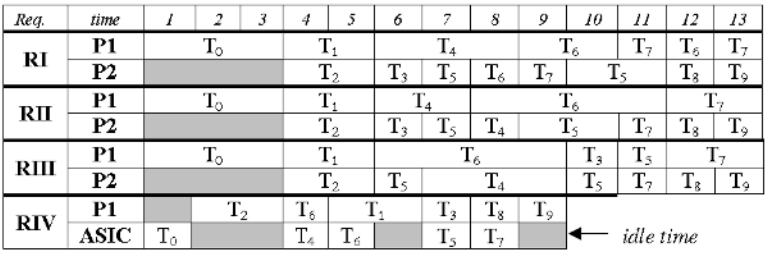


Fig. 3. Realized requirements RI, RII, RIII, RIV

can't be realized on two processors. We will apply specialized resource, which can execute tasks: T_0, T_4, T_5, T_6, T_7 with a triple speed (as compared to the standard processor) and its cost is C_{ASIC}. Resources structure and processor schedule we are showing in (Figure 3). System cost equals $C_S = C_P + 6 * C_M + C_{ASIC}$.

3.3 Project of Structure and Schedule in Fault-Tolerant System

The cost of the system should be as low as possible, and the architecture conformant with the fault tolerant system model is required, with two-processor testing tasks.

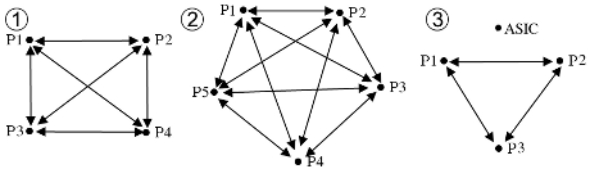


Fig. 4. Structure 1: processors and two-processor testing tasks in a fault tolerant four-processor structure: $T_{12}, T_{13}, T_{14}, T_{23}, T_{24}, T_{21}, T_{34}, T_{31}, T_{32}, T_{41}, T_{42}, T_{43}$. Structure 2: processors and two-processor testing tasks in a fault tolerant five-processor structure - $T_{12}, T_{13}, T_{14}, T_{15}, T_{21}, T_{23}, T_{24}, T_{25}, T_{31}, T_{32}, T_{34}, T_{35}, T_{41}, T_{42}, T_{43}, T_{45}, T_{51}, T_{52}, T_{53}, T_{54}$. Structure 3: processors and two-processor tasks in a fault tolerant three-processor system with a specialized ASIC processor - $T_{12}, T_{13}, T_{23}, T_{21}, T_{31}, T_{32}$.

We shall assume the following labeling for processor testing tasks - T_{gh}, where P_g processor is testing (checking) P_h processor. Implementing the system satisfying the requirement RI, the architecture of a fault tolerant system was shows in (Figure 4).

For such architecture, the optimum tasks schedule, guaranteeing the requirement RI, has been shown in (Figure 5). Taking into account the requirement RII, the following correction is done to the task schedule. Thus, we obtain the schedule shown in requirement RII. The system architecture and costs remain unchanged. The next requirement RIII is reflected in a corrected schedule presented in requirement RIII.

Req.	time	1	2	3	4	5	6	7	8	9	10	11	12	13
RI	P1	T_{12}	T_0	T_0	T_{13}	T_1	T_{31}	T_{14}	T_{21}	T_7	T_{41}	T_7	T_6	T_{12}
	P2	T_{12}	T_{23}		T_1	T_{24}	T_4	T_4	T_{21}	T_{32}	T_6	T_{42}	T_8	T_{12}
	P3	T_0	T_{23}	T_{34}	T_{13}	T_2	T_{31}	T_5	T_4	T_{32}	T_5	T_5	T_{43}	T_7
	P4			T_{34}	T_2	T_{24}	T_3	T_{14}	T_6	T_6	T_{41}	T_{42}	T_{43}	T_9
RII	P1	T_{12}	T_0	T_0	T_{13}	T_1	T_{31}	T_{14}	T_{21}	T_7	T_{41}	T_6	T_7	T_{13}
	P2	T_{12}	T_{23}		T_1	T_{24}	T_4	T_4	T_{21}	T_{32}	T_6	T_{42}	T_8	T_{12}
	P3	T_0	T_{23}	T_{34}	T_{13}	T_2	T_{31}	T_5	T_4	T_{32}	T_5	T_5	T_{43}	T_7
	P4			T_{34}	T_2	T_{24}	T_3	T_{14}	T_6	T_6	T_{41}	T_{42}	T_{43}	T_9
RIII	P1	T_{12}	T_0	T_0	T_{13}	T_1	T_{31}	T_{14}	T_{21}	T_7	T_{41}	T_1	T_7	T_{12}
	P2	T_{12}	T_{23}		T_1	T_{24}	T_6	T_6	T_{21}	T_{32}	T_4	T_{42}	T_8	T_{12}
	P3	T_0	T_{23}	T_{34}	T_{13}	T_2	T_{31}	T_5	T_4	T_{32}	T_5	T_5	T_{43}	T_7
	P4			T_{34}	T_2	T_{24}	T_3	T_{14}	T_6	T_6	T_{41}	T_{42}	T_{43}	T_9

Fig. 5. Tasks schedule satisfying the requirement RI in a four-processor system (Req. RI), the requirements RI and RII in a four-processor system (Req. RII), the requirements RI, RII, and RIII in a four-processor system (Req. RIII) - Structure 1

St.	time	1	2	3	4	5	6	7	8	9	10	11	12	13
2	P1	T_{12}				T_{13}	T_3	T_4	T_{14}	T_9	T_{15}	T_{21}		
	P2	T_{12}	T_{23}		T_2	T_2	T_{24}	T_5	T_5	T_{25}	T_8	T_{21}	T_{32}	
	P3		T_{23}	T_{34}	T_6	T_{13}	T_5	T_{35}	T_4	T_4	T_9		T_{32}	T_{43}
	P4	T_0	T_0	T_{34}	T_{45}	T_1	T_{24}	T_6	T_{14}	T_7	T_7			T_{43}
	P5			T_0	T_{45}	T_6	T_6	T_{35}	T_7	T_{25}	T_{15}			
3	P1	T_{12}	T_2	T_{13}	T_{21}	T_1	T_{31}	T_{12}	T_8	T_{13}				
	P2	T_{12}	T_{23}	T_2	T_{21}	T_{32}	T_1	T_{12}	T_{23}	T_{13}				
	P3		T_{23}	T_{13}	T_6	T_{32}	T_{31}	T_3	T_{23}	T_9				
	ASIC	T_0			T_4	T_6		T_5	T_7					

Fig. 6. Tasks schedule satisfying the requirements RI, RII, RIII and RIV: in the five-processor system - Structure 2 and in the three-processor system with the specialized processor - Structure 3

Considering the RIV requirement, the system structure change is necessary. Two variants of the structure shall be proposed. The first structure consists of five identical parallel processors, with two-processor testing tasks (Figure 4) (Structure 2). Task schedule in such structure is depicted in (Figure 6). In the second variant, a specialized module (ASIC) was applied, that may perform the tasks: T_0, T_4, T_5, T_6 and T_7 with a triple speed (as compared with the standard universal processor). The system structure and schedule are shown in (Figure 4) (Structure 3) and (Figure 6), respectively. The universal processor completes processing of usable tasks in 9 time units, while ASIC processor completes performing its function in 8 time units. Accordingly, the required deadline was reached in 9 units.

The cost of the developed system shall be estimated as follows. If we assume that each usable task performed by a universal processor needs one memory unit dedicated to such task, and task assigned to ASIC processor do not need dedicated memory units the system cost is: $C_S = m * C_P + n_u * C_M + p * C_{ASIC}$ (2) where: m - the number of identical parallel processors, n_u - the number of tasks assigned to universal processors, p - the number of specialized ASIC

processors devoted for processing remaining $(n - n_u)$ tasks. For the requirements: RI, RI+RII, RI+RII+RIII : $m = 4, n_u = 10, p = 0$. For the requirements RI+RII+RIII+RIV, in the first variant, $m = 5, n_u = 10, p = 0$ and in the second variant, where one ASIC processor is applied, $m = 3, n_u = 6$ and $p = 1$.

4 Experimental Verification of the Coherent Synthesis

As mentioned earlier, the problems presented in this work (resource partitioning, task scheduling) belong to NP-complete class. We shall present the final results of the computer experiments obtained with coherent and non-coherent approach. A meta-heuristic algorithms were applied: neural network and Tabu search. The algorithms were presented in [5], [7], [8]. The optimality criteria used in our experiments were minimum cost of the system and minimum processing time. The computational results are presented in Table 1.

Table 1. Example coherent and non-coherent synthesis

Number of tasks	Minimum processing time				Minimum system cost			
	non-coherent		coherent		non-coherent		coherent	
	cost	time	cost	time	cost	time	cost	time
5	1.0	5.67	1.0	5.67	1.0	5.67	1.5	5.1
10	1.25	7.75	1.25	7.73	1.8	7.4	1.8	7.42
15	1.5	8.4	1.5	8.1	3.5	7.7	3.1	7.05
20	1.5	11.4	1.5	11.2	3.6	8.65	3.7	7.7
25	1.5	14.2	1.5	14	4.2	7.95	3.9	7.7
30	1.5	17.6	1.5	16.8	4.1	9.2	4.3	7.95
35	2.5	15.75	2.5	13.7	5.6	8.45	5.5	7.77
40	2.5	18.25	2.5	17.1	7.1	8.65	6.35	7.7
45	2.5	19.5	2.5	19	8.6	9.6	7.4	5.1
50	2.75	19.4	2.75	16.3	8.4	8.65	8.2	7.45
55	2.75	18	2.75	17.8	9.51	9.54	8.9	7.95

Analyzing the presented results one may conclude that the coherent algorithm obtains better solutions in terms of the implementation costs criterion, as well as in terms of the operating speed criterion for the designed implementations. These results are collected in the charts as depicted in Figure 7 charts 1 and 2.

As for the time minimization (Figure 7 chart 1), both algorithms produce similar values of the cost for all sample task sets. Coherent algorithm improve the task processing time substantially, in particular for graphs with more then 30 tasks. For instance, 15% improvement of total processing time was obtained for graphs with 50 tasks.

The diagram presenting the relationship between the costs and the number of tasks (Figure 7 chart 3) suggests that the solutions obtained by coherent algorithms are considerably cheaper than the ones obtained by a non-coherent algorithm. A coherent algorithm obtains similar task time performance times in

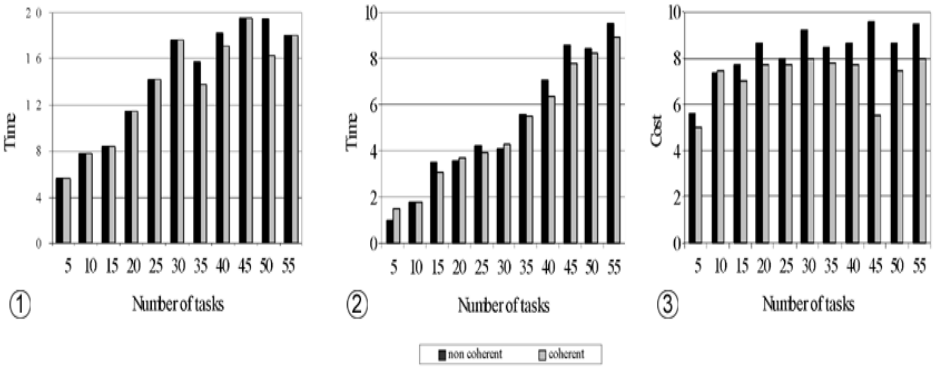


Fig. 7. Chart 1: Minimum processing time. Chart 2: Minimum system cost. Chart 3: Minimum system cost.

cheaper structures that the ones defined by the non-coherent algorithm. This is particularly visible for the case of graphs with 45 tasks.

Regarding the cost minimization, the diagram of the relationship between the time and number of tasks in the system (Figure 7 chart 2) shows that the obtained processing time is similar for both approaches.

5 Conclusions

The direction and the synthesis method presented in this paper are an attempt of comprehensive and coherent approach to high-level system synthesis. This synthesis is directed to systems with tasks self-testing the system main resources, namely the processors. Such approach in designing fault tolerant systems accounts for indivisible testing tasks [1], [11]. The following system optimization criteria are accepted: operating time minimization and cost minimization. Synergic solution is a result of cooperation between the scheduling algorithms and the algorithms responsible for resource partition. One may also specify additional optimality criteria, i.e. minimum power consumption of the designed system (which is particularly significant for embedded and mobile systems). For the proposed system's relevance to real systems, one should take into account the processes of communication between resources and tasks, preventing resource conflicts, as well as extend the available resources sets, for example by programmable and configurable structures.

This work presents only one of the methods of providing self-testability of the designed system. Fault tolerant is particularly significant for real time systems [2], [4], that is why the synthesis should include the criterion of the task scheduling optimality before deadlines. The problem of coherent synthesis is a multi-criteria optimization problem. Taking into account several criteria and the fact that optimization of one criterion results often in worsening of the second one, indicated at selecting optimization in the Pareto sense. The optimum

solution in such case shall be the whole expanse of solutions. The above issues are now studied.

References

1. Berrojo L.,Corno F.,Entrena L.,Gonzales I.,Lopez C.,Sonza Reorda M.,Squillero G.: An industrial environment for high-level fault tolerant structures insertion and validation. In: Proc. 20th IEEE VLSI Test Symp. Monterey (2002) 229-236
2. Blazewicz J., Drabowski M., Weglarz J.: Scheduling multiprocessor tasks to minimize schedule length. In: IEEE Trans. Computers C-35, No.5. (1986) 389-393
3. Chen L.,Dey S.,Sanchez P.,Sekar K.,Chen Y.: Embedded hardware and software self-testing methodologies for processor cores. In: Proc. 37th Design Automation Conf. Los Angeles (2000) 625-630
4. Drabowski M.: Co-synthesis of real time system. In: Processing of IX Conference Real time systems. The Silesian University of Technology (2002) 279-289
5. Drabowski M.: Coherent synthesis of real time systems the neural approach. In: Modern problems of Real Time Systems. WNT, Warszawa (2004) 13-24
6. Drabowski M., Czajkowski K.: Coherent Synthesis of Heterogeneous System - a Tabu Search Approach. In: Proceedings of Artificial Intelligence Studies Vol. 2(25), Proceedings on VII Int. Conference AI'20/2005. Siedlce (2005) 139-147
7. Drabowski M., Czajkowski K.: Task scheduling in coherent co-synthesis of computer system. In: Image Analysis, Computer Graphics, Security Systems and Artificial Intelligence Applications, Vol. 15. Bialystok (2005) 53-62
8. Drabowski M., Wantuch E.: Coherent concurrent tasks scheduling and resources assignment in dependable system design. In: Proc. of European Safety & Reliability Conference. London (2005) 487-495
9. Drabowski M., Wantuch E.: Concurrent synthesis of heterogeneous information systems - deterministic approach. In: Proceedings of Int. Congress InBus. Krakow (2004) 53-58
10. Hyunok Oh., Soonhoi Ha.: Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In: Proceedings of tenth Int. Conf. on Hardware / Software codesign. IEEE Computer Society Press (2002) 133-138
11. Xie Y.,Li L.,Kandemir M.,Vijaykrishnan N.,Irwin M. J.: Reliability-aware Cosynthesis for Embedded Systems. In: Proc. of ASAP'04. (2004) 41-50

A Model of Exception Propagation in Distributed Applications

Paweł L. Kaczmarek and Henryk Krawczyk

Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Poland
{pkacz, hkrawk}@eti.pg.gda.pl

Abstract. The paper presents a method of modeling exception propagation in distributed applications. The method is based on sequential control flow analysis and distributed control dependence analysis, that are used to detect potential interactions between subsystems. The consequences of exception occurrence are identified in local and remote contexts by calculating both explicit propagation paths and implicit influences between parallel subsystems. In order to present the adequacy of exception handling constructs, a representative, multithreaded application is prepared and quantitative information about its behavior is gathered.

1 Introduction

Exception handling (EH) is commonly used to increase software dependability in sequential and distributed systems. An exception is a special event, usually the result of an error, that causes a change in the control flow. After throwing an exception, a handling function is called. If an exception can not be handled, it is propagated to the caller that either handles it or further propagates it.

In the paper, we use control flow analysis (CFA) [1] to identify potential exception propagation paths. CFA expresses paths of program execution, it uses a CF graph ($CFG(V, E)$), in which vertexes (V) represent program statements (denoted as $v, v' \in V$) and edges (E) represent potential transitions. If an edge exists from v to v', the execution of v' is immediately preceded by v. Control dependence analysis(CDA) [2] is derived from CFA. Informally, v' depends on v if the execution of v' depends on the result of v.

There are many approaches to analyze exception handling mechanisms used in applications. The problem of failure propagation in distributed systems is addressed in [3], in which authors present the concept of $f - maps$, that are constructed from potential points of exception throws and the abilities of propagation of methods. Initial and continued propagation of exceptions is described in [4]. The author performs static code analysis to present source code metrics. An attempt to simplify the analysis of CFG in the presence of exceptions has been made in [5], in which the concept of Factored CFG is presented. $FCFG$ is constructed by grouping many blocks in one. Although industrial tools for CFA exist (e.g. [6]), they neglect parallelism and, in most cases, also exceptions.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 994–1001, 2006.

There is no work known to us, that uses CFA to analyze exception propagation in distributed applications. In the work, we adopt CFA to such analysis. Additionally, we present algorithms to calculate implicit influence of exceptions on parallely executing subsystems (i.e. parts of application).

2 Control Flow Analysis of Distributed Applications in the Presence of Exceptions

We use a new concept of distributed CFG with exceptions ($DCFGE(V, E)$) that is constructed by extending the basic CFG with three features: exception handling representation [7], distributed calls and dependencies [8] [9], and state information about application. $DCFGE$ contains the following structure:

- standard CFG - vertexes for program statements and edges for intra- and inter-procedural control flows,
- vertexes that represent exceptional exits of methods,
- additional edges that represent inter-method transitions in case of exceptional exits, the edges direct from exceptional exits to handling functions or to exceptional exit vertexes of another method in the call chain [7],
- parallel execution edge that represents distributed control flow [8],
- edges that represent distributed dependencies: synchronization and communication [9].

In CDG of a distributed application, the dependencies of synchronization and communication are defined. Informally, statements v, v' are communication dependent if the value of a variable assigned at v' is used at u by an interprocess communication. Statements v, v' are synchronization dependent if the start and/or termination of v' directly determines the start/termination of v. A detail and formal description of the basic concepts is presented in [7] [8] [9].

The proposed approach allows to analyze applications on different levels of abstraction. In the paper, we consider the instruction level, however, CFG can be combined on higher levels to reduce computational complexity. In the compact analysis, methods or subsystems are represented by their regular exits, exceptional exits and distributed dependencies. The compact form is less precise than a detail $DCFGE$, however allows to simplify the analysis. The nodes and edges of an example $DCFGE$ are presented in Fig. 1. The figure does not contain the state information described in Sect. 2.2, distributed dependencies have been presented summarily for methods.

2.1 DCFGE of an Example Application

The part of the source code (Java) of an example program that has been analyzed is presented in Listing 1.1. The full source code and the libraries for execution tracing (logging and log analysis) are available at author's web site:

www.eti.pg.gda.pl/∼pkacz/analysis.html.

```
1
2  public class Server extends
3  Thread {public void run(){ ... }
4  ...
5  synchronized void acceptBuyOffer
6  (...)throws AlreadySoldExc,
7   InvalidServerStateExc ,...
8  {if (state!=Consts.BUYOFFERS)
9     throw new
10   InvalidServerStateExc();
11   ...
12   if ((!offerFound)||
13     (!offerAvailable))
14     throw new AlreadySoldExc();
15   ... }
16 synchronized OfferData getOffer
17 (...)throws InvalidServerStateExc
18 {...
19  if (state!=Consts.BUYOFFERS)
20  throw new
21   InvalidServerStateExc();
22  ...}
23 private synchronized void
24 endOfSell () throws ...
25 {for (...)
26  {try {
27   switch
28    (soldOffers[i].getStatus()){
29   case Consts.SECONDNOTIF:
30    throw new ChangeBuyerExc ();
31   ... } }
32   catch (...) {}
33   catch(ChangeBuyerExc e)
34   {if (...) { ...
35     sendNotification
36        (soldOffers[i])}
37     else {...} }
38  ... } ... }
39 private void sendNotification
40  (Offer offer) throws ...
41 {if(offer.getBuyer()!=null){
42   offer.getBuyer().acceptBuy
43     (offer.getOfferData());
44   ... }
45  else {...}
46  } }
47
48 public class Buyer extends
49   Thread { ...
50 public void run (){
51  for (...)
52  { try { ... ;
53     giveBuyOffer(); }
54  catch (AlreadySoldExc e)
55    {...}
56  catch(InvalidServerStateExc
57     e) {...}
58   catch (...) {...}
59 } }
60 void giveBuyOffer ()
61   throws AlreadySoldExc,
62 InvalidServerStateExc, ...
63 {...
64 offerData=server.getOffer(...);
65  if (offerData!=null){
66    ...
67    server.acceptBuyOffer
68      (offerData,this)};
69  ... }
70 void acceptBuy (OfferData o)
71   { ... }}
```

Listing 1.1. Example code

The program is a simulation of an auction system. The server object supports synchronization between sell and buy offers, and maintains the list of items. At given time intervals, the server ends the auction and sends information to the buyers. If the buyer does not confirm the buy, a second notification is send or the buyer is changed.

2.2 Information About Application State in *DCFGE*

Vertexes in *DCFGE* are described with additional information about the state corresponding to anomaly handling, which allows for an automated and precise description of propagation with the use of *DCFGE*. Only explicitly thrown exceptions are expressed with vertexes in the graph, while other constructs require other analysis or human interpretation. Anomalies are divided into three groups: (i) explicit *EH* constructs, (ii) anticipated (known) anomalies (e.g. pre-condition violation in a request, returned error codes), (iii) unanticipated anomalies. The first group is detected straightforwardly from application code. The second group (i.e. known fault-tolerance mechanisms) can be detected automatically, however initial human tuning is necessary, e.g. defining code patterns for condition checks. The third group concerns typical errors in application code, which is not

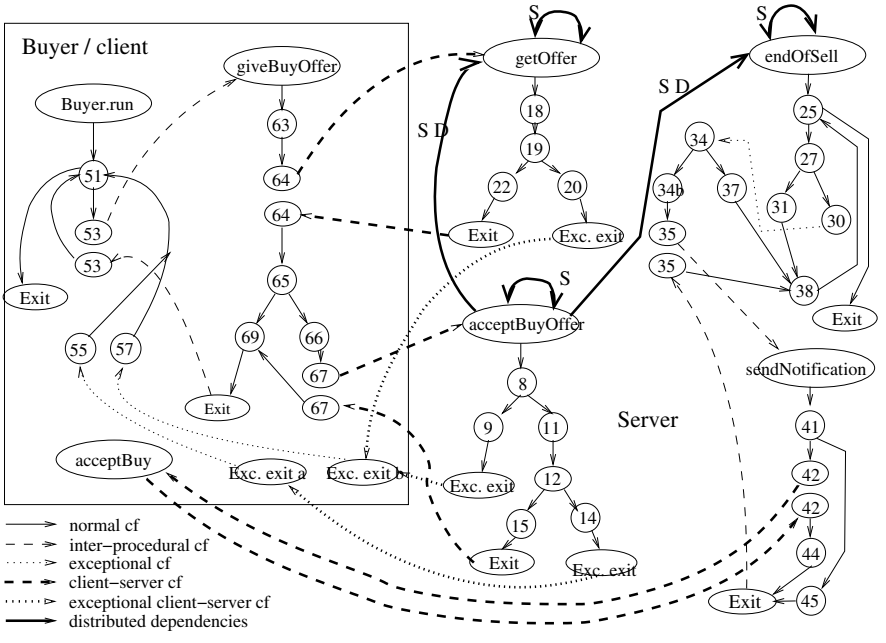


Fig. 1. *DCFGE* of the application from Listing 1 - distributed dependencies are presented summarily for methods, *cf* denotes control flow, *S*, *D* denote synchronization and communication dependencies respectively

in the main scope of the paper, although it can be analyzed with explicit human interpretation.

Information about application state is added to nodes of *DCFGE* either automatically (*auto*) or by a human (*human*). Human-driven modifications extend the range of the analysis, however they are not necessary. Selected nodes are marked with a flag that describes the current state of an application:

- exception throw (*ET*) - throw statements (*auto*),
- handling function (*HF*) - instructions within catch blocks (*auto*),
- error notification (*EN*) - if *EN* is passed through a node, (*human* or *auto* with pattern comparison - if possible)
- incorrect handling (*IH*) - if an error occurs in handling, (*human*),
- incorrect (*IC*) - if a fault is detected in the code, this situation is not likely to occur unless fault injection is used (*human*).

Additional extensions are made due to distributed aspects:

- remote call (*RC*) - represents a remote call (*auto*),
- asynchronous call (invocation) (*AC*) - represents a split of control flow (*auto*),
- remote exception throw (ET_{Remote}) - assigned to a node if an exception is propagated from a remote call (*auto*)

- remote error notification (EN_{Remote}) - assigned to a node if an error notification is passed from a remote call (*auto* or *human*)
- asynchronous exception throw ($ET_{Asynchronous}$) - assigned to a node if an exception is propagated asynchronously to remote components (*auto*),
- cooperative handling (CH) - communication within HF (*auto*) [10].

3 The Analysis of *DCFGE*

DCFGE is used to derive two kinds of information: precise information about explicit exception propagation (sequential and distributed) and estimation of the implicit influence of an exception on distributed subsystems through distributed dependencies. The analysis is done in a "scope", i.e. the guarded block that supplies an adequate handling function. If the block is not available, the "scope" ranges from the beginning to the end of the method propagating an exception (e.g. a server method).

3.1 Deriving Information About Explicit Propagation

Automated analysis of explicit exception propagation takes on input the *DCFGE* with state information and calculates propagation paths. The calculation is done for statements that may initialize exceptional events (referred to as starting points - SP): explicit throw statements, throws declarations of methods, remote invocations.

```
1. identify all starting points (SP) of exceptions;
2. foreach (SP) {
3. use the depth first (DF) algorithm to find all control flow paths
   in DCFGE from the SP to the end of each scope
4. foreach (path)  {
5.   record the nodes together with corresponding flags;
6.   if IH or IC nodes exist, mark the CF as incorrect;
7.   if RC or AC nodes exist, mark the CF as such;
8.   if ET(Remote), EN(Remote) or ET(Asynchronous) nodes exist,
         mark the CF as exception multiplication;
9.   if CH nodes exist, mark the CF as cooperative handling;}}
```

The information about propagation contains SP, nodes and corresponding application states. Although the number of all CFs in a CFG has exponential complexity, we assume that propagation graphs are sparse within their scopes. As an example, consider the exception thrown by the server in getOffer (line 20) (ET), the exception is propagated to the client (ET_{Remote}) in which it is handled. The CF is flaged as exception multiplication.

3.2 Deriving Information About Implicit Impact on Remote Subsystems

Apart from explicit changes of the control flow and communication, an exception can implicitly influence remote subsystems. The impact is estimated, because we

intend to present a fast and simple method for identifying potential influences. Two levels of influence of distributed dependencies are distinguished:

- level 1 - operations invoked before an exception throw, the $DCFGE$ is analyzed from the SP to the beginning of each scope,
- level 2 - potential execution paths if the exception has not occurred, the paths from the first instruction excluding the SP to the end of the scope.

The algorithm of approximating the impact of an exception on remote subsystems works as follows:

```
1. foreach (starting point (SP))
2. { identify the list of scopes for the SP;
3.   foreach (scope)
4.   { agregate level 1 dependencies and their types:
       walk backward from the SP to the beginning of the scope
         create the list of met dependencies
5.     agregate level 2 dependencies and their types:
       assuming the branch with the SP has not been chosen, gather
         nodes on paths from the branch node to the end of the scope
         create the list of met dependencies  }  }
```

After the analysis, information about each propagation path is extended with the list of potentially affected remote subsystems. We assume that method calls are analyzed in a compact way. As an example consider again the exception thrown at line 20 (ET). The scope of this SP is the try/catch block in Buyer.run. Let us assume, that the application performs a modification of a shared variable at line 18. Consequently, a level 1 communication dependency is present. The situation is a potential error in application code and should be inspected.

3.3 Quantitative Values Describing Control Flow Paths

$DCFGE$ presents information about potential execution paths, which is constructed by static analysis of program code. Additionally, dynamic analysis is used to describe the behavior of a running application [11]. The number of executions of control flows, method exit results and anomaly events are counted. The information describes a relative importance of each flow and rates the relevance of the constructs. If the number of exceptional control flows is high, the application or a subsystem need to be improved. Quantitative characteristics are gathered from a real execution of an application in a working environment.

The analysis is done in the following steps: (i) apply logging functions to register all anomaly situations, (ii) execute the modified application, (iii) analyze the execution log. Currently, we use a dedicated logging library together with a tool for automated gathering of general execution characteristics.

4 Experimental Results

The technique is demonstrated on a snippet of code from Listing 1.1 and $DCFGE$ from Fig. 1. The method described in previous sections is used to

Table 1. Propagation paths

Path name	Path description - vertexes(states)	The number of executions
$PP9$	9(ET), acceptBuyOffer.ExcExit (ET), giveBuyOffer.ExcExit_b (ET), 57 (HF/EN)	4
$PP14$	14 (ET), acceptBuyOffer.ExcExit (ET), giveBuyOffer.ExcExit_a (ET), 55(HF/EN)	9
$PP20$	20(ET), getOffer.ExcExit (ET), giveBuyOffer.ExcExit_b (ET), 57 (HF/EN)	114
$PP30_a$	30(ET), 34(HF), 34b(HF), 35(HF), 41(HF), 42 (HF,CH), acceptBuy (CH)	2
$PP30_b$	30(ET), 34(HF), 37(HF)	8
$PP30_c$	30(ET), 34(HF), 34b(HF), 35(HF), 45(HF)	0

gather: static description of propagation paths and quantitative information about execution. Existing tools that generate CFG of Java programs do not concern distributed dependencies and generally neglect exceptional constructs. Therefore, the exemplary graph has been generated manually. Currently, we work on a tool that generates a graph with: exceptional information, application states and distributed dependencies. Quantitative information was gathered automatically by counting the number of executions and exception throws.

The part of code contains starting points at lines: 9, 25, 34, 45 that start propagation paths (denoted PP). Propagation paths are presented in table 1, in which each path is described with nodes and node states from $DCFGE$.

The execution of the application was logged and control flows were counted. In total, 1000 invocations of the method Buyer.giveBuyOffer have been made with randomly generated request parameters and offer decisions. The invocations resulted in 849 regular correct outputs and exceptional outputs in other cases. In order to better demonstrate the use of quantitative characteristics, the code was injected with two extra errors: (i) a delay has been made in the client source to simulate latencies, (ii) the buyer refuses randomly to buy a declared product.

As it can be seen, the InvalidServerState exception occurs most often. The exception is thrown usually in the getOffer method ($PP20$). In rare cases, the server state changes between the call to getOffer and acceptBuyOffer ($PP14$). Cooperative handling occurs in $PP30_a$. Some control flows from $Line30$ do not perform cooperative handling, which depends on method input values.

Table 2. Dependencies in propagation paths

Path name	Path scope	Path dependencies
$PP9$, $PP14$	Buyer.run()	acceptBuyOffer, endOfSell, getOffer - synchronization, communication
$PP20$	Buyer.run()	acceptBuyOffer, endOfSell, getOffer- synchronization
$PP30_a$	lines 26:37	notifyBuy - communication, cooperative handling
$PP30_b$, $PP30_c$	lines 26:37	none

Potential influence of an exception on remote subsystems is calculated from method dependencies and propagation paths. Table 2 presents the scopes and dependencies of propagation paths.

5 Applications and Future Work

DCFGE is especially useful in software maintenance and improvement. Static control flow description presents information about the control flows with high anomaly propagation (i.e. a long sequential propagation path or many distributed dependencies). The control flows should be either modified or inspected in detail. Additionally, quantitative characteristics of execution are used to describe methods or subsystems, which allows to identify those areas that are especially prone to anomalies. The information is used to improve selected parts of the application. Static and dynamic description is used to compare alternative code constructs for their adequacy for a given problem.

The most important aspect of future work is the implementation of a tool that automatically generates *DCFGE*. Other research will focus on detecting those *EH* mechanisms that guarantee high reliability and are used often.

The work was supported in part by KBN under the grant number 4T11C 00525.

References

1. Hetch, M.S.: Flow Analysis of Computer Programs. (1977)
2. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graphs and its use in optimization. ACM Trans. on Progr. Lang. and Systems (1987)
3. Candea, G., Delgado, M., Chen, M.: Automatic failure-path inference: A generic introspection technique. In: 3rd IEEE Workshop on Internet Applications. (2003)
4. Brennan, P.T.: Observations on program-wide ada exception propagation. In: Annual International Conference on Ada. (1993)
5. Choi, J., Grove, D., Hind, M.: Efficient and precise modeling of exceptions for the analysis of java programs. ACM SIGLAN-SIGSOFT (1999)
6. Imagix Corporation http://www.imagix.com/products/lowlevel.html: Flow Charts and Control Flow Analysis. (2005)
7. Sinha, S., Harrold, M.J.: Analysis and testing of programs with exception handling constructs. IEEE Trans. Software Engineering (2000)
8. Cheng, J.: Dependence analysis of parallel and distributed programs and its applications. In: IEEE Conf. on Advances in Parallel and Distributed Comp. (1997)
9. Zhao, J.: Multithreaded dependence graphs for concurrent java programs. In: Int. Symp. on Software Engineering for Parallel and Distributed Systems. (1999)
10. Kaczmarek, P.L., Krawczyk, H.: Remote exception handling for pvm processes. LNCS 2840, X Conf. EuroPVM/MPI, Venice, Italy (2003)
11. Kaczmarek, P.L., Krawczyk, H.: Exception handling model with influence factors for distributed systems. LNCS 3019, V Conf. PPAM, Poland (2003)

Parallel Processing Subsystems with Redundancy in a Distributed Environment

Adrian Kosowski[1], Michał Małafiejski[1], and Paweł Żyliński[2]

[1] Department of Algorithms and System Modeling,
Gdańsk University of Technology, Poland
{kosowski, mima}@sphere.pl
[2] Institute of Mathematics, University of Gdańsk, Poland
buba@sphere.pl

Abstract. We consider the problem of dividing a distributed system into subsystems for parallel processing with redundancy for fault tolerance, where every subsystem has to consist of at least three units. We prove that the problem of determining the maximum number of subsystems with redundancy for fault tolerance is NP-hard even in cubic planar 2-connected system topologies. We point out that this problem is APX-hard on cubic bipartite graphs. At last, for subcubic topologies without units connected to only one other unit, we give a linear time 4/3-approximation algorithm.

1 Introduction

The concept of parallel processing with redundancy is well studied in literature [1, 5, 12]. The idea of a fault-tolerant processing system with redundancy is that a set of K machines (units) is assigned the task of solving a given problem P for a set of input test data I. The task is treated as indivisible among machines, i.e. the i-th unit receives the input pair (P, I) and independently generates output data O_i. It has to be assumed that the output produced by every unit may, with some small probability, be corrupt due to software or hardware malfunction. Certain time-critical applications require that the computational process may never be repeated, and consequently the considered system must show resilience and be able to recover a correct result despite faults in some part of the processing units.

In this paper we consider the problem of dividing a distributed processing system into subsystems for parallel processing with redundancy for fault tolerance. Each subsystem must be able to operate independently of all other processing subsystems, i.e. no two subsystems may share computing units. Every subsystem has to consist of at least three units, to allow for the detection of a faulty unit in a so called *voting process*, and for establishing which of the outputs obtained by units of the system is correct. In the case of heuristic computation, such a system also guarantees reduced probability of a major error in calculations, since the median of outputs of all nodes of the subsystem (or the mean of all outputs after discarding extremal values) can be treated as the final result of the subsystem.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1002–1009, 2006.

Model and problem definition. The connections between nodes of the system can be modelled in the form of a connection graph $G = (V, E)$, and every subsystem is assumed to be a connected subgraph of G with no fewer than 3 vertices. All subsystems are vertex-disjoint and capable of processing independent tasks. This does not mean, however, that they may not be employed concurrently for solving different subproblems of one larger problem, and the results of all subsystems may be combined in a final global communication and processing stage. The purpose of this paper is to characterise the problem of maximizing the number of independent fault-tolerant subsystems in the system graph for certain chosen system topologies. We first observe that a subsystem is legal in accordance with the adopted definition if and only if it has a subgraph in the form of the three vertex path P_3. The discussed problem may therefore be posed as packing the maximum possible number of vertex-disjoint paths P_3 in the system graph (the MaxP_3M problem for short). This problem has been the subject of intensive research of a both theoretical and practical nature due to its wide range of applications in parallel processing, distributed test cover [3], and guarding segments in grid connections [9].

Our results. We prove that the problem of determining the maximum number of subsystems with redundancy for fault tolerance is NP-hard even in cubic planar 2-connected system topologies. Next, we point out that this problem is APX-hard on cubic bipartite topologies. Finally, for subcubic topologies without units connected to only one other unit, we give a linear time 4/3-approximation algorithm for maximisation the number of fault-tolerant subsystems.

2 Problem Complexity for Different System Topologies

Following the observation asserting the one-to-one correspondence between the problem of dividing a distributed processing system into subsystems with the desired redundancy for fault tolerance and the maximum P_3-matching problem in the given connection graph of units, a formal definition of the considered problem is given below.

Redundancy for Fault Tolerance Problem (the RFT problem)

Instance: A connection graph of units $G(V, E)$.
Question: Are there at least k vertex-disjoint paths of order 3 in G?

However, it turns out that the RFT problem is NP-complete (and thus its maximisation version – the MaxRFT problem – is NP-hard) even for system topologies which are 2-connected, cubic, bipartite and planar. The idea of the NP-completeness proof is based upon reduction from three dimensional matching (3DM) problem and it is presented in detail in Section 4.

Theorem 2.1. *The* MaxRFT *problem is* NP*-hard in bipartite planar cubic unit topologies.*

It is worth pointing out that, as far as we know, our result appears to be the first classical problem in graph theory ever shown hard for cubic bipartite planar

graphs. Vertex coloring, edge coloring and most forms of domination are easy, and the complexity of other problems usually remains open.

Let us recall that even for the 3DM3 problem (with maximum degree at most 3) it is NP-hard to decide whether a maximum matching is perfect or misses a constant fraction of the elements [11]. Hence, using this result, a technique developed by De Bontridder *et al.* [3], and gadgets from Section 4, it is easy to prove the following theorem (due to the limited space we omit the proof).

Theorem 2.2. *The* MaxRFT *problem is* APX*-hard in bipartite cubic unit topologies.*

For general graphs, the best known approximation algorithm for the maximum P_3-matching problem, and thus for the MaxRFT problem as well, achieves 3/2-ratio [3], whereas in the case of cubic graphs, there is a quadratic 4/3-approximation algorithm (a 20-page proof can be found in [8]). Thus we get

Corollary 2.3. *There is a* $3/2$*-approximation algorithm for the* MaxRFT *problem, and in cubic system topologies, this ratio can improved to* $4/3$*.*

However, in the next section, we give an approximation algorithm which provides a major improvement on this result, as it can be applied to any connection graph with a known $(2,3)$-factor, that is, a spanning subgraph with the minimum degree at least 2 and the maximum degree at most 3. Although this class of system topologies seems to be restricted, note that in reality system topologies are usually dense enough to have a $(2,3)$-factor (cubic graphs, 4-regular graphs).

From now on, we shall refer to a *RFT-decomposition* as a set of vertex-disjoint paths of order at least 3.

3 An Approximation Algorithm for (2,3)-Regular Systems

The idea of our algorithm is based upon the well-known DFS algorithm – we modify it in order to get a *pendant-P_2-free* spanning tree of a connection graph, and the pendant-P_2-freedom will imply the existence of the RFT-decomposition of cardinality at least $\lceil \frac{n}{4} \rceil$.

Definition 3.1. *A graph is said to be* pendant-P_k-free *if it contains no path of* $k+1$ *vertices such that one of the end vertices of the path is of degree* 1 *in* G*, the other is of degree* 3*, while all other vertices of the path are of degree* 2*. In particular, a graph is* pendant-P_2-free *if none of its vertices of degree* 2 *is adjacent to both a vertex of degree* 1 *and a vertex of degree greater than* 2*.*

Corollary 1. *If a subcubic tree* T *of order* $n \notin \{1,2,5\}$ *is pendant-P_2-free, then there exists a P_3-packing in* T *of at least* $\lceil n/4 \rceil$ *paths.*

As a side note, let us recall that Masuyama and Ibaraki [10] showed that the maximum P_i-matching problem in trees can be solved in linear time, for any $i \geq 3$. The idea of their algorithm is to treat a tree T as a rooted tree (T, r)

(with an arbitrary vertex r a the root) and to pack i-vertex paths while traversing (T, r) in the bottom-up manner.

Corollary 3.2. *If system graph G has a spanning forest whose trees fulfill the assumptions of Corollary 3.1, then there exists a P_3-matching (and consequently also an* RFT*-decomposition) of cardinality at least $\lceil n/4 \rceil$. Such a decomposition can be found in linear time.*

Theorem 3.3. *There exists a linear time 4/3-approximation algorithm for the maximum P_3-matching problem for subcubic graphs without pendant vertices and without connected components of order 5.*

Proof. Consider a subcubic graph G of order n which fulfills the assumptions of the theorem. Taking into account Corollary 3.2, it suffices to show a linear-time algorithm for constructing a spanning forest in G whose trees fulfill the assumptions of Corollary 3.1. Such an algorithm is presented below.

1. For each connected component of G successively consider all edges e connecting vertices of degree 3. If removal of the edge from G does not create a new connected component of order 5, remove the edge from G and continue the process. Otherwise mark one of the endpoints of e as a cut vertex and proceed to the next connected component.
2. For each connected component H of G construct a Depth First Search (DFS) spanning tree T. The tree should be rooted following one of the rules below:
   (a) If H has a cut vertex v marked in Step 1, let v be the root of the tree.
   (b) If there exists an induced path (u_1, u_2, u_3) in H such that u_2 and u_3 are adjacent and of degree 2 in H, let u_1 be the root of the tree and let u_2 be the first vertex visited while recursing.
   (c) If neither rule (a) nor rule (b) can be applied, let any vertex of degree 2 in H be the root of the tree.
3. For each connected component H, if the resulting DFS tree T is not pendant-P_2-free, then for each DFS leaf v at the end of a pendant P_2, remove from T the edge incident to v and insert into T any other edge which is incident to v in H. The set of spanning trees obtained in this way (taken over all components of G) is the sought pendant-P_2-free spanning forest. □

Notice that the presented result holds for all graphs having a spanning subgraph with the property required in the assumptions of Theorem 3.

Corollary 3.4. *There exists a linear time 4/3-approximation algorithm for the* MaxRFT *problem in system topologies having a $(2, 3)$-factor without isolated components of 5 vertices.*

4 The MaxRFT Problem in Cubic Bipartite Planar Graphs

Dyer and Frieze [4] proved that the three-dimensional matching problem is NP-complete even for planar instances ($\overline{\text{3DM}}$ PROBLEM):

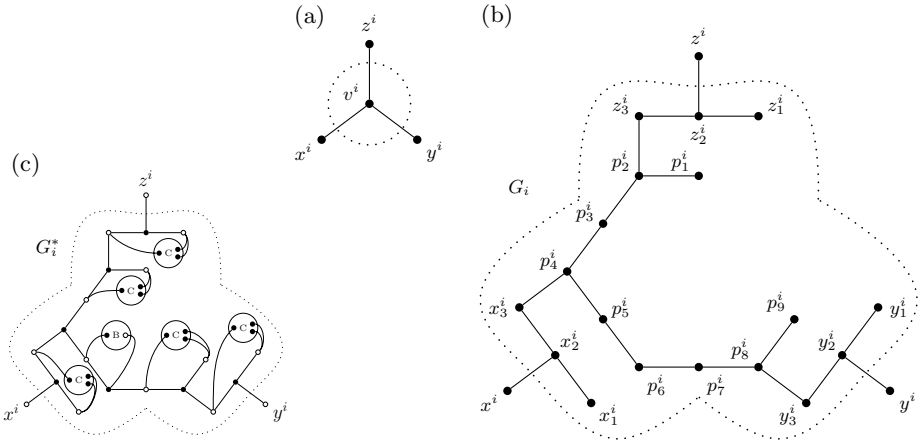


Fig. 1. (*a*) Vertex v^i is replaced by graph G_i (*b*). (c) After adjoining gadgets B and C all but x_i, y_i, z_i vertices are of degree 3.

Instance: A subcubic bipartite planar graph $G(V \cup M, E)$ without pendant vertices, where $V = X \cup Y \cup Z$, $|X| = |Y| = |Z| = q$. And, for every vertex $m \in M$ we have that $deg(m) = 3$ and m is adjacent to exactly one vertex from each of the sets X, Y and Z.
Question: Is there a subset $M' \subseteq M$ of cardinality q covering all vertices in V?

Theorem 4.1. [4] *The* $\overline{\text{3DM}}$ *problem is* NP*-complete.*

In this section, using Theorem 4.1, we will show that the problem of existence the RFT-decomposition of cardinality $\lfloor \frac{n}{3} \rfloor$ in cubic bipartite (2-connected) planar graphs is NP-complete. Our proof consists of two steps: first, we prove NP-completeness for "almost" cubic bipartite graphs, then we modify the constructed graph and show NP-completeness for cubic and planar graphs.

Theorem 4.2. *The perfect* P_3*-matching problem in cubic bipartite planar* 2*-connected graphs is* NP*-complete.*

Proof. Let $G(V \cup M, E)$ be a subcubic bipartite planar graph without pendant vertices, where $V = X \cup Y \cup Z$, $|X| = |Y| = |Z| = q$, every vertex $m \in M$ has degree 3, m is adjacent to exactly one vertex from each of the sets X, Y and Z. Next, let $G^*(V^*, E^*)$ be a graph obtained from G by replacing each vertex $v^i \in M$, $i = 1, \ldots, |M|$, (and all edges incident to it) with the graph $G_i(V_i, E_i)$ presented in Fig. 1, formally:

- $V^* = V \cup \bigcup_{i=1,\ldots,|M|} V_i$, where $V_i = \{p_j^i\}_{j=1,\ldots,9} \cup \{x_j^i, y_j^i, z_j^i\}_{j=1,2,3}$;
- $E^* = E \setminus E^- \cup E^+$, where $E^- = \bigcup_{i=1,\ldots,|M|} \{\{x^i, v^i\}, \{y^i, v^i\}, \{z^i, v^i\}\}$, $E^+ = \bigcup_{i=1,\ldots,|M|} (E_i \cup \{\{x^i, x_2^i\}, \{y^i, y_2^i\}, \{z^i, z_2^i\}\})$ and z^i are neighbours of vertex v^i in graph G.

Note that $x^i = x^j$ iff $dist(v^i, v^j) = 2$. Clearly, graph G^* is bipartite and it has $|V| + 18|M|$ vertices and $|E| + 17|M|$ edges, and $\Delta(G^*) = 3$.

Lemma 4.3. *There exists a solution of the* $\overline{\text{3DM}}$ *problem in graph* $G(V \cup M, E)$ *iff there exists a perfect* P_3*-matching of cardinality* $q+6|M|$ *in graph* $G^*(V^*, E^*)$.

Proof. ($\Rightarrow$) Let M' be a solution of the $\overline{\text{3DM}}$ problem in graph $G(V \cup M, E)$, $|M'| = q$. A perfect P_3-matching in graph G^* consists of the following paths:

- if vertex v^i corresponding to graph G_i is in M', then in graph G_i with attached vertices x^i, y^i and z^i (see Fig. 1) we choose the following 3-vertex paths: $x_1^i x_2^i x^i$, $y_1^i y_2^i y^i$, $z_1^i z_2^i z^i$, $x_3^i p_4^i p_3^i$, $y_3^i p_8^i p_9^i$, $z_3^i p_2^i p_1^i$, $p_5^i p_6^i p_7^i$;
- otherwise, if vertex v_j corresponding to graph G_j is not in M', then in graph G_j with attached vertices x^j, y^j and z^j we choose the following 3-vertex paths: $x_1^j x_2^j x_3^j$, $y_1^j y_2^j y_3^j$, $z_1^j z_2^j z_3^j$, $p_1^j p_2^j p_3^j$, $p_4^j p_5^j p_6^j$, $p_7^j p_8^j p_9^j$.

($\Leftarrow$) Let P be a maximum P_3-matching of cardinality $q + 6|M|$ in a graph $G^*(V^*, E^*)$ (P is perfect, of course). Let us consider any subgraph G_i, together with vertices x^i, y^i and z^i. First, let us note that x^i (or resp. y^i or z^i) cannot be a center of a 3-vertex path in a perfect matching. The following claim holds.

Claim. If one of the paths $x_1^i x_2^i x^i$, $y_1^i y_2^i y^i$ or $z_1^i z_2^i z^i$ is in matching P, then all of them are in matching P.

Consider, for example, $x_1^i x_2^i x^i \in P$ and $y_1^i y_2^i y^i \notin P$. Then $y_1^i y_2^i y_3^i$, $p_7^i p_8^i p_9^i$ and $p_4^i p_5^i p_6^i$ are in P, hence x_3^i is not covered by any path, a contradiction. The other cases can be proved analogously. Therefore, perfect P_3-matching P of graph G^* either (1) consists of paths $x_1^i x_2^i x^i$, $y_1^i y_2^i y^i$ and $z_1^i z_2^i z^i$ or (2) none of them is in this matching. So the set $M' = \{v^i \in M : G_i \text{ is covered in a way (1)}\}$ is a solution to the $\overline{\text{3DM}}$ in graph $G(V \cup M, E)$. □

Now, we will transform graph G^* to a bipartite graph where all vertices have degree 3 apart from x_i, y_i, z_i, which are of degree 2 or 3. We will need some additional gadgets. It's easy to verify the following properties of graphs B, A, L and C presented in Fig. 2, (the dotted edges are external ones, the bold edges belong to any perfect P_3-matching):

- every perfect P_3-matching of graph B uses only its internal edges, i.e., edges e_l and e_r are not used in any perfect matching of B,
- every perfect P_3-matching of graph A uses exactly one of edges e_l and e_r,
- every perfect P_3-matching of graph L always uses edge e_l,
- every perfect P_3-matching of graph C uses only its internal edges.

Let G^{**} be a graph obtained from G^* by replacing each graph G_i with a graph G_i^* presented in Fig. 1. Graph G^{**} has $|V| + 369|M|$ vertices and all but x_i, y_i, z_i have degree 3. It is easy to observe that there is a straightforward equivalence between any perfect P_3-matching in graph G_i and G_i^*. By Lemma 4.3, we get

Lemma 4.4. *There exists a solution of the* $\overline{\text{3DM}}$ *problem in graph* $G(V \cup M, E)$ *iff there exists a maximum* P_3*-matching of cardinality* $q + 133|M|$ *in graph* G^{**}.

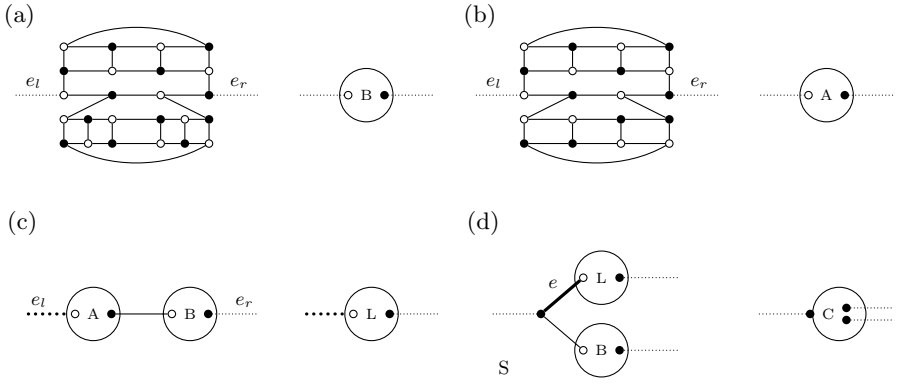


Fig. 2. (a) None of edges e_l and e_r is covered by any perfect P_3-matching of B. (b) Only one of edges e_l or e_r is used in any perfect P_3-matching of A. (c) Any perfect P_3-matching of L is using edge e_l. (d) None of three external edges of graph C are used by any perfect P_3-matching.

Now, we construct planar bipartite graph H with all vertices of degree 3. Note that every vertex $a \in X \cup Y \cup Z$ has degree 2 or 3 and the rest of the vertices of G have degree 3, so the number of vertices of degree 2 is divisible by 3. The idea is as follows: we attach a pendant vertex to every vertex of degree 2 and move it to the outer face of graph. Possible crossing of edges we eliminate by the following operation: if edge ab crosses the pendant edge $x_i x_i^*$, we replace both edges with the graph shown in Fig. 3. If edge ab belongs to some P_3-matching and a is an end vertex of some path abx, then we choose the following paths in the P_3-matching of graph H: av_1v_2, $v_3v_4v_5$, v_6bx (if vertex b is an end vertex, analogously). If edge ab does not belong to any 3-vertex path, we construct two paths $v_1v_2v_3$, $v_4v_5v_6$. Hence, any perfect P_3-matching of graph G^{**} can be easily transformed to a perfect matching of graph H and vice versa.

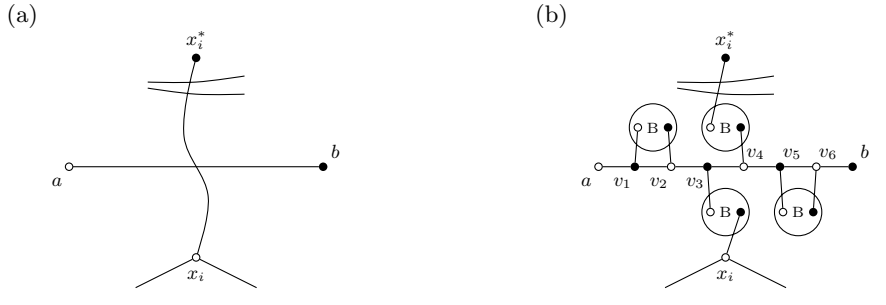


Fig. 3. An auxiliary graph used to eliminate crossing of pendant edge connected to vertex of degree 2

If all pendant vertices are on the outer face of graph, we attach a graph C to every triple of vertices from the same partition in such a way that pendant vertices belong to graph C. This operation is possible, because the number of vertices of degree 2 in both partitions must be the same and divisible by 3. Observe that such a graph is still bipartite and cubic (hence it is 2-connected). Obviously, all these operations can be done in polynomial time. From the above and Lemmas 4.3 and 4.4, the thesis of Theorem 4.2 follows. □

5 Conclusions

In the paper we study the problem of dividing a distributed processing system into the maximum number of subsystems for parallel processing with redundancy for fault tolerance. In the case when every subsystem has to consist of at least two units, the problem is equivalent to finding the maximum matching in the system topology, which can be solved in a polynomial time. In the model described in the paper every subsystem has to consist of at least three units, to allow for the detection of a faulty unit by way of a voting process. On the one hand we have presented a detailed analysis of the complexity status of the MaxRFT problem, on the other hand, we have proposed approximation algorithms for some network topologies.

References

1. P. Baptiste, V.G. Timkovsky, On preemption redundancy in scheduling unit processing time jobs on two parallel machines, *Operations Research Letters* **28** (2001), 205-212.
2. F. Berman, D. Johnson, T. Leighton, P.W. Shor, L. Snyder, Generalized planar matching, *J. of Alg.* **11** (1990), 153-184.
3. K.M.J. De Bontridder, B.V. Haldórsson, M.M. Haldórsson, C.A.J. Hurkens, J.K. Lenstra, R. Ravi, L. Stougie, Approximation algorithms for the test cover problem, *Mathematical Programming* **98** (2003), 477-491.
4. M.E. Dyer, A.M. Frieze, Planar 3DM is NP-complete, *J. of Alg.* **7** (1986), 174-184.
5. Y. I-Ling, E.L. Leiss, F.B. Bastani, Exploiting redundancy to speed up parallel systems, *IEEE Parallel & Distributed Technology: Sys. & Appl.* **1** (1993), 51 - 60.
6. M. Jackson, Problem Structure and Dependable Architecture, in *Architecting Dependable Systems* III, Springer 2005.
7. A. Kaneko, A. Kelmans, T. Nishimura, On packing 3-vertex paths in a graph, *J. of Graph Theory* **36** (2001), 175-297.
8. A. Kelmans, D. Mubayi, How many disjoint 2-edge paths must a cubic graph have?, *J. of Graph Theory* **45** (2004), 57-79.
9. M. Małafiejski, P. Żyliński, Weakly cooperative guards in grids, *Proc.* of ICCSA'05, LNCS 3480 (2005), 647-656.
10. S. Masuyama, T. Ibaraki, Chain packing in graphs, *Algorithmica* **6** (1991), 826-839.
11. E. Petrank, The hardness of approximations: gap locations, *Computational Complexity* **4** (1994), 133-157.
12. G.S. Shedler, M.M. Lehman, Evaluation of redundancy in a parallel algorithm, *IBM Systems Journal* **6** (1967), 142-149.

Dependable Information Service for Distributed Systems

Jan Kwiatkowski, Piotr Karwaczyński, and Marcin Pawlik

Institute of Applied Informatics,
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{jan.kwiatkowski, piotr.karwaczynski, marcin.pawlik}@pwr.wroc.pl

Abstract. A notion of a Dependable Information Service (DIS) is advocated as the means for applications in large scale distributed systems to select service instances meeting their requirements, functional as well as non-functional. Applications that need to provide certain correctness of results to the end user can only do so by requiring the services they use to guarantee particular level of correctness. When no such guarantees can be made, applications cannot provide the service and thus become unavailable. A system model is proposed to let applications temporarily accept lower correctness of used services in order to increase the availability caused by application accepted tradeoff. The proposed Dependable Information Service provides applications with a set of services appropriate for their demands and informs them of changes that can affect their quality level.

1 Introduction

The ability of a service provider hosted in a large scale distributed system (Grid [3], peer-to-peer networks) to fulfil requirements, imposed by the client application, varies greatly with time. Unpredictable events like changing rate of incoming requests, different network conditions, consumption of local resources by other services, node and link failures etc. all prevent the service provider from meeting the demands of many applications. A number of applications would benefit from being able to adapt to changes in the capabilities of services by controlled degradation of their own requirements or by utilizing better service providers. Since the service dependability is defined in the context of its users' requirements, making the application expectations flexible can result in the higher overall system dependability. As currently no solutions allow such a pattern, this paper attempts to present a new Information Service (IS) able to discover service providers not capable to exactly respond to the application requirements but still useful if the application expectations are lowered. We call it the Dependable Information Service (DIS). The mechanism is proposed to offer information reflecting the best state of valuable knowledge about current system state. The result is a dependable service-oriented distributed system where an application can depend on the information provided to it even in the overwhelmingly large and dynamic system.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1010–1017, 2006.

The proposed information service is a part of the solution being developed in the scope of the DeDiSys project[1]. The major project objective is to improve availability of data-centric and service-centric architectures by sacrificing application-specific consistency [7]. For service-centric applications, consistency equals correctness criteria imposed on services, e.g. their quality/dependability level, resources they are able to provide, etc. In case of failures or a system dynamism, the criteria can be temporarily lowered in order to keep a system available.

The rest of the paper is organized as follows. Section 2 presents a short overview of various network Information Services. In section 3 we describe typical IS usage scenario. Section 4 shows availability improvements present in our model of the Dependable Information Service. Section 2 presents its architectural design. Finaly, in section 6, we conclude with a summary of our work and its future directions.

2 Information Services Overview

The majority of distributed systems need some means to locate specific services (objects, components, etc. - for the purpose of this paper all of these entities will be referred to as services), possibly dispersed among different nodes and LANs. Services can be located by their attributes or - more generally - by requesting certain capabilities. With respect to them, they can formulate requirements of different kinds. The degree of complexity of the requested capabilities, as well as the dynamicity of the system environment, determines the level of sophistication of the necessary lookup mechanism.

The most basic form of requirements is a specification of the service name. The common service that provides this functionality is a naming service. Traditionally it implements direct, single level mapping between requested names and service locations. To this end, it maintains a set of bindings, which relate names to addresses. All service names must be consistent with a naming convention supported by the naming service. The examples of existing naming services include CORBA Naming Service and DNS.

An example of more advanced requirements is a set of attributes describing a service, instead of an explicit name. In order to support this kind of lookup, a directory service as an extension of a naming service was introduced. Directory services are simple databases. Instead of locating an entry only by name, these services allow clients to locate entries based on a set of search criteria. In general, they manage a directory of entries. An entry has attributes associated with it. An attribute consists of a name or identifier and one or more values. These attributes describe the entry, and the exact set of attributes depends on the type of the entry.

Both naming and directory services were designed with static target environments in mind. Any changes in locations of components, due to e.g. mobility

[1] Research supported by the European Community Framework Programme 6 project DeDiSys, http://www.dedisys.org/, contract No 004152.

or failures, result in stale information being served by them. Since dynamicity in large scale, wide area distributed systems usually cannot be avoided, discovery services appeared. The discovery service updates information it stores automatically as the network configuration changes. It reacts to component renewal messages or monitors components itself. The examples of the discovery service are Globus MDS (Monitoring and Discovery Service) and Jini Lookup Service.

3 Typical Information Service Usage Scenario

In a typical implementation, the application first opts for the maximum precision and requires information about a set of services that are able to deliver it. If the constraints on the required number of nodes can not be met, the application degrades its expectations and chooses to use some nodes that do not satisfy the constraints. The application monitors satisfaction of constraints and if they become unsatisfied, acts by lowering its requirements on some nodes or utilizing different nodes that satisfy the constraints, if available. A typical installation of the Globus Toolkit for the scenario described in previous section is illustrated in Fig. 1.

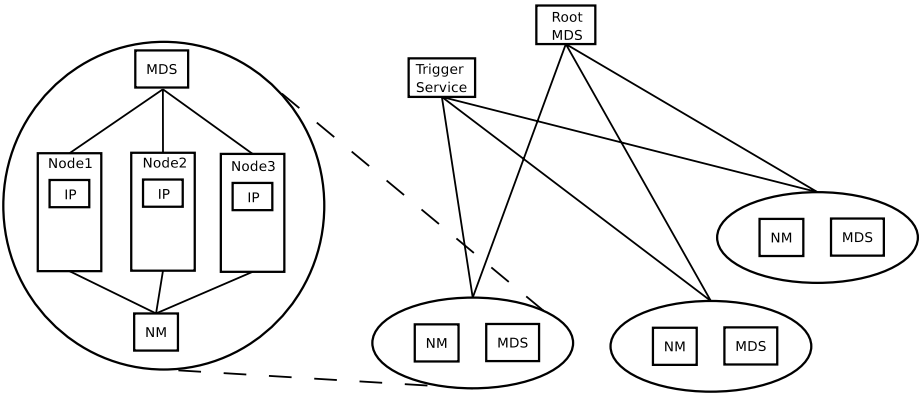


Fig. 1. Typical IS usage scenario

The root Monitoring and Discovery Service (MDS) of the Globus Toolkit gathers information from the local MDS instances. NodeMonitor (NM) subsystem registers information on available node resources and reports the changes to the application. NM subsystem can be created with Ganglia or Hawkeye and WS GRAM information providers, reporting to a dedicated Trigger Service. With this scenario in mind, the following problems that need to be addressed can be identified:

1. Discovery of service instances capable of delivering the desired capabilities
2. Monitoring the ability of service instances to provide the desired capabilities

3. Adjusting (degrading or upgrading) application requirements in accordance with the capabilities of the service instances
4. Rescheduling the workload to other service instances to meet the requirements if necessary and possible.

4 Dependable Information Service

Dependability is a general concept, and in different usage scenarios different attributes of dependability can be emphasized. The most significant attributes of dependability are reliability, availability, safety, and security [1]. Reliability deals with continuity of service, availability with readiness for usage, safety with avoidance of catastrophic consequences on the environment, and security with prevention of unauthorized access and/or handling of information. In our case the most important attributes are availability and reliability. Safety is more application dependant, although applications are always informed about the quality of DIS answers and thus are always able to act accordingly. Security depends on the actual system implementation and we do not discuss it here. To simplify our discussion we present only the way in which the availability can be increased, however in most cases the means to increase it will also increase reliability of the service.

To see how the system availability can be raised, we define the notion of the Quality of Environment (QoE). QoE describes how the system state corresponds to the application expectations. It can be seen as a function of available system resources, requested by particular application, and the system failures, resulting in inexact data provided by the DIS to the application. The quality of data provided by DIS will be measured by Quality of Answer (QoA) which defines how the answer provided by DIS corresponds both to the actual system state and application requirements. QoA is highest if all the resources requested by application are fulfilling its functional and non-functional requirements and the failure rate permits DIS to exactly answer application query. In this situation the answer exactly corresponds to the application query. In the classical scenario if the decrease of quality of environment ($dQoE$) exceeds some threshold (due to the rate of failures or absence of requested resources) QoA is lowered from the maximal value (exact answer) to the minimal one (the answer stating that the resources cannot be discovered or are not present). In our proposed system the possibility of approximate answers, based on stale data or data prediction, lets us deal with the situations when parts of DIS are not responding but the resources they were supposed to monitor are still available. The query language defines the quality of information provided about particular resource as a requirement imposed on the answer about that resource. The possibility to respond not only with the exact answer but also with the one offering the application information about resources which are of lower quality than it expects, lets us sacrifice the QoA to achieve higher system availability, moving the point where the system answer is of no use to the application from T_{low} to T_{high}.

5 Architecture of the Dependable Information Service

The idea presented above extends classical Information Service with the possibility to provide answers inexactly corresponding to the application query in order to improve availability. The query is compound and includes functional as well as non-functional (e.g. related to the expected accuracy or time of answer) requirements. An application specifies its requirements against desirable business service in a Requirements document consisting of three sections:

1. Requested functional capabilities (hard - minimally satisfactory, and soft - completely satisfactory)
2. Requested non-functional capabilities (hard - minimally satisfactory, and soft - completely satisfactory)
3. Satisfaction function

Requested functional capabilities (R_{CF}) define a set of functionalities that the requester expects from a service provider. Hard R_{CF} cannot be traded. If a service is not able to provide them, it is of no use to the requester. An example of such a capability is the ability to perform desired computations. On the other hand, soft R_{CF} can be traded. Such a capability can be, for example, that the data set returned to the requester is normalized. The requester prefers the normalization to take place, but without it is still able to utilize the result. Typical examples of a description of the required functional capabilities are class of the object in distributed object oriented systems (CORBA), or port type of a Web service [2].

Requested non-functional capabilities (R_{CNF}) describe capabilities that the service should provide. Hard R_{CNF} cannot be traded. A service that is not able to provide them is useless to the requester. For example, the application performing computations can require these to be finished in no more than 10 minutes with an expected error rate smaller than 5%. If these constraints are not met, the results cannot be used. Soft R_{CNF} define the satisfactory set of non-functional capabilities. If all of them are provided, the requester's needs are completely satisfied. These capabilities may be traded. E.g., for the computational application they could be: time to finish computations smaller than 1 minute and expected error smaller than 3%. If such capabilities are not provided, but the time and precision values are within hard R_{CNF} , the result can be accepted.

A satisfaction function represents the capabilities' quality metric analogous to the Rank expression in the ClassAds language [6]. A higher satisfaction function value indicates a better provider's offer. The actual method used to define the satisfaction function is based on the semantics used in the ClassAds language. For the example described in the previous paragraph, the satisfaction function could be, for instance, defined as: $SF = 0.5/other.expectedTime + 2/other.expectedError$. The function value is computed if all hard, but not all soft requested capabilities are satisfied. If hard capabilities cannot be provided, the offer is rejected and the minimal function value is assumed. Similarly, when both hard and soft requested capabilities are satisfied, the offer is fully satisfactory and the function value is treated as maximal. The ability to accept or

reject a service offer without further considerations speeds up the search process. From the formal point of view satisfaction function can be seen as a part of non-functional requirements, but as it is of special importance for our trade-off it is discussed separately.

The proposed draft architecture is built around the notion of a dependable information service, where service requesters ask for needed capabilities, and service providers advertise offered capabilities (Fig. 2).

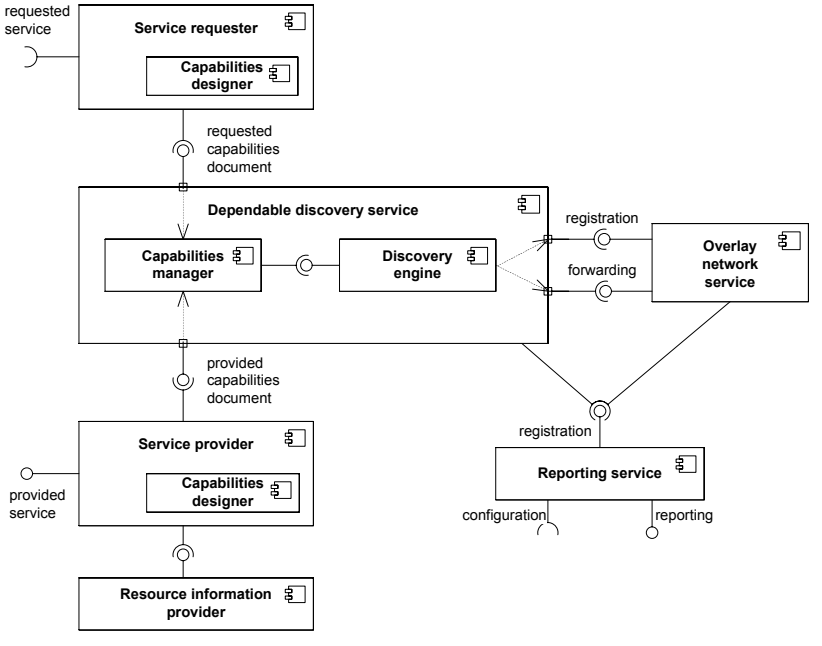


Fig. 2. Dependable Information Service

Service providers and requesters are distributed among system nodes. Providers register their capabilities in DIS via provided capabilities registration documents. The preparation of these documents is facilitated by capability designers and resource information providers (RIP). A capability designer is implemented in an application-specific way. It can simply import capabilities from a file or database or generate it algorithmically.

A requester willing to gain access to a certain service utilizes a capabilities designer to specify the detailed requested capabilities discovery document. This document contains a set of requested functional and non-functional capabilities, as well as a satisfaction function, and is forwarded to the DIS. The DIS in turn employs a capabilities manager to analyse it. A capabilities manager interprets the request, deals with priorities if necessary and uses a discovery engine to find a service completely satisfying the requested capabilities. In case it is not possible to find such a service, the requested soft capabilities are lowered and

a satisfaction function is used to identify services providing the requested capabilities best. A discovery engine cooperates with the Overlay Network Service (ONS) to find the requested service. The ONS provides application-layer routing based on the peer-to-peer paradigm[4],[5]. Currently, we envision the utilization of the decentralized object location and routing (DOLR) algorithms. The Reporting Service records the information about the operation of the DIS and ONS components. This information can be used to fine-tune their behaviour.

After the successful discovery of appropriate services, the service requester is provided with a proposed contract with the service provider. If it finds the proposed contract satisfying, it communicates with the service provider directly in order to establish the contract.When a service provider process starts, it retrieves the state of the resources from its local RIP in order to calculate the provided non-functional capabilities. These, as well as functional capabilities, are included in the PC registration document, which is sent to DIS. DIS registers the provided capabilities of the service and replies with a result of the registration, which may or may not succeed. On the other hand, a service requester wishing to discover an appropriate service, sends a document stating the requested functional and non-functional capabilities (RC query) that a service should provide to DIS. DIS matches the required capabilities against the provided capabilities of the discovered services and proposes a contract to the requester. If the contract is satisfactory to the requester, then it forwards the contract document to the service provider. Depending on scenario needs, a contract confirmation message can be used or not. The simple option is to provide the required and provided capabilities to DDS and let it do the matching automatically with the confirmation being implied by the match. However, in some scenarios, an explicit accept/reject message may be applicable and reasonable. In this case the provider may explicitly accept or reject the contract by sending a contract confirmation back to the requester. If the provider chooses to accept the contract, or the contract is implicitly accepted, then the requester may use the service until cancelling the contract with a contract cancellation document. DIS is notified about all changes of the capabilities of a service provider (e.g. after receiving and accepting a contract) with a PC update document.

The efficient and fault-tolerant information service is a focal point of the developed architecture. The Dependable Information Service (DIS) is an extension of a discovery service. The added value is an improved selection process: the service requester discovers the appropriate providers by specifying the precise requested capabilities, including critical needs (hard requested capabilities), fully satisfactory needs (soft requested capabilities), and information on priorities of needs (satisfaction function). With this information, DIS is able to accurately match the requested capabilities with the provided capabilities of the available service providers.

DIS is replicated on every node within the system. The overall information on application level system services is distributed among different DIS instances in a redundant way. Consistency of the information is provided by soft state mechanisms. The Dependable Information Service joins the ability to work in the

large-scale Grid environments, fault-tolerance resulting from distributed design and flexibility of user expectations to achieve higher dependability of the whole system.

6 Conclusions

In the paper a new approach to developing Information Service has been proposed. The new IS improves the overall dependability of the system, by allowing applications to flexibly state their requirements and trade them to achieve higher availability. The DIS itself is also highly fault-tolerant. Replicated among all the system nodes, it no longer appears as a single point of failure. After identifying the problems that appear in typical implementations, an extension to the classical Information Service framework has been proposed. The study of such an extension suggests that it alleviates some of the problems of the initial approach. Implementation of the proposed framework requires further study of usage scenarios in order to come to a proper level of abstraction to accommodate as wide a scope of applications as possible.

References

1. Avizienis A., Laprie J-C., Randell B., Landwehr C: Basic Concepts and Taxonomy of Dependable and Secure Computing IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, January-March 2004
2. Christensen E., F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language 1.1 W3C Note, 2001, http://www.w3.org/TR/wsdl
3. Foster, C. Kesselman, S. Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations nt. Journal of High Performance Comp. Applications, 15(3), 2001, Globus Alliance. GT 4.0 Information Services (MDS).
4. Karwaczyński P., J. Kwiatkowski J.: Analysis of Overlay Network Impact on Dependability Proc. 38th Hawaii International Conf. on System Sciences (HICSS'05), IEEE CS Press, 2005, pp. 290c - 290c
5. Karwaczyński P, Pipan G: On Availability Improvements in Unstructured Peer-to-Peer Systems Innovation and the Knowledge Economy: Issues, Applications, Case Studies, IOS Press, October 2005
6. Nabrzycki J. edt.: Grid resource management: state of the art. and future trends, Kluwer Academic, 2003
7. Smeikal R., Goeschka K.: Trading Constraint Consistency for Availability of Replicated Objects Proc. 16th Int'l Conf. on Parallel and Distributed Computing and Systems, 2004, pp. 232-237.

Fault-Tolerant Protocols for Scalable Distributed Data Structures

Krzysztof Sapiecha and Grzegorz Lukawski

Department of Computer Science, Kielce University of Technology, Poland
{k.sapiecha, g.lukawski}@tu.kielce.pl

Abstract. Scalable Distributed Data Structures (SDDS) consists of two components dynamically spread across a multicomputer: records belonging to a file and a mechanism controlling record placement in file space. Record placement in the file is SDDS specific mechanism. It is spread between SDDS servers, their clients and dedicated process called split coordinator. In the paper fault-tolerant protocols for SDDS components are given. The protocols use Job Comparison Technique along with TMR. Basic and extended SDDS architectures are compared with the help of SDDS oriented software fault injector. Time overhead due to redundancy introduced is estimated, too.

1 Introduction

SDDS [1] consists of two components dynamically spread across a multicomputer: records belonging to a file and mechanism controlling record placement in file space. Methods of making fault-tolerant records were discussed in [2-5]. As far as methods of making fault-tolerant controlling mechanism is concerned, the first solutions and estimations were done in [6]. It seems that fault tolerance of SDDS controlling mechanism is very important for system's dependability. Faults causing wrong record placement can lead whole application to crash, while faults concerning record data can cause invalid computations at most.

Controlling record placement in the file is SDDS specific mechanism. It is spread between SDDS servers, their clients and dedicated process called split coordinator, running on any multicomputer node.

In the paper extended SDDS architecture with fault-tolerant record placement is introduced and estimated. Section 2 introduces principles of SDDS. Section 3 describes SDDS controlling mechanism and its possible operational faults. In section 4 fault-tolerant protocols for SDDS components are given. Section 5 contains comparison considerations. The paper ends with conclusions.

2 Scalable Distributed Data Structure (SDDS)

A *record* is the least SDDS element. Every record equipped with an unique *key* is loaded into *bucket*. The buckets are stored on multicomputer nodes called *servers*. Every server stores one or more buckets. All the servers can communicate with each other. If a bucket is full it performs a *split*, moving about half of the records to a new bucket.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1018–1025, 2006.

Any number of machines called *clients* can simultaneously operate SDDS file. Each client is not aware of other clients' existence. Every client has its own *file image* (information about an arrangement and number of the buckets), not exactly reflecting actual file state. A client may commit *addressing error*, sending a query to inappropriate bucket. In that case the bucket *forwards* the query to appropriate one and the client receives *Image Adjustment Message* (IAM), updating this client's file image near to actual state. A client never is going to commit the same addressing error twice.

There is a single dedicated process called *split coordinator* (SC), controlling buckets' splits. The clients, the servers and the SC are connected with a *net*.

SDDS file grows as the amount of required storage space increases[1]. There is no need for central directory. Splitting and addressing rules are based on modified linear hashing (LH*) [1]. The rules may be implemented in two ways: centralized and decentralized. In the former case the SC is applied. In the latter case there is no SC and servers send each other a *token*, which allows for splitting a bucket actually holding it. Decentralized LH* was considered in [6].

3 Centralized LH* and Its Failures

LH* controlling mechanism is distributed between all SDDS elements. Below centralized LH* will be considered.
The client functions are as follows:

- Computing destination bucket address (may be wrong) for a given key, using local file image. Computing server physical address then and sending a query.
- Receiving a response message.
- Updating a file image using received IAM message, if there was any.

The server functions are as follows:

- Receiving incoming messages (sent by clients or servers).
- Verifying destination bucket address (check if the query was properly sent):
  - if correct, operation is performed and results are sent back to the client;
  - if incorrect, probable destination address is calculated and the message is forwarded. IAM message to the client is sent then.
- Sending a collision message to SC in case of bucket overload.
- Performing a split after a message from SC. Committing the split then.

The split coordinator functions are as follows:

- Maintaining real LH* file parameters.
- Collecting information about full buckets, implementing file scaling strategy.
- Controlling file splits - sending a split message to the bucket pointed with n pointer. Receiving split committing messages and updating file parameters.

[1] If all the buckets are stored in RAM memory then SDDS highly improves data access efficiency [1].

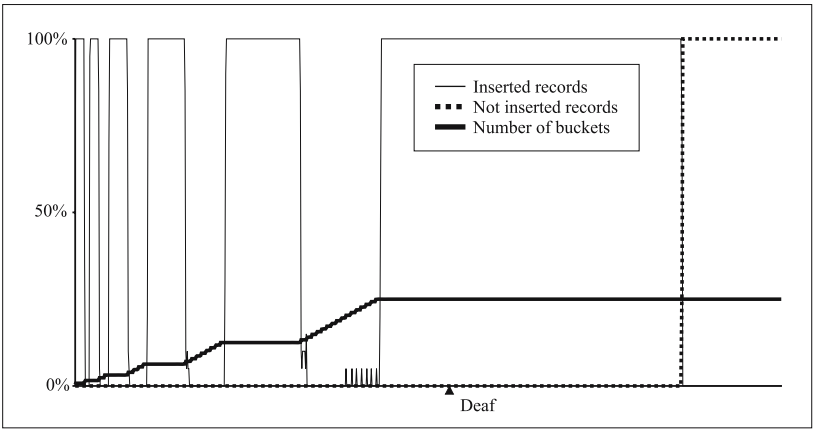


Fig. 1. Single 'deaf' split coordinator

Possible failures of SDDS system results in LH* operational faults. To investigate LH* behavior in fault conditions software fault injector called 'SDDSim' was developed. It is a single computer MS Windows/X-Window application, behaving as SDDS LH* file. Each of SDDS components (buckets, clients and SCs) runs as a separate process. Interprocess communication uses shared memory and semaphores to simulate network layer. SDDSim user is able to see SDDS mechanism at work, inject faults into each component, see system's reaction and then analyze its behavior using log files.

Sample SDDS LH* file was created and faulty situations were simulated. The file was able to expand to 128 buckets, each capable of storing 256 records. Two series of record insertions were done. Results are presented in form of two-dimensional graph. On X axis the progress (total number of messages send through the net) of simulation is shown. On Y axis it is shown how much of the net throughput is taken by messages of given type (what is actually transmitted through the net).

The result of making the SC 'deaf' is obvious. After some successful insertions the file became overloaded and it stopped to accept more data (Fig.1). Turning the SC 'berserk' led to chaotic bucket splits. Many of the insertion messages gone outside SDDS file bounds because of the corrupted file structure (Fig.2).

Summarizing, *client operational faults* are the following:

- The client breakdown. Such a client is not a danger for SDDS file.
- Wrong address calculation (client has gone berserk) - query would be sent to wrong server. Such fault may lead to file damage, as it was explained above.

Server operational faults are the following:

- Invalid recipient - the server received a message concerning a bucket stored on different server. There are two possibilities:

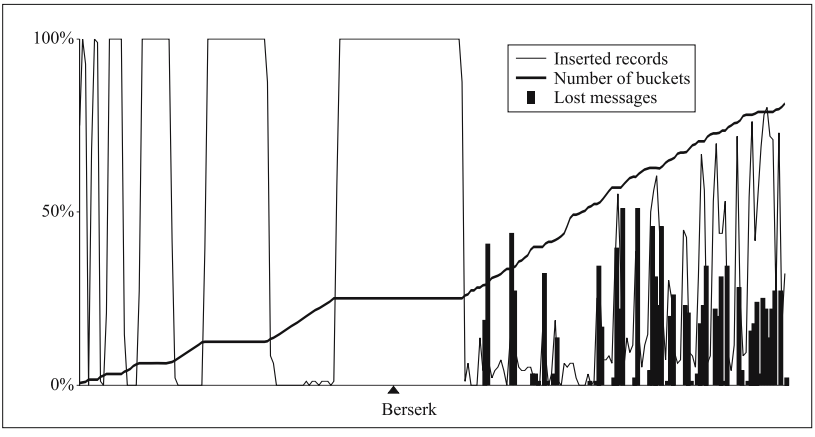


Fig. 2. Single 'berserk' split coordinator

  - The message is received by server having smaller id number than the proper server. Such a fault is not a danger and the message will be forwarded properly.
  - The message is received by server having bigger id. Basic SDDS LH* scheme for growing file would never forward a query to a server having id smaller than current one's id. This may lead to data access problems.
- Collision - bucket overload, nothing unusual for SDDS file, the server sends a collision message then.

Split coordinator operational faults are the following:

- Deaf SC - collision messages do not initiate bucket splits and SDDS file expansion. This leads to file overload.
- SC sending invalid messages (SC has gone berserk). Such a messages can cause splits incompatible with LH* scheme, this leads to file structure crash and massive data access problems.

4 Fault Tolerant Protocols for Centralized LH*

A multicomputer means a net of computers. Hence, rather obvious assumption is taken that at least three-node multicomputer is considered. Under such assumption all single transient or permanent faults of centralized LH* should be tolerated.

Proposed mechanism uses *Job Comparison Technique* (JCT) and *Triple Modular Redundancy* (TMR) [7] to arrange fault tolerant split coordination. Fault-tolerant bucket and SC protocol procedures were written in Promela language [8]. The source consists of about 100 lines of code. Most important fragments of it that concerns fault tolerance are presented below.

The bucket process is defined as follows:

```
proctype Bucket(byte id) {
       bool splitPool[3]; bool doSplit=0; int splitIgnore=-1;
       ...
BLOOP: bChan[id] ? msgOp, msgNum;     /* Message received */
       (msgOp == MSG_SPLIT) -> {
          ...              /* Ignore third correct split message */
          splitPool[msgNum] = 1;      /* Remember this decision */
       }
       /* Two identical decisions sufficient to perform a split */
       (splitPool[0] && splitPool[1]) -> { doSplit = 1; ... }
       ...                 /* The same for 1&2, 0&2 coordinators */
       (doSplit) -> {
          SPLIT();         /* Perform the split now */
          ...        /* Send commit to every active coordinator */
       }
       ...   /* Process other message types */
       goto BLOOP;
}
```

Split coordinator process is defined as follows:

```
proctype Coord(byte id) {
       int myDecision=-1, hisDecision=-1; bool faultDetected=0;
       ...
CLOOP: cChan[id] ? msgOp, msgNum;     /* Message received */
       (msgOp == MSG_COLL) -> {       /* Collision message */
          ...             /* Bucket to split address computation */
          bChan[destAddr] ! MSG_SPLIT,id; /* Split message send */
          myDecision = destAddr;        /* Remember my decision */
          /* Send my decision to the next coordinator: */
          cChan[(id+1)%numCoords] ! MSG_DECISION, myDecision;
       }
       /* Previous coordinator's decision received */
       (msgOp == MSG_DECISION) -> hisDecision = msgNum;
       /* Set fault flag if no other decision after a time-out */
       /* Two different decisions: */
       (myDecision!=-1 && hisDecision!=-1) -> {
          (myDecision != hisDecision) -> faultDetected=1;
          ...
       }
       /* In case of fault detection, run third SC: */
       (faultDetected && numCoords<3) -> {run Coord (2); ...}
       goto CLOOP;
}
```

Process initialization is the following:

```
init {          /* First bucket is created and two SCs are run */
   run Bucket (0); run Coord (0); run Coord (1); numCoords = 2;
}
```

Moreover, the following rules hold:

- Every bucket sends a collision message to every active SC.
- Each active SC sends split message to the bucket.
- The bucket performs a split if at least two identical decisions were received.
- SC number id (0 or 1 or 2) is error protected (error correcting code is used). Hence, we assume that it is always correct.

This LH* scheme allow invalid coordinators be replaced with new instances. As it was explained above, the client may commit addressing fault, sending a query to wrong server. Basic LH* scheme would forward such message to a proper server if the message was originally sent to the server (bucket) having too small id. We must define *backwarding*, if we want LH* tolerates every client addressing fault. Hence, the destination address *a'* should be calculated as follows:

a' = h_j(C);
if a' != a then
a" = h_{j-1}(C)
if a < a" and a" < a' then a' = a";
if a" < a and a < a' then a' = a";

The last (added) line does not allow the message to be sent beyond LH* file space. If destination address *a'* is smaller than current bucket address *a* then it is still valid and the message would be sent there.

5 Comparison Considerations

To compare basic and fault tolerant SDDS architectures the experiment with software fault injection was repeated for the same SDDS file. However, from the very beginning two SCs were run and JCT was used.

After first series of insertions one of the SCs was turn 'berserk', so the third SC started according to the procedure shown in section 4 and the file kept working properly. There was no single lost message and no visible activity pause.

Client's faults are not as dangerous as coordinator ones, but they leads to data access problems. The SDDS file was created again. First series of insertions were performed by properly working client. It was turn 'berserk' then. Forwarding and backwarding ensured that every message sent to any of the buckets (with no respect to the file image), reached the correct bucket at last. Unfortunately most of the network throughput was taken by forward/backward messages.

The SC is waked up only if some collisions were reported and there is a need to make a split. How often the SC is making a decision, depends on single bucket capacity and amount of incoming data. Our analysis includes only a new data

inserted into LH* file, because newly inserted data fills buckets and initiates splits. The frequency of LH* splits may be calculated as follows:

$$P = (D/(N * 1024)) * 3600 \tag{1}$$

where:
N denotes bucket capacity, represented as the number of records * 1024;
D denotes incoming data stream intensity in number of records per second;
P denotes the number of splits per hour (the frequency of SC's decisions).

The number of data messages is constant for any bucket capacity. During a split, overloaded bucket sends about half of its content to a new one. We assume that the number of messages for transferring data between buckets is identical to the number of messages needed to transfer data between a client and a bucket.

The number of IAM messages is not considered, it depends on the number of clients and their activity. It is relatively small and for larger buckets the number of addressing errors decreases [1].

Three SC variants are considered. Each one requires DC number of messages:

- Single coordinator: 1 collision + 1 split decision + 1 commit, $DC1 = 3$;
- Double coordinator with JCT: 2 collisions + 2 result comparison + 2 decisions + 2 commits, $DC2 = 8$;
- Triple coordinator with TMR: 3 collisions + 3 result comparison + 3 decisions + 3 commits, $DC3 = 12$.

Total number of SC and data messages can be calculated as follows:

$$\Sigma\ DCn = DD + P * DCn, for\ n = 1, 2, 3 \tag{2}$$

where: DD denotes the number of messages and commits per hour.

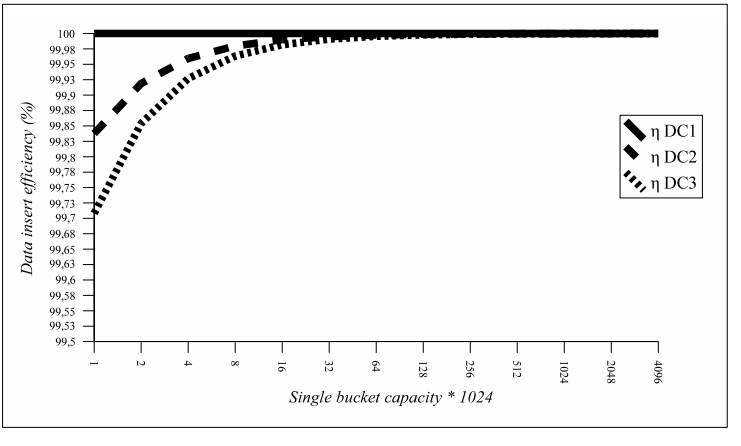


Fig. 3. The efficiency decrease caused by additional SC messages (compared to single coordinator SDDS)

Efficiency decrease caused by additional SC messages for JCT and TMR variants can be calculated as follows (Fig.3):

$$\eta DC2 = (\Sigma\ DC1/\Sigma\ DC2) * 100\% \quad (3)$$

$$\eta DC3 = (\Sigma\ DC1/\Sigma\ DC3) * 100\% \quad (4)$$

Theoretical estimations were verified experimentally with the help of the fault injector. 100000 of records were inserted into SDDS file working in two modes, first with single then with triple SC. The experiment was repeated 5 times and the best and the worst values were dropped. Finally, average time for each of the mode was calculated. It was 13.97s for the first and 14.02s for the other of the modes. This meant that the triple SC worked 99.64% as efficient as the single one what agreed with estimated 99.7% (Fig.3, 1k-bucket).

6 Conclusions

Split coordination is a crucial SDDS mechanism. As it was proved, its breakdown results in SDDS file damage and data corruption. The extensions of SDDS architecture we have discussed here results in SDDS dependability.

The application of JCT and TMR allows the SDDS works correctly in spite of any single SC fault. Proposed protocols were verified with software fault injector presented here, the results proved their usability. Time and cost overhead is low.

As it was shown, the number of additional messages required for double or triple SC is so small that it has no meaning compared to the number of messages transferred during basic SDDS scheme activity. It becomes even smaller as the bucket capacity increases. It is worth to mention that our analysis was focused on new data inserts only. In real system with more types of operations performed (search, delete, modify) decreasing SDDS efficiency seems to be even smaller.

References

1. Litwin, W., Neimat, M-A., Schneider, D.: LH*: A Scalable Distributed Data Structure. ACM Transactions on Database Systems ACM-TODS, December 1996.
2. Litwin, W., Neimat, M-A.: High-Availability LH* Schemes with Mirroring. Intl. Conf. on Coope. Inf. Syst. COOPIS-96, Brussels 1996.
3. Litwin, W., Risch, T.: LH*g: a High-availability Scalable Distributed Data Structure by record grouping. U-Paris 9 Tech. Rep., May 1997.
4. Litwin, W., Menon, J., Risch, T.: LH* Schemes with Scalable Availability. IBM Almaden Research Rep., May 1998.
5. Litwin, W., Schwarz, T.: LH*RS: A High-Availability Scalable Distributed Data Structure using Reed Solomon Codes. CERIA Res. Rep. & ACM-SIGMOD 2000.
6. Sapiecha, K., Lukawski, G.: Fault-tolerant Control for Scalable Distributed Data Structures. Annales Universitatis Mariae Curie-Sklodowska, Informatica, 2005.
7. Pradhan, Dhiraj K.: Fault-Tolerant Computing: Theory and Techniques. Prentice-Hall, Inc, May 1986.
8. Basic Spin Manual: http://spinroot.com/spin/Man/Manual.html

Quantifying the Security of Composed Systems

Max Walter and Carsten Trinitis

Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR),
Technische Universität München, D-80290 München, Germany
Max.Walter@in.tum.de, Carsten.Trinitis@in.tum.de

Abstract. The authors recommend to quantify the security of a complex system by first quantifying the security of its components, and, in a second step, by calculating the overall security according to a given method. This paper summarizes the state of the art of security measures for components and presents a new method for combining these measures into the system's security. The proposed method starts with an intuitive graphical representation of the system. This representation is converted into an algebraic expression using abstract AND, OR, and MEAN operators. Applying application-dependent semantics to these operators will allow for an evaluation of the model.

1 Introduction

From a rigorous point of view, no computer system is secure. For instance, any real-world message encryption may be successfully attacked because its key-length is finite, because the implementation will probably contain errors and vulnerabilities, and because some users may not chose their passwords carefully enough.

However, it is obvious that some systems are more secure than others. Thus, it would be interesting to have some kind of measure for the security of a given system. Such a measure could be used for several purposes: First, to quantify the risk of operating a system. For example, a bank might decide to provide an online banking service only if the probability of a successful attack lies below a given threshold. Second, besides security, there are other so-called non-functional properties like e.g. performance, costs, reliability, availability, safety, etc.. Optimizing a system towards one of these properties may deteriorate other properties. For example, making a system more secure by using encrypted data exchange will certainly cost money and performance. Additionally, the encryption algorithms may be implemented incorrectly which may make the system less reliable. Thus, designing a system is always a tradeoff between several non-functional properties. To find an optimal tradeoff, these properties (including security) must be quantified. Third, we might want to compare two designs/implementations with respect to their security.

Obtaining a meaningful security measure for a given system or design is obviously a difficult task. We therefore advocate a divide-and-conqueror approach which is well known from the area of reliability/availability modeling. For this

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1026–1033, 2006.

purpose, the system is mentally decomposed into several subsystems. Each of these subsystems is then in turn again decomposed, until the system is broken up into basic components (hardware and software) for which availability/reliability values are known or can be measured, modeled or (in the worst case) guessed.

In this paper, we propose a similar approach for gaining information on the security of a given system. For such an approach, we need:

- a meaningful measure for the system's security, which can also be applied to each of the system's components,
- methods to obtain these measures for the basic components, and
- a method to compute the overall system's security from the components.

This paper mainly contributes to the third issue (see Sec. 3-5). In the next section, the state of the art concerning the first two issues is summarized, categorized, and discussed.

2 Security Measures and Metrics

Several measures for secure systems have been presented in the literature, including adversary work factor [1], adversary financial expenses [2, 3], adversary time [4] probability like measures [5, 6], or simply defining a finite number of categories for secure systems.

We can discriminate these measures into the type of scale used (either discrete or continuous scales) and the range of this scale (either bounded or unbounded). Without loss of generality, we can map all bounded scales to the interval $[0; 1]$ and all discrete scales to continuous scales. Thus, if a modeling method supports bounded continuous scales with bounds $[0; 1]$, it will support all bounded scales.

Unbounded continuous measures include adversary work factor, time, and expenses. For example, we could measure the security of a door by the mean time the attackers need to break through this door. In the general case, the time interval is not deterministic but randomly distributed following a certain probability distribution. Reasonable distributions include:

- The exponential distribution, which can be used for repeated attacks where neither the attackers nor the defenders can learn from unsuccessful attacks. The exponential distribution is memoryless. Thus, the attackers' success to break into the system within the time interval $[t, t + \Delta t]$ is independent on t. The exponential distribution is defined by a single parameter called λ. The security of the door, i.e. its mean time to successful attack is given by λ^{-1}.
- The Weibull distribution can be used for systems which age due to attacks or become more secure due to attacks, i.e. for systems with increasing or decreasing strength. This behavior is modeled by an additional aging parameter α. For the special case $\alpha = 1$, the Weibull distribution equals the exponential distribution. If α is less than 1, the system becomes stronger with time, if α is greater than 1 the system ages and becomes weaker.

– The normal distribution can be used to model differently skilled attackers. We assume that the average attackers' time to break into the system is μ. Additionally, the parameter σ controls the variance of the distribution.

Other distributions are also imaginable depending on the application domain. In particular, combinations are possible, e.g. a door whose mean time to a successful attack and aging parameter are itself random variables which follow a normal distribution.

As we work towards an application independent modeling method in this paper, we can not narrow our scope to a certain measure. However, without loss of generality, we can assume that all basic security measures of system components are either from:

– a bounded continuous scale with bounds $[0; 1]$, or
– an unbounded random variable with a given distribution function from which a mean time to successful attack can be derived.

3 Security in Medieval Times

In the middle ages, security was obtained by building castles. To be useful in times of peace, castles possess doors, which we assume are the only targets for the attackers in times of war. In this section, we show how the security of a castle with up to two doors can be computed under the assumption that the security of each door is known.

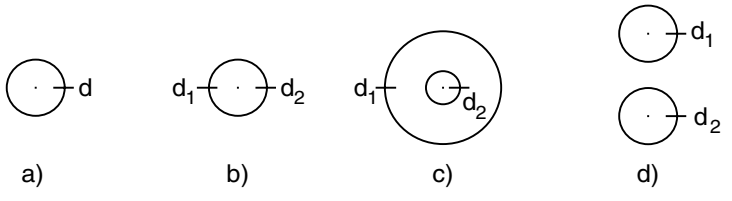


Fig. 1. Four different medieval castles

Castle a) in Fig. 1 is the simplest castle we can think of. It has a wall (depicted as a circle), a treasure room (depicted by a dot) which is the attackers' main target, and a door d, which is the only way for the attackers to get into the castle. Thus, the security of the castle equals the security of the door.

The wall of Castle b) has two doors d_1 and d_2, and we assume that d_1 is weaker than d_2. The two doors allow the attackers to strike the castle at two points simultaneously. Thus, the castle's security will be weaker than or equal to the security of d_1 (We assume that attacking d_2 may weaken the defense at d_1).

In contrast, the security of castle c) may be stronger than the security of a castle with only one door. Here, the attackers must break into two doors to get into the treasure room. If we again assume that d_1 is weaker than d_2, the castle's security is at least as good as the security of d_2.

In the last example (castle d)), the attackers have two castles they may choose to attack. We consider an attack to be successful if the attackers get into one of the two treasure rooms. However, the distance of the castles is too large to allow for a simultaneous attack of both castles. Thus, the security of both castles will be in the interval $[d_1; d_2]$.

4 Security of Systems with 2 Components

Formalizing the ideas from the previous section, the following pieces of information are needed to compute the security of a (medieval or modern) system:

- a security Measure M
- the securities of its components (or doors): $d_1, d_2, \ldots, d_n$
- a function: $d : M^n \rightarrow M$, which maps the security of the doors to the security of the overall system

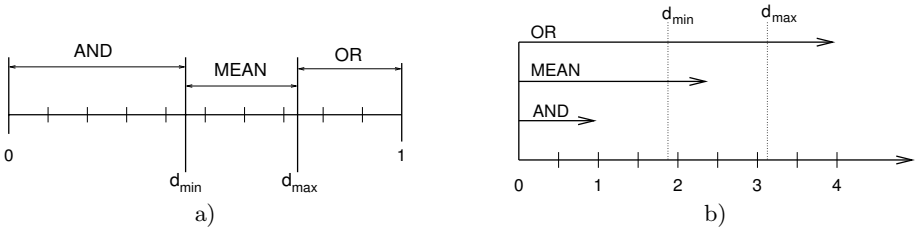


Fig. 2. Three classes for possible operators: AND, MEAN, OR

To deal with the complexity of today's systems, we assume that the function s can be depicted as a term using different binary operators. A modeling method for secure systems is thus constructed by first defining a security measure and second defining a set of operators to combine the components' measures. Fig. 2a shows a generic operator for the bounded continuous case with interval $[0; 1]$. The operator maps 2 operands $d_{1,2} \in [0; 1]$ to a single element $d \in [0; 1]$. If we call d_{min} the smaller operand and d_{max} the larger operand, we can classify the possible operations into:

- AND-Operations, if $0 \leq d \leq d_{min}$
- MEAN-Operations, if $d_{min} \leq d \leq d_{max}$
- OR-Operations, if $d_{max} \leq d \leq 1$

If both operands are unbounded random variables with expected value d_{min}, $d_{max} \in [0; \infty]$, the operators can be similarly classified. (see Fig. 2b). The only difference is that the result of an OR-operation may be arbitrarily large ($d_{max} <= d$).

4.1 AND-Operations

An AND-Operation should be used for a system which resembles castle b) from Fig. 1. In this case, the defenders have to defend both doors. Thus, the system is as weak as or weaker than d_{min}.

If M is in $[0;1]$ and we assume that the event of breaking into a door is statistically independent, we can calculate the security of an AND-system by $d_{AND} = d_{min} \cdot d_{max}$ which is the expected value of a probabilistic event that both doors are successfully attacked. Assuming independency is pessimistic from a defenders' point of view. It can be used for systems in which the defenders are weakened by being responsible for two doors at the same time whereas the attackers can attack both doors at the same time without additional efforts or costs. In the case of dependent doors (i.e. two doors are built using the same technique for the lock), the number of possible vulnerabilities becomes smaller. Thus, dependencies among the doors will strengthen the system in this case.

For the unbounded random case, we can compute the mean time to successful attack of the AND-system if the probability distributions of both doors are known. If $d(t)$ is the probability that a door withstands an attack of length (or effort) t we can compute $d_{AND}(t)$ of the system by:

$$d_{AND}(t) = d_{min}(t) \cdot d_{max}(t)$$

if the doors are statistically independent from each other. For example, if the lifetime of both doors is exponentially distributed with a mean $\lambda_{min,max}^{-1}$, $d_{AND}(t)$ is also exponentially distributed, because of

$$d_{AND}(t) = d_{min}(t) \cdot d_{max}(t) = e^{-\lambda_{min} t} \cdot e^{-\lambda_{max} t} = e^{-(\lambda_{min}+\lambda_{max})t},$$

Furthermore, if the lifetimes are deterministic, we can easily derive $d = min(d_{min}, d_{max})$. Depending on the distributions, calculating $d_{AND}(t)$ and/or deriving d may be impossible using closed-form expression. However, the result may be approximated with Monte Carlo simulations in a possible tool implementation.

4.2 OR-Operations

For systems corresponding to castle c) in Fig. 1, OR-operations can be used. To defend the castle, either d_{min} or d_{max} has to be defended. A corresponding system is at least as strong d_{max}.

If the security measure is in $[0;1]$ (and again, breaking the doors are statistically independent events), we calculate the security of an OR-system by:

$$d_{OR} = 1 - (1 - d_{min}) \cdot (1 - d_{max}))$$

This time, dependencies amongst the doors will weaken the system. E.g. if both doors are protected with a similar type of lock, lock-picking the second lock might be much easier after successfully opening the first door.

Under the assumption that the security of a door is measured by an unbounded random value (i.e. a mean time to successful attack), and that both doors are attacked simultaneously (e.g. by long range weapons) d_{OR} can be computed via $d_{OR}(t)$ which is defined as:

$$d_{OR}(t) = d_{min}(t) + d_{min}(t) - d_{min}(t) \cdot d_{min}(t)$$

Again, the computation of the expected value di_{OR} of $d_{total}(t)$ depends on the probability distribution of $d_{min}(t)$ and $d_{max}(t)$, and can be obtained by simulation in the general case. For the deterministic case, under the assumption that the doors are attacked one after another, i.e. that the second door cannot be attacked before the attackers successfully broke the first door, the security of the castle can be computed by $d = d_{min} + d_{max}$.

4.3 MEAN-Operations

In some systems, the attackers may first choose from one of two doors and, in a second step, attack the system as if it had only one door. This scenario was introduced before as castle d) in Fig. 1. Clearly, such a system is stronger than any system with two doors which can be attacked concurrently, but weaker than any system where the attackers must break into two doors. Thus, its security lies in the interval $[d_{min}; d_{max}]$.

Therefore, it is straightforward to apply a mathematical mean operation to model these kinds of systems, which has the same property ($d_{min} \leq d \leq d_{max}$). If the attackers randomly choose a door with equal probability 0.5 for each door, the security of the system is:

$$d_{MEAN} = \frac{d_{min} + d_{max}}{2}$$

which is the arithmetic mean of d_{min} and d_{max}. In a more general case, the attackers might have some knowledge which doors are more vulnerable and prefer the doors with a lower security. In other scenarios, the defenders will have some information on the attackers' preferences and be able to strengthen the doors which are most likely to be attacked. In this case, it is more likely that the attackers choose the strong door. Both scenarios can be taken into account for using the general power mean M_p defined as:

$$d_{MEAN} = M_p = \left(\frac{d_{min}^p + d_{max}^p}{2}) \right)^{\frac{1}{p}}$$

The parameter p determines the amount of knowledge the attackers and defenders have. If p equals one, the power mean equals the arithmetic mean, i.e. neither the attackers nor the defenders have an influence on which door is chosen. If p becomes greater, the knowledge of the defenders increases and thus the probability that the attackers choose the strong door. For example, if p is 2, M_p becomes the root of squared means , which can be computed by:

$$d_{MEAN} = M_2 = \text{RSM} = \sqrt{\frac{d_{min}^2 + d_{max}^2}{2}}$$

In the extreme case, i.e. $p \to \infty$, the attackers always choose the strong door and thus $d_{MEAN} = d_{max}$.

If p is smaller than 1, the attackers know the castle well and the weak door is more likely to be under attack. For example, if p is 0, M_p is called the geometric mean G defined as:

$$d_{MEAN} = M_0 = G = \sqrt{d_{min} \cdot d_{max}}$$

In the other extreme case, i.e. $p \to -\infty$, the attackers will choose the weakest door and thus $d_{MEAN} = d_{min}$.

The mean operations previously proposed in the literature [5] include the weakest link, which is equivalent to $M_{-\infty}$, the weighted weakest link, and priorized siblings. In the latter two, the modeler specifies weights in $[0; 1]$ which determine the preferences of the attackers towards choosing a certain door. However, choosing the right weights is very difficult in practice, which might be an advantage for the method proposed here, as it requires only one parameter: p.

5 Security of Systems with n Components

Fig. 3a shows a castle with eight doors and two treasure rooms. Again, we define that the attackers are successful if they are able to get into one of the two treasure rooms. A semantically equivalent representation of the system is shown in Fig. 3b. This figure was created by starting at the doors, which now form the leaves of a tree. Then, the nodes were repeatedly connected in pairs by an OR, AND or MEAN gate, respectively. For system evaluation, we can replace the abstract gates by some formal ones. E.g., for a simple evaluation, we replace each OR function by a minimum-operator, each AND-function by the maximum operator, and each MEAN-function by the arithmetic mean function.

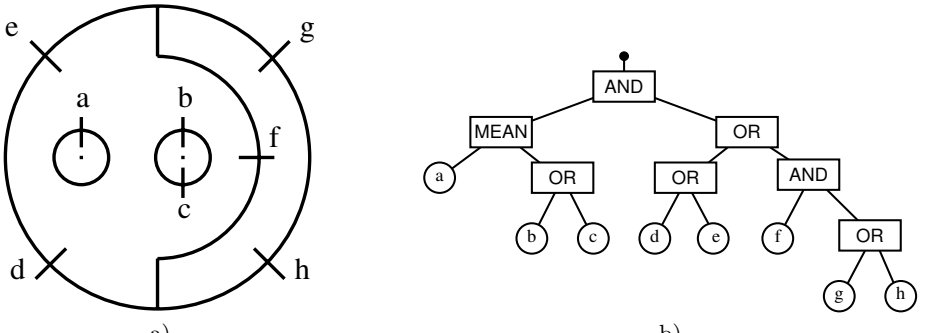


Fig. 3. Example of a castle with eight doors and corresponding system tree

6 Summary

This paper presents a step towards modeling secure system by a divide and conquer approach as it is widely known from the area of reliability/availability modeling. For this purpose, basic security measures known from the literature were enumerated and categorized in Sec. 2. Then, Sec. 3-5 explain the basic ideas of how to compute a system's security from the securities of its components. Repeatedly, two security measures of components are combined into a single value by binary operators.

This paper proposed several new binary operators for doing this and classifies them into three groups: OR, AND and MEAN operators. It also shows that all operators presented in previous works can be categorized in the same way.

References

1. Schuedel, G., Wood, B.: Adversary Work Factor as a Metric for Information Assurance. In: Proc. of the New Security Paradigm Workshop. (2000)
2. Schneier, B.: Attack trees. Dr. Dobb's Journal (1999)
3. Schneier, B.: Secrets and Lies – Digital Security in a Networked World. Wiley and Sons (2000)
4. Jonsson, E., Olovsson, T.: A quantitative model of the security intrusion process based on attacker behavior. IEEE Transactions on Software Engineering **23** (1997) 235–245
5. Wang, C., Wulf, W.: Towards a Framework for Security Measurement. In: Proc. of the National Information Systems Security Conference (NISSC 97). (1997)
6. Kotenko, I., Mankov, E.: Experiments with Simulation of Attacks against Computer Networks. In: Proc. of the International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security. Volume 2776 of Lecture Notes in Computer Science., Springer-Verlag (2003) 187–198
7. Nicol, D.M., Sanders, W.H., Trivedi, K.S.: From Dependability to Security. IEEE Transactions on Dependable and Secure Computing **1** (2004) 48–65
8. Tidwell, T., Larson, R., Fitch, K., Hale, J.: Modeling internet attacks. In: Proc. of the 2001 IEEE Workshop on Information Assurance and Security. (2001) 54–59
9. Steffan, J., Schumacher, M.: Collaborative attack modeling. In: Proc. of the 17th ACM Symposium on Applied Computing (SAC 2002). (2002) 253–259
10. Hunstad, A., Hallberg, J.: Design for securability - Applying engineering principles to the design of security architectures. In: Proc. of the Workshop of Application of Engineering Principles to System Security Design (WAEPSSD). (2002)
11. Voas, J., Ghosh, A., McGraw, G., Charron, F.: Defining an Adaptive Software Security Metric from a Dynamic Software Failure Tolerance Measure. In: Proc. of the 11th Annual Conference on Computer Assurance (COMPASS'96). (1996)
12. Levi, D.: Lessons learned in using live red teams in IA experiments. In: Proc. of the DARPA Information Survivability Conference and Exposition. Volume 1., IEEE (2003) 110–119
13. Vaughn, R.B., Henning, R., Sira, A.: Information Assurance Measures and Metrics - State of Practice and Proposed Taxonomy. In: Proc. of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03). Volume 9. (2003)

Increasing Dependability by Means of Model-Based Acceptance Test inside RTOS*

Yuhong Zhao, Simon Oberthür, Norma Montealegre,
Franz J. Rammig, and Martin Kardos

Heinz Nixdorf Institute, University of Paderborn, Paderborn, Germany

Abstract. Component-based self-optimizing systems can adjust themselves over time to dynamic environments by means of exchanging components. In case that such systems are safety-critical, the dependability issue becomes paramountly significant. This paper presents a novel model-based runtime verification to increase dependability for the self-optimizing systems of this kind. The proposed verification approach plays a role of an alternative acceptance test transparently integrated in RTOS, named model-based acceptance test. The verification is performed at the level of (RT-UML) models representing the systems under consideration. The properties to be checked are expressed by RT-OCL where the underlying temporal logic is restricted to either time-annotated ACTL or LTL formulae. The applied technique is based on the on-the-fly model checking, which runs interleaved with the execution of the checked system in a pipelined manner. More specifically, for ACTL formulae this means an on-the-fly solution to the NHORNSAT problem, while in the case of LTL formulae, the emptiness checking method is applied.

1 Motivation

Mechatronic systems represent a special class of complex cross-domain embedded systems. The design of such systems involves a combination of design techniques and technologies used in the mechanical and electrical engineering as well as in computer science. The increasing complexity, even emphasized by the system heterogenity, is one of the major problems in today's mechatronic industry (e.g., automotive industry). To deal with this complexity one approach is to build mechatronic systems in a self-reflecting, self-adapting and self-optimizing way. In the Collaborative Research Center 614 "Self-optimizing concepts and structures in mechanical engineering" we are investigating such an approach. The main focus is put on self-optimizing applications with highly dynamic software components which are optimized and even replaced at runtime. Moreover, the considered applications run under real-time constraints (see Fig. 1). As failures of these technical systems usually have severe consequences, dependability is of

* This work is developed in the course of the Collaborative Research Center 614 - Self-Optimizing Concepts and Structures in Mechanical Engineering - Paderborn University, and is published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1034–1041, 2006.

paramount importance. This puts new demands on verification of such complex and highly dependable systems.

For real-time systems with a dynamic task set, acceptance tests with respect to schedulability are state of the art in RTOS. In reconfigurable and dependable systems the safety and consistency after component replacement has to be checked as well. This extends the classical area of on-line acceptance testing. Traditionally in real-time systems one tries to execute as many checking activities as possible off-line. In systems of dynamic structure this would mean that all components that may be used in a substitution have to be checked (e.g., using conventional model checking) to be correct in an arbitrary context, i.e., in the most general context. Of course this very general correctness requirement would result in highly over-dimensioned and thus inefficient components, what would be a contradiction to the overall objective of self-optimization. Therefore we decided to develop a novel technique for on-line model checking context-specific parts of components at runtime. This verification technique is presented as an acceptance test service of the underlying RTOS. The real-time restrictions make it necessary to perform the on-line model checking at the level of (UML) models, to assume that the models are implemented correctly, and to assume that any non context specific internals of components have been verified off-line.

2 Related Work

A dependable operating system has to be robust (error free) and must support fault tolerance. Thus, it has to deal with error detection in the temporal and the value domains. Classical methods for error detection are: monitoring task execution times, double execution of tasks and watchdogs [1]. Furthermore dependable real-time operating systems must provide a predictable service to the application task. The worst case administrative overhead of the operating system must be known so that the temporal properties of the behavior of the complete host can be determined analytically. To make such an analysis, the real-time operating system must be very careful in supporting dynamic services like dynamic task creation [2]. Reconfigurable systems demand task creation at runtime. As the main responsibility of any operating system is the activation of tasks, the scheduling policy is of high importance. In addition, overload conditions have to be considered. Scheduling schemes for overload are divided into: best effort, guarantee and robust. The last class, suitable for hard real-time systems, considers different policies for task acceptance and for tasks rejection. Whenever a task enters a system, i.e., implied by the reconfiguration, an acceptance test verifies the schedulability of the new task set based on the worst case assumption. If the new set is found schedulable, the new task is accepted in the ready queue; otherwise, one or more tasks are rejected based on a different policy [3]. Before a safety critical system can be put into operation with a RTOS, it has to pass a rational analysis. Unfortunately, fully automatic verification environments that cover the complete system from the high level specification to the hardware are beyond the current state of the art [1].

3 Case Study

Let's take a typical example (Fig. 1) to show how the model-based runtime verification mechanism is applied to self-optimizing systems with safety-critical requirements. Suppose a real-time application contains four components A, B, C and D running in parallel. Now, due to the change of environment, a substitution request is passed to the RTOS at time point t_r that the component C should be replaced by component E at the t_d'th time step after t_r. Before the replacement is really done at time point $t_r + t_d$, the RTOS will request the model-based runtime verification service to check if the system still keeps safety and consistency after the replacement. According to the response from the verification service, Yes, No or $Unknown$, the RTOS will decide to accept or reject the requirement for substitution.

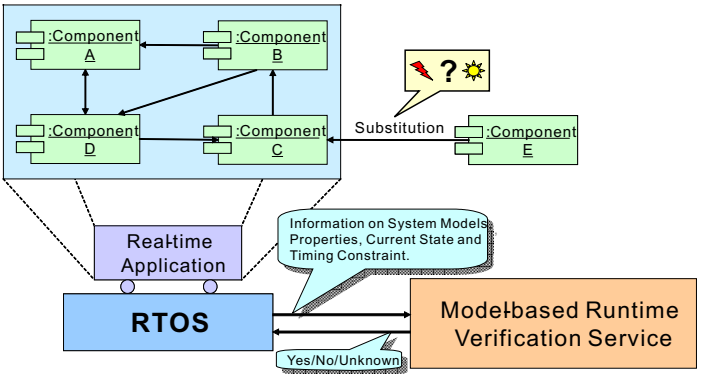


Fig. 1. Case study

Obviously, the substitution of the component E for the component C will cause the environment of each component in the system to be changed at runtime directly (i.e., B, D and E) or indirectly (i.e., A). For component-based systems, each component is verified *correct* under the given assumptions to the environment of the component. As in our case study, the environment of each component in the system might be changed dynamically due to the runtime reconfiguration. The following question arises: "Does the changed environment still satisfy the required assumption?". To answer this question, traditional model checking unfortunately is not suitable any more: on the one hand, it is difficult to predict how and when the reconfiguration will happen; on the other hand, it is difficult to check the safety and consistency of the reconfiguration within the given timing constraint. In practice, it is irrealistic to check off-line all the possible cases of the reconfigurations due to the huge time and space complexity. To our knowledge, the state of the art runtime verification [4, 5, 6] is not suitable for our needs. On the one hand, only linear temporal logic formulae as well as trivial assertions and invariants can be checked by tracing the program execution. On the other hand, potential errors can be detected only when they have

really happened. Due to the above reasons, we propose a model-based runtime verification mechanism working as an acceptance test of RTOS, by which we are able to on-line check the safety and consistency of self-optimizing systems at model level so that the potential errors can be predicted and thus avoided in time before they really happen.

4 Model-Based Acceptance Test

4.1 Overview

Note that our runtime verification service is working at model level. That is, both the models and the properties to be checked must be known beforehand. We can achieve this by applying the design technique [7] presented within the Collaborative Research Center 614 to design the self-optimizing systems based on the modeling concepts of UML 2.0 with the CASE tool suite Fujaba[1]. As a result, the architecture of such a system is specified by a component diagram together with the definitions for ports and connectors; the overall behavior of the system is specified by UML state machines with real-time extension, called real-time UML statecharts [8], associated to each component, port and connector. In addition, the properties required to each component are specified in real-time OCL [9] at the design phase by developers.

In order to apply the model checking technique in an efficient way, we transform the real-time UML models of the system and the associated real-time OCL constraints into *Kripke* structures and *Büchi* automata respectively, and then store them in a repository in advance. Thus, whenever a verification request from the RTOS is received (Fig. 1), the verification service can fetch the related *Kripke* structures and *Büchi* automata from the repository and immediately start the on-the-fly verification. For this purpose, the real-time UML statecharts and the related real-time OCL constraints exported from the Fujaba Tool Suite are translated into the corresponding AsmL [10] models and time-annotated ACTL/LTL formulae (including trivial assertions and invariants) respectively. Then, the *Kripke* structure of each AsmL model is derived by applying to the AsmL model the exploration functionality of the AsmL tool suite and the time-annotated ACTL/LTL formulae are transformed into *Büchi* automata. The time-annotated ACTL formulae are just RTCTL (Real-time Computation Tree Logic) [11] formulae with only universal quantifiers allowed. The time-annotated LTL is defined in a similar way. However, to avoid the fairness conditions caused by the *eventuality* operators, we require that the *eventuality* operators must be bounded ones if any. In this way, the bounds on the *eventuality* operators prevent indefinite postponement.

The self-optimizing operation may cause the system to be reconfigured at runtime in many ways. This paper mainly concerns such a case that one component is replaced with another one. Obviously, the replacement may change the environment of every active component in the system directly or indirectly.

[1] www.fujaba.de

On the other hand, the only constraint on the components replaceable with each other is that they must follow the compatible protocols, which means that the protocol of the new one must be the same as or the refinement of the old one. Therefore, it is quite necessary to invoke the runtime verification service to make sure that such a reconfiguration is really safe and consistent.

Without loss of generality, suppose that a real-time system model M contains n components $C_1, C_2, \cdots, C_n$ $(n \geq 2)$ working in parallel and is requested at time point t_r to replace the component C_k $(1 \leq k \leq n)$ with another component C'_k at time point t_d relative to t_r, denoted as $M' = M(C'_k/C_k)@(t_r \triangleright t_d)$. Accordingly, let B' be the new property automaton to be satisfied by M'. Consequently, the goal of our runtime verification is to check within the time interval t_d starting from t_r if $M(C'_k/C_k)@(t_r \triangleright t_d) \models B'$.

To do this, we adopt on-the-fly model checking in a top-down way for both ACTL and LTL formulae so that a pipelined working manner can be applied between the verification service and the real-time application to gain more execution time for verification. Simply speaking, we do on-the-fly ACTL model checking by checking the simulation preorder between M' and B' incrementally [12]. That is, the decision problem of checking simulation preorder is converted into the satisfiability problem for weakly negative Horn formulae, called NHORNSAT problem. The basic idea is to encode the properties of the simulation relation between M' and B' into a type of CNF (Conjunctive Normal Form) formula Γ, i.e., weakly negative Horn formula, and then prove on-the-fly that the CNF formula Γ is satisfiable in polynomial time. [13] presents an efficient on-the-fly algorithm that receives one Horn clause at a time and allows fast queries about the satisfiability of the whole formula so far received. Similarly, a dualization of the algorithm in [13] also gives an efficient linear time on-the-fly solution to the NHORNSAT problem [12]. As for the on-the-fly LTL model checking, we follow the emptiness checking technique [14]. However, we need some extensions to make the above on-the-fly model checking cooperate seamlessly with the application via RTOS as intermediary in a pipelined manner. Due to the limited space, the details are omitted here, but can be found in [15].

Of course, to make our model-based runtime verification feasible, the implementation of each component in the system must conform to the corresponding model of the component. In our design environment this is automatically achieved by using Fujaba to generate code directly from the design model. Therefore, the implementation of a component is the refinement of the model of the component or, put it another way, the model is the abstraction of the corresponding implementation. Thus, an ACTL (LTL) formula being *true* at the model level implies that it is also *true* at the implementation level, while it being *false* at the model level does not imply that it is also *false* at the implementation level. That is, our runtime verification is conservative due to its being applied to model level. However, the benefit of predicting and avoiding errors is gained just due to its being applied to model level. Note that we implicitly assume that the components under consideration own finite state machines and that they have been prechecked *correct* under the given assumptions on the environments

they depend on during the design phase. In addition, the processing speed for verification is assumed to be faster enough than that for application.

4.2 Pipelined Working Principle

It is easy to see that the timing constraint is the main barrier for our model-based runtime verification. To leap over this barrier, we adopt a pipelining technique to gain more execution time for verification. The sequence diagram in Fig. 2 illustrates the cooperation between the verification service and the real-time application. More precisely, the pipelined working mode is done between the RTOS and the verification service and thus transparent to the application.

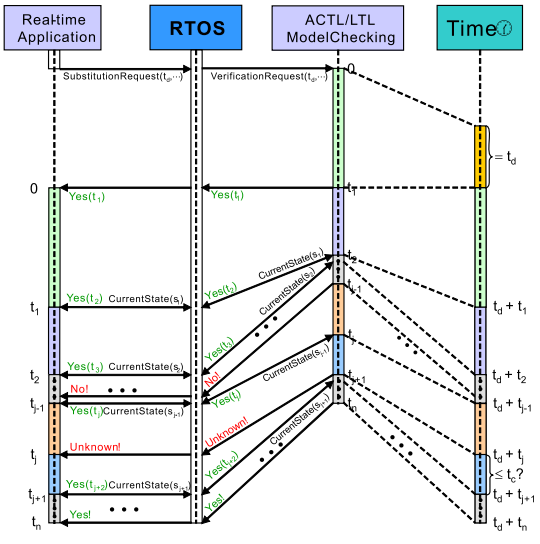


Fig. 2. Pipelined working principle

Whenever the RTOS receives a component substitution request from the application, it will invoke the verification service to check if the substitution is legal or not. The answer must be given within the required timing constraint, say t_d, in our example. If lucky, the verification may finish the checking task before the timing constraint is over. Unfortunately, it might be not the case for more complex systems. Therefore, it is quite possible that, within t_d time units, only the next t_1 time steps starting from the initial states are checked Yes, which means that the substitution is safe up to the coming t_1 time steps. In this case, the RTOS does allow the application to make the substitution and execute forward t_1 time steps. During this period, the verification continues to check, say the next $t_2 - t_1$ time steps. Accordingly, the application can then go ahead the next $t_2 - t_1$ time steps. Note that at each time point $t_d + t_i$ $(i \geq 1)$ with respect to t_r, the application can report its current state, say s_i, to the verification. Based on this runtime information, the verification

can locate in the system model the corresponding state with respect to s_i and thus avoid checking the whole state space of the system model by only checking a sufficient sub-space reachable from this specific state mapped from s_i. In this way, the computation load of the verification can be reduced to a greater extent.

The above process is repeated. If to some time point an error is detected, then the verification can be terminated with the answer No to the RTOS. Another situation occurs when at some time point, say $t_d + t_{j+1}$ relative to t_r, the checking result is still positive, but the time interval $t_{j+1} - t_j$ is less or equal to the pre-defined time constant t_c, which denotes the minimum time steps that the verification must keep ahead of the application. In this case the verification process has to be stopped and $Unknown$ is reported to the RTOS. Note that these two cases only mean that the errors might happen in the future, because we do not know if the errors are spurious or not. To avoid that the errors really happen, we have to conservatively choose to reject the substitution request and inform the application that an error might emerge in the future. That is, an exception will be raised by the RTOS together with a counterexample if necessary. It is possible to let the application to handle the predicted failure in this case, because failure recovery is integrated into the self-optimizing application itself. Finally, if a sufficient sub-space that covers this actual run of the real-time application is successfully checked, then we can report definitely Yes to the RTOS and terminate the verification process. From now on, the application can guarantee to execute safely and consistently after the substitution. In fact, Fig. 2 just illustrates an ideal pipelined cooperation between the application and the verification via the RTOS as intermediary without considering any implementation details.

5 Conclusion

This paper presents our ongoing research on model-based acceptance test by means of runtime verification at model level, which will be applied to the self-optimizing systems. In fact, our model-level runtime verification can be seen as an extension to the state of the art runtime verification. Our runtime verification can check time-annotated ACTL and LTL properties, but is not really limited to them. By introducing the pipelined cooperation between the verification and the application via the RTOS as intermediary, we can check if a component substitution still keeps the safety and consistency at runtime, provided that a constraint on the checking time is required. Up to now, we have not measured yet what the performance of our runtime verification looks like. Nevertheless, experience demonstrates that the properties to be checked in practice are usually not very complex. Therefore, the size of the *Büchi* automata derived from these properties tends to be reasonable, what makes our model-based acceptance test applicable to the self-optimizing systems.

References

1. Kopetz, H.: Real Time Systems, design principles for distributed embedded applications. (1997)
2. Maehle, E., Markus, F.J.: Fault tolerant dynamic task scheduling based on dataflow diagram. International Paralell and Distributed Processing Symposium (1997)
3. Buttazzo, G.: Hard Real-Time Computing Systems. (2000)
4. Barnett, M., Schulte, W.: Spying on components: A runtime verification technique. In Leavens, G.T., Sitaraman, M., Giannakopoulou, D., eds.: Workshop on Specification and Verification of Component-Based Systems. (2001)
5. Chen, F., Rosu, G.: Towards Monitoring-Oriented Programming: A Paradigm Combining Specification and Implementation. In: Proceedings of the 2003 Workshop on Runtime Verification (RV 2003), Boulder, Colorado, USA (2003)
6. Havelund, K., Rosu, G.: Java PathExplorer — a runtime verification tool. In: Proceedings 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS'01), Montreal, Canada (2001)
7. Giese, H., Tichy, M., Burmester, S., Schäfer, W., Flake, S.: Towards the Compositional Verification of Real-time UML Designs. In: Proceedings of the European Software Engineering Conference (ESEC), Helsinki, Finland (2003)
8. Giese, H., Burmester, S.: Real-time Statechart Semantics. Technical Report tr-ri- 03-239, Computer Science Department, Paderborn University (2003)
9. Flake, S., Mueller, W.: An OCL Extension for Real-time Constraints. In Clark, T., Warmer, J., eds.: Object Modeling with the OCL. Number 2263 in LNCS, Heidelberg, Germany, Springer Verlag (2002)
10. Gurevich, Y., Schulte, W., Campbell, C., W.Grieskamp: AsmL: The Abstract State Machine Language Version 2.0. (http://research.microsoft.com/foundations/AsmL/)
11. Emerson, E.A., Mok, A.K., Sistla, A.P., Srinivasan, J.: Quantitative temporal reasoning. In: Proceedings of the 2nd International Workshop on Computer Aided Verification, London, UK, Springer-Verlag (1991) 136–145
12. Shukla, S., Rosenkrantz, D.J., Hunt III, H.B., Stearns, R.E.: A HORNSAT Based Approach to the Polynomial Time Decidability of Simulation Relations for Finite State Processes. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society **35** (1997)
13. Ausiello, G., Italiano, G.F.: On-line algorithms for polynomially solvable satisfiability problems. J. Log. Program. **10**(1) (1991) 69–90
14. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Form. Methods Syst. Des. **1**(2-3) (1992) 275–288
15. Zhao, Y., Oberthür, S., Kardos, M., Rammig, F.J.: Model-based runtime verification framework for self-optimizing systems. In: Proceedings of the 2005 Workshop on Runtime Verification (RV'05), Edinburgh, Scotland, UK (2005)

A Cache Oblivious Algorithm for Matrix Multiplication Based on Peano's Space Filling Curve

Michael Bader and Christoph Zenger

Dept. of Informatics, TU München,
80290 München, Germany

Abstract. Cache oblivious algorithms are algorithms that are designed to inherently exploit any kind of cache memory—regardless of its size or architecture. In this article, we discuss a cache oblivious algorithm for matrix multiplication. The elements of the matrices are stored according to a Peano space filling curve. A block recursive approach then leads to an algorithm where memory access to matrix elements is strictly local. Consequently, the algorithm shows several interesting properties considering cache performance, prefetching strategies, or even parallelization.

1 Introduction

One of the current problems concerning the efficient use of available computing power is the increasing gap between processor speed and the speed of memory access. To overcome this speed difference, caches are used to accelerate the access to frequently used data. This poses an additional demand when developing software—algorithms that do not consider the presence of cache memory will not be able to use a satisfying portion of the available performance. In fact, very often the performance achieved by straightforward algorithms is rather poor.

In linear algebra, this situation has been – rather successfully – tackled by libraries such as BLAS, LAPACK, and similar. Heavily optimized, cache aware implementations, such as ATLAS[8], GotoBLAS[5], or even vendor supplied libraries, are consistently competing for optimal performance on different hardware platforms. However, in addition to cache optimization, processor specific instruction sets and code optimizations have to be taken into account plus lots of additional hardware-specific aspects. This has made ATLAS, for example, turn to a self-tuning approach, where an optimal implementation for specific hardware is obtained by a generic optimization process that can pick the best implementation or algorithm from several options.

This makes it quite hard for any new implementation to come up and try to beat the runtime of the existing libraries. Thus, in this paper we will set our sights lower, and concentrate on only one aspect of the optimization queue: locality of data access, which leads to inherently cache efficient implementations.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1042–1049, 2006.

2 Peano Curves and Data Locality

Space filling curves have become a valuable tool in many applications, where data locality is an issue. A well known example in scientific computing is the parallelization of data obtained from the discretization of partial differential equations. In this context, space filling curves have quasi-optimal locality properties[9]. In the following, we will demonstrate how to exploit these locality properties in the context of matrix multiplication.

2.1 A Stricty Local Scheme for 3×3-Matrices

Neither the commonly used schemes to store matrices – row-major and column-major –, nor a straightforward implementation of the matrix multiplication such as

```
for i from 1 to n do
   for j from 1 to n do
      C[i,j] := 0;
      for k from 1 to n to
          C[i,j] := C[i,j] + A[i,k] * B[k,j];
      end do;
   end do;
end do;
```

is motivated by aspects of data locality. Row-major and column-major storage allow the fastest access to single element in a matrix, and the nested for-loops in the given algorithm are probably the most simple solution when using the most common programming languages.

To optimize locality of data access, we should rather start from a more neutral formulation of the algorithm, such as the following:

```
// matrix C is assumed to be initialized
for all triples (i,j,k) in {1..n}x{1..n}x{1..n} do
   C[i,j] := C[i,j] + A[i,k] * B[k,j];
end do;
```

With absolutely no constraints to execution order, we can start to design an execution order that ensures optimal locality of data access. Let us consider the following matrix multiplication, where all three matrices are ordered in an improved way.

$$\underbrace{\begin{pmatrix} a_0 & a_5 & a_6 \\ a_1 & a_4 & a_7 \\ a_2 & a_3 & a_8 \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} b_0 & b_5 & b_6 \\ b_1 & b_4 & b_7 \\ b_2 & b_3 & b_8 \end{pmatrix}}_{=:B} = \underbrace{\begin{pmatrix} c_0 & c_5 & c_6 \\ c_1 & c_4 & c_7 \\ c_2 & c_3 & c_8 \end{pmatrix}}_{=:C} \tag{1}$$

To find a suitable order of execution, we use a graph representation. The nodes of the graph are given by the triples (p, q, r), where each triple represents an operation $c_r := c_r + a_p * b_q$. Two nodes of the graph will be connected by

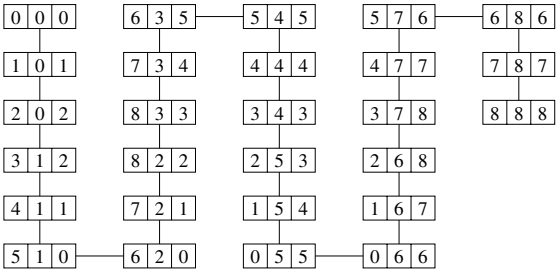


Fig. 1. Graph representation of the operations of the 3×3 matrix multiplication given in equation (1)

an edge, if data access remains local: the difference between two indices must not be larger than 1 in any of the components p, q, and r.

The respective graph is shown in figure 1. It leads immediately to an optimal order of execution. While this result might not seem to be too extraordinary, it is remarkable in the sense that it is not possible to obtain a similar scheme for 2×2-matrices!

In the following, we will extend this scheme to the multiplication of any two matrices of odd dimensions. Therefore, we have to extend the mapping of the matrix elements to a more general case, and afterwards introduce a block recursive scheme for the multiplication.

2.2 Mapping Matrix Elements to Memory

The indexing scheme in equation (1) was, of course, motivated by a Peano space filling curve. We will now introduce a general scheme that uses iterations of a Peano curve to map matrix elements to memory. Figure 2 illustrates the recursive construction principles for the Peano curve. Note the four different patterns marked as P, Q, R, and S. The patterns will be used to characterize the numbering of a matrix-block:

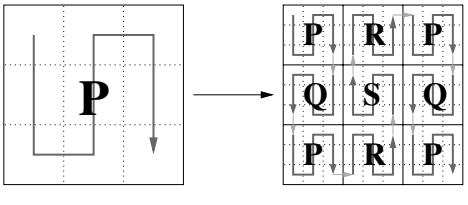


Fig. 2. Recursive construction of a Peano curve (first two iterations)

- in a P-numbered block, the numbering starts in the top-left corner, and ends in the bottom-right corner;
- a Q-numbered block is numbered from the top-right to the bottom-left corner,

– an R-numbered block is numbered from bottom-left to top-right, and
– an S-numbered block is numbered from bottom-right to top-left.

Then the numbering scheme can be described in terms of a grammar, as illustrated in figure 3. Note that the numbering scheme is not restricted to square matrices as might be suggested by figure 2. Every block will be split in its larger direction (rows or columns). For odd dimensions, any block with more than 7 rows or columns will be decomposed into three blocks of, again, odd dimensions (not necessarily of equal size). Thus, we can construct a respective block numbering where the smallest blocks have at most m rows and n columns—where $m, n \in \{3, 5, 7\}$. Matrices of even dimensions will have to be padded by a zero column or row (or both).

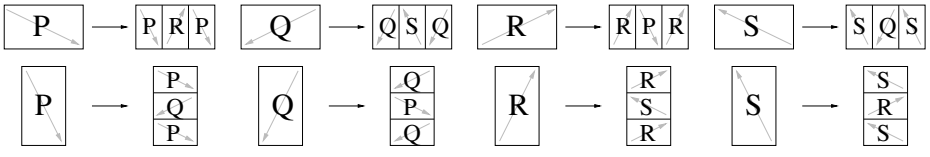


Fig. 3. Splitting and numbering schemes of matrix blocks. The arrows indicate the positions of the first and last element, respectively.

3 Block Recursive Multiplication

Block-recursive multiplication of matrices is a quite well-known approach to achieve cache-efficient algorithms [2, 3, 6]. In our case, the matrix blocks for multiplication naturally derive from the block numbering scheme. However, there are three characteristic matrix dimensions involved, where the recursive splitting could be applied. For example, the multiplication of two P-numbered matrix blocks onto a P-numbered target matrix can lead to either of the following three block multiplications:

$$\begin{pmatrix} P \\ Q \\ P \end{pmatrix} P = \begin{pmatrix} P \\ Q \\ P \end{pmatrix} \longrightarrow \begin{cases} PP = P \\ QP = Q \\ PP = P \end{cases} \tag{2}$$

$$\begin{pmatrix} P\ R\ P \end{pmatrix} \begin{pmatrix} P \\ Q \\ P \end{pmatrix} = P \longrightarrow \begin{cases} PP = P \\ RQ = P \\ PP = P \end{cases} \tag{3}$$

$$P \begin{pmatrix} P\ R\ P \end{pmatrix} = \begin{pmatrix} P\ R\ P \end{pmatrix} \longrightarrow \begin{cases} PP = P \\ PR = R \\ PP = P \end{cases} \tag{4}$$

Consider a block multiplication $AB = C$, and let m be the number of rows of A and C, k be the number of columns of A (and rows of B), and n be the number of columns of B and C. In the block numbering scheme, the matrix

blocks will be split in the largest dimension first. Hence, to make the block multiplication compatible with the block numbering, we again have to divide the largest dimension first:

- if $m \geq n$ and $m \geq k$, we will adopt scheme (2);
- if $n > m$ and $n \geq k$, we will adopt scheme (3);
- if $k > m$ and $k > n$, we will adopt scheme (4).

From figure 1, we know that a multiplication on three P-numbered blocks ($PP = P$) always starts with the elements in the three respective top-left corners of the matrices, and ends with the elements in the respective bottom-right corners. This coincides with the numbering schemes. Thus, for the block multiplication $QP = Q$, the second block multiplication in scheme (2), we have to make sure that

- there exists a multiplication scheme similar to that given in figure 1, and
- this scheme runs from the top-right corners of the Q-blocks towards their bottom-left corners, while
- the P-block is traversed from bottom-right to top-left, i.e. in the opposite direction of its numbering.

Then, the $QP = Q$ multiplication neatly fits between the two $PP = P$ schemes: no index jump will occur even during the transition between the two block multiplications. Thus, the elements of all block matrices will be traversed in a purely local manner: after an element is accessed it will either be directly re-used, or one of its direct neighbours will be used next.

In the schemes (2) to (4) for the $PP = P$ multiplication, three further possible combinations for the multiplication of differently numbered blocks come up: $QP = Q$, $RQ = P$, and $PR = R$. Of course, these can again lead to new possible combinations. A careful examination of all possible scenarios leads to a closed system of only eight possible situations. These are listed in table 1. For every combination, a scheme similar to that given in figure 1 can be derived. In fact, the schemes are identical up to the fact that the execution order can be reversed for one, two, or all three of the matrices. Table 1 lists the execution order for the eight different schemes indicating a '+' for the original execution sequence given in figure 1, and indicating a '−', if the reverse sequence is applied.

Table 1. Possible combinations of block numberings during the block recursive matrix multiplication and their respective execution order

$AB = C$	$PP = P$	$PR = R$	$QP = Q$	$QR = S$	$RQ = P$	$RS = R$	$SQ = Q$	$SS = S$
A	+	−	+	−	+	−	+	−
B	+	+	−	−	+	+	−	−
C	+	+	+	+	−	−	−	−

Finally, the schemes (2) to (4) have to be formulated for all eight combinations given in table 1. For all resulting schemes, we again find that no index jumps occur when leaving one block multiplication and entering the next.

4 Implementation and Properties of the Algorithm

Algorithm 1 shows a prototypical recursive implementation of the $PP = P$ block multiplication of three matrices of size $m \times k$, $k \times n$, and $m \times n$. The variables `idxA`, `idxB`, and `idxC` are considered to be global variables, and contain the (Peano) index of the currently accessed matrix elements. These variables will only be changed by increment and decrement operations. During the multiplication of the atomic blocks, this results from the execution order given in 1. The final algorithm consists of eight nested recursive procedures: one for each of the schemes listed in table 1. The algorithm has the following properties:[1]

Algorithm 1. Block multiplication of three P-numbered blocks

```
PPP_mult(int m, int k, int n) {
   if (m < 9 && k < 9 && n < 9) {
      // multiplication of atomic blocks
   }
   else if ( (m > k) && (m > n) ){
      PPP_mult( subblocksize(m,0), k, n); idxA++; idxC++;
      QPQ_mult( subblocksize(m,1), k, n); idxA++; idxC++;
      PPP_mult( subblocksize(m,2), k, n);
   }
   else if ( (m <= k) && (k > n) ){
      PPP_mult(m, subblocksize(k,0), n); idxA++; idxB++;
      RQP_mult(m, subblocksize(k,1), n); idxA++; idxB++;
      PPP_mult(m, subblocksize(k,2), n);
   }
   else {
      PPPmult(m, k, subblocksize(n,0)); idxB++; idxC++;
      PRRmult(m, k, subblocksize(n,1)); idxB++; idxC++;
      PPPmult(m, k, subblocksize(n,2));
   }
}
```

- The matrix indices are only changed by increment or decrement operators. This ensures that we will usually stay with the current memory block (a cache line, for example).
- For any p, any sequence of p^3 floating point operations will affect a range of only $\mathcal{O}(p^2)$ *contiguous* elements in each matrix. This is obviously the optimal ratio we can achieve in the long range. It does not only guarantee optimal re-use of data, but also makes it possible to predict precisely when neighbouring blocks of memory will be accessed.

With these properties, the number of cache line transfers can, for example, be computed for a so-called *ideal cache* [4]—a cache that will always make the best choice when removing cache lines from the cache. The number of transfers is then of order $\mathcal{O}\left(\frac{N^3}{L\sqrt{M}}\right)$, or more precisely $T(N) \approx 6\sqrt{3}\frac{N^3}{L\sqrt{M}}$, where N is the

size of the square matrices, L is the length of the cache lines, and M is the number of lines in the cache. This is also asymptotically optimal [7].

5 Performance

For performance tests, two simple code optimizations were applied:

- To reduce the number of recursive calls, several levels of recursion can be combined. For example, if a matrix can be divided in 3×3 subblocks, the subsequent three recursive steps lead to a sequence of 27 recursive calls—these can be coded directly.
- To speed up recursive calls, parameters and local variables should be avoided. There are, in fact, no essential parameters that can not be turned into global or static variables.

As we did not attempt any processor specific optimizations so far, it is difficult to provide a fair quantification of our algorithm's performance in comparison to that of standard libraries. In order to suppress effects that result from processor specific optimization, we chose to compare our algorithm with the *default kernel* of Intel's MKL[1] (Math Kernel Library). The *default* kernel does not use SSE instructions, for example, which makes it applicable on any Pentium processors. The code for the Peano multiplication was also compiled without using processor specific instructions. The achieved MFLOPS rates are given in figure 4. The Peano based implementation exceeds the generic MKL by a factor of about 3%. Of course, this can by no means be regarded as a serious comparison of performance. However, it should at least give a rough impression of the real-world performance that can be achieved by the Peano algorithm.

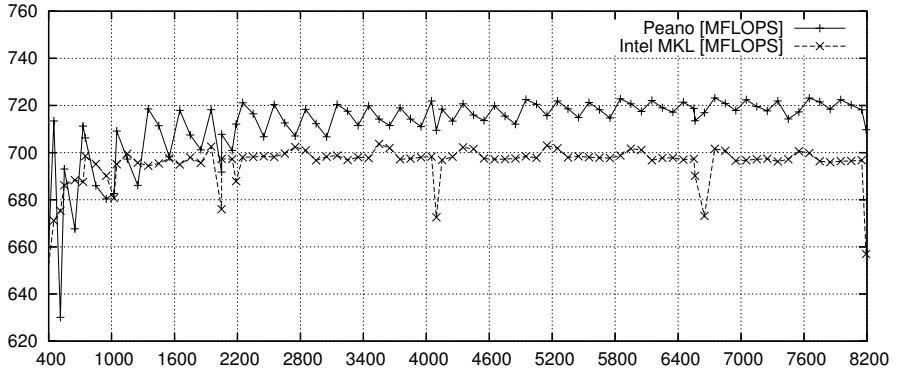


Fig. 4. MFLOPS of the Peano multiplication in comparison to Intel's MKL (generic library, no SSE instructions allowed) on a Pentium 4 processor (3.4 GHz)

[1] Intel Math Kernel Library 7.2, `http://www.intel.com/software/products/mkl/`

6 Conclusion

We have presented a cache oblivious algorithm for the multiplication of full matrices. The algorithm has a completely local access pattern to the stored matrix elements. Together with the block recursive structure this leads to asymptotically optimal performance with respect to re-use of data and number of cache line transfers. Apart from its potential to cache efficient implementations, the algorithm should also be especially qualified for parallel scenarios—both, for shared memory and distributed memory architectures. At least to some extent, the locality properties might also be exploitable for sparse matrices, which together with the parallelization outlines our intended future work in this field.

References

1. M. Bader, C. Zenger. Cache oblivious matrix multiplication using an element ordering based on the Peano curve, *Linear Algebra and its Applications*, submitted
2. S. Chatterjee, V.V. Jain, A.R. Lebeck, S. Mundhra, M. Thottethodi. Nonlinear Array Layouts for Hierarchical Memory Systems, in *International Conference on Supercomputing (ICS'99)*, 1999
3. J. Frens, D.S. Wise. Auto-Blocking Matrix-Multiplication or Tracking BLAS3 Performance from Source Code. In *Proceedings of the 6th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1997.
4. M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 285–297, New York, October 1999.
5. K. Goto, R. van de Geijn. On Reducing TLB Misses in Matrix Multiplication. TOMS, under revision, `http://www.cs.utexas.edu/users/flame/pubs.html`
6. F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development 41 (6)*, 1999
7. J.-W. Hong, H.T. Kung. I/O complexity: the red-blue pebble game. In *Proceedings of ACM Symposium on Theory of Computing*, 1981
8. R. C. Whaley, A. Petitet, J.J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing 27(1–2)*, 2001, pp. 3–35
9. G. Zumbusch. Adaptive Parallel Multilevel Methods for Partial Differential Equations. Habilitation, Universität Bonn, 2001.

On Using an Hybrid MPI-Thread Programming for the Implementation of a Parallel Sparse Direct Solver on a Network of SMP Nodes

Pascal Hénon, Pierre Ramet, and Jean Roman

LaBRI, UMR CNRS 5800 & ENSEIRB,
INRIA Futurs ScAlApplix Project,
351, cours de la Libération, F-33405 Talence, France
{henon, ramet, roman}@labri.fr

Abstract. Since the last decade, most of the supercomputer architectures are based on clusters of SMP nodes. In those architectures the exchanges between processors are made through shared memory when the processors are located on a same SMP node and through the network otherwise. Generally, the MPI implementations provided by the constructor on those machines are adapted to this situation and take advantage of the shared memory to treat messages between processors in a same SMP node. Nevertheless, this transparent approach to exploit shared memory does not avoid the storage of the extra-structures needed to manage efficiently the communications between processors. For high performance parallel direct solvers, the storage of these extra-structures can become a bottleneck. In this paper, we propose an hybrid MPI-thread implementation of a parallel direct solver and analyse the benefits of this approach in terms of memory and run-time performances.

1 Introduction and Background

Solving large sparse symmetric positive definite systems $Ax = b$ of linear equations is a crucial and time-consuming step, arising in many scientific and engineering applications. The authors presented in previous works [2, 3, 4] an efficient static scheduling based on a mixed 1D/2D block distribution for a parallel supernodal version of sparse Cholesky factorization with total *local aggregation on processors*. Parallel experiments, using MPI communications, were run on IBM SP machines at CINES (Montpellier, France) on a large collection of sparse matrices containing industrial 3D problems that reach 1 million of unknowns, and have shown that our PaStiX software compares very favorably to PSPASES [6].

Nevertheless, a major memory bottleneck for our parallel supernodal factorization scheme is caused by this local aggregation mechanism. The local aggregation mechanism is due to the fact that during the factorization of some local column-blocks, a processor has to update several times a block A_{ij} mapped on another processor. Indeed, it would be costly to send each contribution for a same non-local block in separated messages. As illustrated on figure 1, the so-called "local aggregation" variant of the supernodal parallel factorization consists in

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1050–1057, 2006.

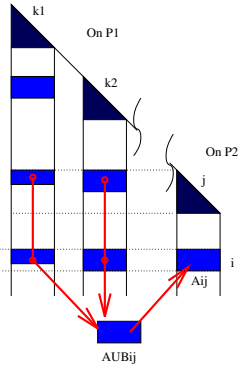


Fig. 1. Local aggregation of block updates. Column-block k_1 and k_2 are mapped on processor P_1, column-block j is mapped on processor P_2. Contributions from the processor P_1 to the block A_{ij} of processor P_2 are locally summed in AUB_{ij}.

adding locally any contribution for a non-local block A_{ij} in a local temporary block noted AUB_{ij} (*aggregated update block*) and sending it once all the contributions destinated to the non-local block A_{ij} have been added.

For very large matrices from 3D problems, the highest peak of memory consumption due to the storage of aggregated block contributions during the factorization amounts several time the local factorized matrix part on a processor. Thus in the *full MPI* version of our parallel solver, we introduced an optimized communication mechanism to control the overhead of memory while loosing acceptable run-time performance [4].

Until now, we have discussed the parallelization in a full distributed memory environment. Each processor was assumed to manage a part of the matrix in its own memory space, and any communication needed to update non local blocks of the matrix was considered under the message passing paradigm. Nowaday, the massively parallel high performance computers are generally designed as networks of SMP nodes. Each SMP nodes consists in a set of processors that share the same physical memory. In [5], we have studied the improvement of our static scheduling to take into account the communication modeling of multilevel memories on network of SMP nodes. However, on those SMP-nodes architectures, to fully exploit shared memory advantages, a relevant approach is to use an hybrid MPI-thread implementation.

In the framework of direct solver, this approach aims at solving 3D problems with more than 10 millions of unkowns, which is now a reachable challenge with these new SMP supercomputers. The rational that motived this hybrid implementation was that the communications within a SMP node can be advantageously substituted by direct accesses to shared memory between the processors in the SMP node using threads. As a consequence, the MPI communication are only used between processors that host threads from different MPI processes.

This kind of approach has been used in WSMP [1] which is a the parallel multifrontal solver. The main difference between WSMP and our work is on

the way to exploit parallelism inside the SMP nodes. In WSMP approach, the parallelism inside SMP nodes is automatically exploited using multi-threaded BLAS. In our approach, we use our static regulation algorithms to decide exactly how the factorization of a supernode has to be splitted in elementary BLAS tasks between threads.

The remainder of the paper is organized as follows: the section 2 describes the main features of our method. We provide some experiments in section 3. At last, we give some conclusions in section 4.

2 Overview of Our Hybrid MPI-Thread Implementation

As said in the previous section, the local aggregation mechanism is inherent to the message passing model used in our parallel factorization algorithm. Thus, in an SMP context, the simplest way to lower the use of aggregate update blocks is to avoid the message passing communications between processors on a same SMP node. In order to avoid these intra-node communications, we use threads that share the same memory space. In an SMP node, a unique MPI process spawns a thread on each processor. These threads are then able to address the whole memory space of the SMP node. Each thread is therefore able to access the matrix part mapped on its MPI process. By this way, though the computational tasks

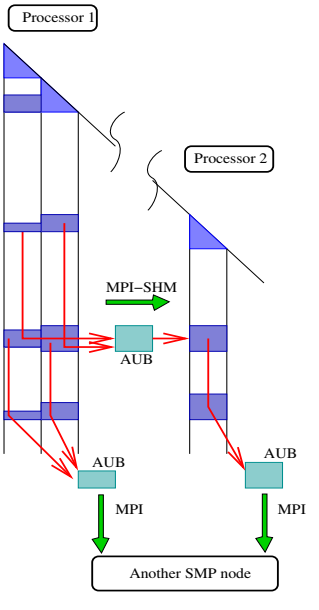


Fig. 2. Processor 1 and 2 belong to the same SMP node. Data exchanges when only MPI processes are used in the parallelization.

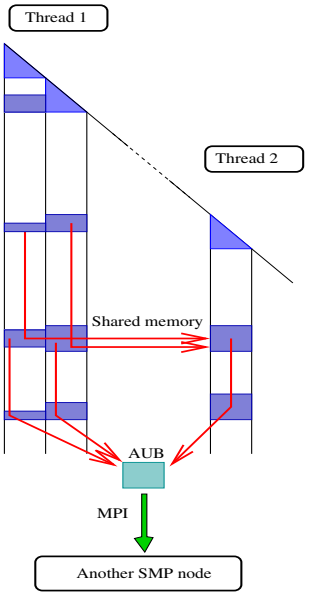


Fig. 3. Thread 1 and 2 are created by one MPI process. Data exchanges when there is one MPI process per SMP node and one thread per processor.

are distributed between the threads, any update of the matrix part of the MPI process is directly performed in the local matrix. The communication between MPI processes still uses the local aggregation scheme we described earlier.

This hybrid MPI-thread strategy is pictured on figures 2 and 3. The figure 2 sketches the situation when each processor is assigned a MPI process. In this case, all the communications are performed using MPI. The extra-memory needed to store the aggregate update blocks (AUB) is maximal since each MPI process communicates their updates to any other MPI process through AUBs. The figure 3 shows the situation when only one MPI process is used per SMP node and inside each SMP node a thread is assigned on each processor. In this case, all the threads on a SMP node share the same memory space and update directly any column block assigned to their MPI process; in addition, the local aggregation is made globally for the whole set of column blocks mapped on the MPI process. This implies that the amount of extra-memory required for the storage of the AUBs depends only on the number of MPI processes. Therefore, the additional memory needed to store the AUBs is all the more reduced that the SMP node are wider (in terms of number of processors).

3 Experiments

All of the algorithms described in this paper have been integrated in the PaStiX software [2, 3], that make use of the static mapping and sparse matrix ordering software package Scotch [8] version 3.4.

The parallel experiments were run on an 28 NH2 nodes [IBM SP3] (16 Power3+ with 16Go per node) located at CINES (Montpellier, France) with a network based on a Colony switch. We have also used 2 SMP nodes [IBM SP4] (32-ways Power4+ with 64Go per node) with a network based on a Federation switch to validate our approach on our largest test cases. Switch interrupts are enabled with default delay to perform non-blocking communication efficiently. All computations are performed in double precision (the relative backward error observed on our problems is arround 10^{-15} for our direct factorization) and all time results are given in seconds. The blocking size parameter for BLAS3 computations is set to 60 and we use here a one dimensional distribution as default. We have apply an LDL^t factorization on symmetric matrices and an LU factorization for unsymmetric matrices (but with a symmetric pattern). In all the following tables, the symbol "-" is used when the time measurments are not significant due to memory swapping.

Our experiments were performed on a collection of sparse matrices from the PARASOL ESPRIT Project and from CEA. The values of the associated measurments in Table 1 come from scalar column symbolic factorization.

On each MPI process p, we compute the size of the local allocation for matrix coefficients (denoted coeff_alloc(p)) and of the local allocation for the AUB (denoted AUB_alloc(p)). The memory efficiency $mem_{\text{eff}}(P)$, on P processes, is defined as follow:

$$mem_{\text{eff}}(P) = \frac{\sum_p \{\text{coeff\_alloc}(p)\}}{P \cdot \max_p \{\text{coeff\_alloc}(p) + \text{AUB\_alloc}(p)\}}$$

The memory efficiency measures how the total memory allocations (matrix coefficients and AUBs) is reduced when the number of processors increases. When $mem_{\text{eff}}(P) = 1$, it means that there is no memory overhead due to the parallelization; this can only happen when a single MPI process is used.

In table 2, for the two largest symmetric problems, the number of processors vary between 16 (1 NH2 node) and 128 (4 NH2 nodes) of the IBM SP3. We analyze the factorization time and the memory efficiency when we use 1, 4, 8 and 16 threads per MPI process. As expected, we can see that using 1 thread for each CPU is always profitable for memory efficiency. For instance, with the AUDI matrix, on 128 processors, the memory efficiency increases from 0.23 to 0.70. Beside, we can notice that the best factorization time is almost always obtain with 8 threads by MPI process. Figure 4 shows graphically that we keep a good memory efficiency and a good time scalability when we increase the number of threads by MPI process.

Table 1. Description of our test problems. NNZ_A is the number of off-diagonal terms in the triangular part of matrix A, NNZ_L is the number of off-diagonal terms in the factorized matrix L and OPC is the number of operations required for the factorization. Matrices are sorted in decreasing order of $\frac{\text{NNZ}_\text{L}}{\text{OPC}}$ which is a measure of the potential data reuse [7].

Name	Columns	NNZ_A	NNZ_L	OPC	$\frac{\text{NNZ}_\text{L}}{\text{OPC}}$	Description
OILPAN	73752	1761718	8.912337e+06	2.984944e+09	2.98e-3	symetric
QUER	59122	1403689	9.118592e+06	3.280680e+09	2.78e-3	symetric
SHIP001	34920	2304655	1.427916e+07	9.033767e+09	1.58e-3	symetric
X104	108384	5029620	2.634047e+07	1.712902e+10	1.54e-3	symetric
MT1	97578	4827996	3.114873e+07	2.109265e+10	1.48e-3	symetric
INLINE	503712	18660027	1.762790e+08	1.358921e+11	1.29e-3	symetric
BMWCRA1	148770	5247616	6.597301e+07	5.701988e+10	1.16e-3	symetric
CRANKSG1	52804	5280703	3.142730e+07	3.007141e+10	1.05e-3	symetric
SHIPSEC8	114919	3269240	3.572761e+07	3.684269e+10	0.97e-3	symetric
CRANKSG2	63838	7042510	4.190437e+07	4.602878e+10	0.91e-3	symetric
SHIPSEC5	179860	4966618	5.649801e+07	6.952086e+10	0.81e-3	symetric
SHIP003	121728	3982153	5.872912e+07	8.008089e+10	0.73e-3	symetric
THREAD	29736	2220156	2.404333e+07	3.884020e+10	0.62e-3	symetric
AUDI	943695	39297771	1.214519e+09	5.376212e+12	2.26e-4	symetric
MHD1	485597	24233141	1.629822e+09	1.671053e+13	9.75e-5	unsymetric

Table 2. Factorization time (and memory efficiency) on SP3

Name	Number of processors			
	16	32	64	128
INLINE 1 thread/process	**13.39** (0.77)	7.99 (0.70)	6.07 (0.59)	4.58 (0.42)
INLINE 4 threads/process	14.23 (0.94)	7.08 (0.89)	4.28 (0.84)	3.33 (0.78)
INLINE 8 threads/process	13.46 (0.98)	**6.68** (0.94)	**4.12** (0.89)	**3.07** (0.81)
INLINE 16 threads/process	13.89 (**1.00**)	8.41 (**0.98**)	4.59 (**0.94**)	3.51 (**0.90**)
AUDI 1 thread/process	-	-	211.35 (0.34)	134.45 (0.23)
AUDI 4 threads/process	**472.67** (0.81)	266.89 (0.71)	155.23 (0.60)	87.56 (0.43)
AUDI 8 threads/process	476.40 (0.93)	**265.09** (0.81)	**145.19** (0.70)	**81.90** (0.54)
AUDI 16 threads/process	481.96 (**1.00**)	270.16 (**0.89**)	152.58 (**0.83**)	86.07 (**0.70**)

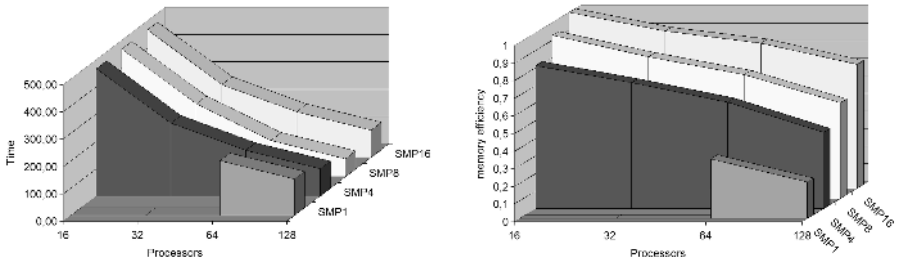


Fig. 4. Factorization time and memory efficiency on IBM SP3 for AUDI problem

Table 3. Factorization time (and memory efficiency) on SP4

Name	Number of processors	
	32	64
AUDI 4 threads/process	94.21 (0.71)	60.03 (0.57)
AUDI 8 threads/process	**93.14** (0.82)	47.80 (0.74)
AUDI 16 threads/process	96.31 (0.92)	**47.28** (0.81)
AUDI 32 threads/process	100.40 (**1.00**)	51.02 (**0.92**)
MHD1 4 threads/process	199.17 (0.29)	115.97 (0.16)
MHD1 8 threads/process	**187.89** (0.49)	**111.99** (0.31)
MHD1 16 threads/process	197.99 (0.73)	115.79 (0.49)
MHD1 32 threads/process	202.68 (**1.00**)	117.80 (**0.78**)

In table 3, we present experiments on IBM SP4. For 1 node (32 processors) then 2 nodes (64 processors), we increase the number of threads by MPI process between 4 to 32. Here again, the memory reduction is really substantial. We also notice that the best factorization time is obtain when using 8 threads by MPI process. This can be explained by the fact that this kind of node presents 4 I/O cards for communications on the Federation switch. This behaviour is verified for both symmetric problem (AUDI test case) and unsymmetric problem (MHD1 test case).

Another interesting remark is that the MPI-thread implementation allows us to manage more efficiently a two dimensional distribution mapping. It is well known that this kind of distribution is payfull in terms of scalability compared to a one dimensional distribution for large matrices arising from 3D problems. Indeed, a two dimensional distribution generates a lot of small messages that are advantageously substituted by direct accesses to shared memory.

In table 4, we provide some results on a 3D magneto-hydrodynamic problem (MHD1 test case) when we set a two dimensional distribution on the 4 first levels of the block elimination tree. A complete analysis is out of the scope of this paper, but this result illustrates another benefit of using our MPI-thread implementation.

Finally, we give some comparisons (see tables 5 and 6) on the IBM SP3 for the whole set of test cases in terms of factorization time and memory efficiency for both MPI and MPI-thread implementations. While the memory reduction is the most crutial criteria to optimize for parallel direct solver implementations,

Table 4. MHD1 with 1D or 2D distribution on IBM SP4 (32 processors)

Name	Implementation	
	Full MPI	MPI-thread
MHD1 with 1D distribution	115.97	117.80
MHD1 with 2D distribution	111.51	**90.68**

Table 5. Factorization time (and memory efficiency) on SP3 (MPI only)

Name	Number of processors			
	16	32	64	128
OILPAN	**.61** (0.74)	**.38** (0.71)	**.37** (0.62)	**.37** (0.58)
QUER	**.90** (0.69)	**.45** (0.69)	**.45** (0.55)	**.47** (0.55)
SHIP001	**1.39** (0.67)	**.78** (0.59)	**.75** (0.48)	**.73** (0.46)
X104	**2.76** (0.58)	**1.86** (0.49)	**1.79** (0.39)	**1.12** (0.28)
MT1	**3.52** (0.62)	**1.55** (0.51)	**1.41** (0.41)	**1.53** (0.33)
INLINE	**13.39** (0.77)	**7.99** (0.70)	**6.07** (0.59)	**4.58** (0.42)
BMWCRA1	**6.32** (0.72)	**3.39** (0.59)	**2.49** (0.43)	**2.18** (0.37)
CRANKSEG1	**4.19** (0.58)	**2.04** (0.47)	**1.58** (0.40)	**1.92** (0.32)
SHIPSEC8	**6.07** (0.45)	**3.87** (0.34)	**3.70** (0.26)	**3.71** (0.23)
CRANKSEG2	**5.19** (0.54)	**2.94** (0.43)	**2.24** (0.40)	**2.64** (0.28)
SHIPSEC5	**9.42** (0.46)	**5.52** (0.38)	**4.67** (0.27)	**4.78** (0.21)
SHIP003	**9.50** (0.55)	**5.84** (0.47)	**4.77** (0.30)	**4.84** (0.23)
THREAD	**6.48** (0.39)	**3.65** (0.30)	**3.90** (0.24)	**3.90** (0.20)
AUDI	-	-	**211.35** (0.34)	**134.45** (0.23)

Table 6. Factorization time (and memory efficiency) on SP3 (MPI-thread)

Name	Number of processors			
	16	32	64	128
OILPAN	**.95** (1.00)	**.60** (0.96)	**.36** (0.91)	**.31** (0.84)
QUER	**.81** (1.00)	**.54** (0.96)	**.46** (0.87)	**.34** (0.86)
SHIP001	**1.02** (1.00)	**.64** (0.94)	**.51** (0.91)	**.51** (0.80)
X104	**2.20** (1.00)	**1.53** (0.95)	**1.11** (0.86)	**.91** (0.77)
MT1	**2.10** (1.00)	**1.28** (0.94)	**.91** (0.89)	**.82** (0.84)
INLINE	**13.89** (1.00)	**8.41** (0.98)	**4.59** (0.94)	**3.51** (0.90)
BMWCRA1	**5.96** (1.00)	**3.25** (0.96)	**2.05** (0.86)	**1.28** (0.83)
CRANKSEG1	**2.87** (1.00)	**1.67** (0.96)	**1.26** (0.88)	**.98** (0.78)
SHIPSEC8	**4.52** (1.00)	**3.09** (0.85)	**2.46** (0.82)	**2.33** (0.76)
CRANKSEG2	**4.15** (1.00)	**2.46** (0.96)	**1.51** (0.83)	**1.45** (0.71)
SHIPSEC5	**7.44** (1.00)	**4.68** (0.87)	**3.40** (0.78)	**3.05** (0.68)
SHIP003	**7.71** (1.00)	**5.22** (0.91)	**3.29** (0.82)	**3.13** (0.70)
THREAD	**3.91** (1.00)	**2.72** (0.81)	**2.53** (0.65)	**2.54** (0.59)
AUDI	**481.96** (1.00)	**270.16** (0.89)	**152.58** (0.83)	**86.07** (0.70)

we focus our study on setting 1 thread for each CPU unit (that is to say 16 threads by MPI process).

These results are in agreement with previous remarks and are also verified on IBM SP4. But, on this architecture, the factorization times obtained are less significant because for those problem sizes, the measured times are often lower than 1 second. As a conclusion for these experiments, we obtain a good memory scalability and the factorization time is most often reduced whatever the number of processors, thanks to the MPI-thread implementation. Up to 64 processors, the factorization time is always reduced what shows that the MPI-thread implementation also improves the time scalability.

4 Concluding Remarks

In this paper, we have proposed an hybrid MPI-thread implementation of a direct solver and have analyzed the benefits of this approach in terms of memory and run-time performances.

To validate this approach, we have performed experiments on IBM SP3 and SP4. In all test cases, the memory overhead is drastically reduced thanks to the hybrid MPI-thread implementation. In addition, we observe a significant decreasing of the factorization time and a better global scalability of the parallel solver. This techniques allow us to treat large 3D problems where the memory overhead was a bottleneck for the use of direct solvers. Recently, we have factorized a 3D problem from CEA with 10 millions of unknowns (NNZ=6.68e+9, OPC=4.29e+13) in less than 400 seconds on 2 SMP nodes (32-ways Power4+ with 64Go per node); for this case, the memory efficiency is about 0.95. We have only considered in this paper the factorization step, which is the most time consuming step, but the same improvements are also verified for the triangular solve step.

Finally, a comparison with WSMP [1] is still in progress and will be published in a survey article on the PaStiX direct solver.

References

1. Anshul Gupta, Mahesh Joshi, and V. Kumar. Wsmp: A high-performance shared- and distributed-memory parallel sparse linear equation solver. Report, University of Minnesota and IBM Thomas J. Watson Research Center, 2001.
2. P. Hénon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *Proceedings of EuroPAR'99*, number 1685 in Lecture Notes in Computer Science, pages 1059–1067. Springer Verlag, September 1999.
3. P. Hénon, P. Ramet, and J. Roman. PaStiX: A Parallel Sparse Direct Solver Based on a Static Scheduling for Mixed 1D/2D Block Distributions. In *Proceedings of Irregular'2000*, number 1800 in Lecture Notes in Computer Science, pages 519–525. Springer Verlag, May 2000.
4. P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, January 2002.
5. P. Hénon, P. Ramet, and J. Roman. Efficient algorithms for direct resolution of large sparse system on clusters of SMP nodes. In *SIAM Conference on Applied Linear Algebra, Williamsburg, Virginie, USA*, July 2003.
6. M. Joshi, G. Karypis, V. Kumar, A. Gupta, and Gustavson F. PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems. Technical report, University of Minnesota and IBM Thomas J. Watson Research Center, May 1999.
7. X. S. Li. *Sparse Gaussian Elimination on High Performance Computers*. PhD thesis, University of California at Berkeley, 1996.
8. F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12:69–84, 2000. Preliminary version published in *Proceedings of Irregular'99*, LNCS 1586, 986–995.

Adapting Linear Algebra Codes to the Memory Hierarchy Using a Hypermatrix Scheme*

José R. Herrero and Juan J. Navarro

Computer Architecture Dept., Univ. Politècnica de Catalunya,
Barcelona, Spain
{josepr, juanjo}@ac.upc.edu

Abstract. We present the way in which we adapt data and computations to the underlying memory hierarchy by means of a hierarchical data structure known as hypermatrix. The application of orthogonal block forms produced the best performance for the platforms used.

1 Introduction

In order to obtain efficient codes, computer resources have to be used effectively. The code must have an inner kernel able to make use of the functional units within the processor in an efficient way. On the other hand, data must be accessible in a very short time. This can be accomplished with an adequate usage of the memory hierarchy. In this paper we present the way in which we have obtained fast implementations of two important linear algebra operations: dense Cholesky factorization and matrix multiplication.

We are interested in the development of efficient linear algebra codes on a variety of platforms. For this purpose, we use a hierarchical data structure known as *Hypermatrix* to adapt our code to the target machine.

1.1 Hypermatrix Data Structure

Our application uses a data structure based on a hypermatrix (HM) scheme [1, 2], in which a matrix is partitioned recursively into blocks of different sizes. A commercial package known as PERMAS uses the hypermatrix structure for solving very large systems of equations [3]. It can solve very large systems out-of-core and can work in parallel. This approach is also related to a variety of recursive/nonlinear data layouts which have been explored elsewhere for both regular [4, 5, 6, 7] and irregular [8] applications.

The HM structure consists of N levels of submatrices. In order to have a simple HM data structure which is easy to traverse we have chosen to have blocks at each level which are multiples of the lower levels. The top *N-1* levels hold pointer matrices which point to the next lower level submatrices. Null pointers in pointer matrices indicate that the corresponding submatrix does not

* This work was supported by the Ministerio de Ciencia y Tecnología of Spain (TIN2004-07739-C02-01).

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1058–1065, 2006.

have any non-zero elements and is therefore unnecessary. This is useful when matrices are sparse. Only the last (bottom) level holds data matrices. Data matrices are stored as dense matrices and operated on as such. Hypermatrices can be seen as a generalization of quadtrees. The latter partition each matrix precisely into four submatrices [9].

1.2 Motivation

In the past, we have been working on the sparse Cholesky factorization based on the hypermatrix data structure. In the sparse codes it was important to avoid unnecessary operation on zeros. At the same time, using small data submatrices produced many calls to matrix multiplication subroutines resulting in a large overhead. For this reason we created efficient routines which operate on small matrices. By small we mean matrices which fit in cache. We grouped such routines in a library called the Small Matrix Library (SML). Information about the creation of the SML can be found in [10]. Further details on the application of SML to sparse hypermatrix Cholesky can be found in [11].

The hypermatrix data structure, however, can also be used for dense matrix computations. Recently, we have applied a similar approach to work on dense matrices. Let us comment on the case of the MIPS R10000 processor. Figure 1 shows the peak performance of the $C = C - A \times B^t$ matrix multiplication routines in our SML for several matrix dimensions on a MIPS R10000 processor. On small matrices, our code (*mxmts_fix*) outperforms the DGEMM matrix multiplication routine in the vendor BLAS library. We have labeled the latter as *dgemm_nts* to note that B is used transposed while A is not. In addition, it denotes that the result of the multiplication is subtracted from the previous value in matrix C.

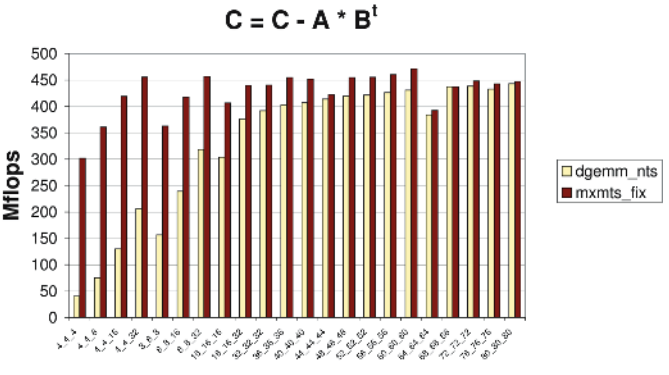


Fig. 1. Peak performance of SML dense matrix multiplication routines on an R10000

We chose the code corresponding to matrices of size 60×60 as our inner kernel since this was the one providing best peak performance. Then, we used this kernel in two codes based on a hypermatrix data structure: a Cholesky factorization and a matrix multiplication. A data submatrix block size of 60×60 allows for a matrix to be almost permanently in L1 cache. Some conflicts might

arise but they should be scarce. A second higher level of 8×8 pointers maps blocks of 480×480 data. This is adequate both for the TLB and second level cache. For the matrix sizes tested those two levels were sufficient to achieve good memory usage. For larger matrices, which required to go out-of-core, or in machines with more levels of caches more pointer levels could be used. The graph on the left part of figure 2 shows the performance obtained on this platform for a dense Cholesky factorization. The graph compares the results obtained by our code (labeled as HM) with those obtained by routine DPOTRF in the vendor library. Both when upper (U) or lower (L) matrices were input to this routine its performance was worse than that of our code. We also tried the matrix multiplication operation $C = C - A * B^T$ since this is the one which takes about 90% of Cholesky factorization. The results can be seen in the right part of figure 2. Our code outperformed the DGEMM matrix multiplication routine in both the vendor and ATLAS libraries. We must note however, that ATLAS was not able to finish its installation process on this platform. Thus, we used a precompiled version of this library which corresponds to an old release of ATLAS. These preliminary results encouraged us to work on dense algorithms based on the hypermatrix data structure.

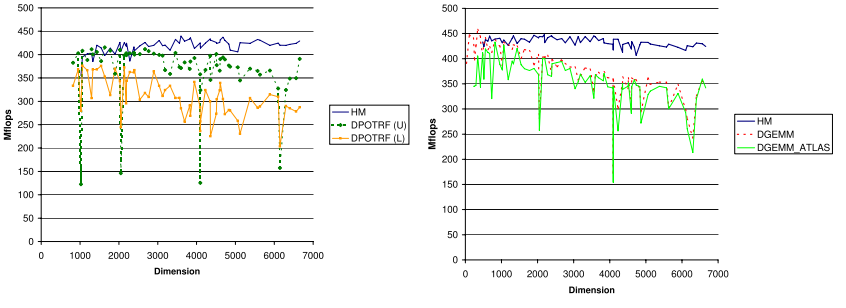


Fig. 2. Performance of dense matrix multiplication and Cholesky factorization on a MIPS R10000 processor

In this paper we present the extension of our work to dense matrix operations. We will explain how our code is adapted to the underlying memory architecture and is able to obtain good performance.

2 Producing Efficient Inner Kernels

In [10] we introduced a Small Matrix Library (SML). The routines in this library operate efficiently on very small matrices (which fit in level 1 cache). Basically, we try several variants of code with different loop orders and unroll factors. By fixing parameters such as matrix leading dimensions and loop trip counts at compilation time we are able to produce very efficient routines on many platforms. We used such routines to improve our sparse Cholesky factorization code based on the *Hypermatrix* data structure. We have extended our SML

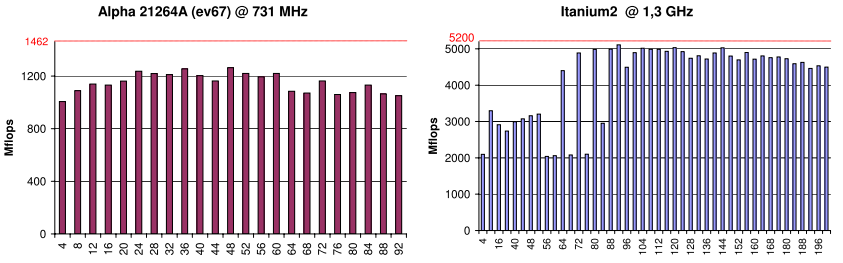


Fig. 3. Peak performance of SML dense matrix multiplication routines

with routines which work with sizes larger than the ones used for the sparse codes. We choose as inner kernel the one providing best performance. On MIPS, ALPHA and Itanium2 platforms we could obtain efficient kernels for the matrix multiplication.

Figure 3 shows the peak performance of the $C = C - A \times B^t$ matrix multiplication routines in our SML for several matrix dimensions on two different processors. On an Alpha 21264 we chose the routine which works on matrices of size 48×48. On the Intel Itanium2 we chose size 92×92. It is interesting to note that on the Itanium2 the highest performance was obtained for matrix sizes which exceed the capacity of the level 1 (L1) data cache. The following tables show information about caches for each of the three platforms used. Table 1 shows the number of caches and their sizes. Table 2 shows the minimum latency for a floating-point load when it hits in each cache level.

Table 1. Cache sizes

Cache Level	R10000	ALPHA 21264	Itanium2
L1	32 KB	64 KB	16 KB
L2	4 MB	4 MB	256 KB
L3	-	-	3 MB

Table 2. Floating-point load latency (minimum) when load hits in cache

Cache Level	R10000	ALPHA 21264	Itanium2
L1	3	4	-
L2	8-10	13	**6**
L3	-	-	13

The Itanium2 has three levels of cache. In the first level it has separate instruction and data caches with 16 Kbytes each. Then, it also has a 256 Kbytes L2 cache and an off-chip L3 cache with possible sizes ranging from 1.5 up to 9 MB. The configuration used had a 3 MB L3 cache. The most interesting point to note is the fact that this machine has low latency when a load hits in its level 2 (L2) cache [12]. The Intel Fortran compiler applied the software pipelining technique automatically for tolerating such latency and produced efficient codes.

This is the reason why on this machine the best peak performance for our SML matrix multiplication routines was found for matrices which exceed the L1 data cache size.

3 Exploiting the Memory Hierarchy

3.1 Number of Levels and Dimension of Each Level

We use the hypermatrix data structure to adapt our codes to the underlying memory hierarchy. Our code can be parameterized with the number of pointer levels and block sizes mapped by each level. For the dense codes we follow the next approach: we choose the data submatrix block size according to the results obtained while creating our SML's matrix multiplication routine. The one providing the best performance is taken. As seen in section 2 we do this even when the matrix size is too large to fit in the L1 cache. Then, for the upper levels we choose multiples of the lower levels close to the value $\sqrt{C/2}$, where C is the cache size [13]. We found that, for the machines studied, we only needed two levels of pointers for dense in-core operations.

Figure 4 shows the performance of our hypermatrix multiplication routine on an Itanium2 processor for several matrix dimensions. We have tried several dimensions for the second level of pointers. Values 368 and 460, close to the value $\sqrt{C/2}$, were the best. We tried using a third level with size 736 when the second one has size 368. It didn't produce any benefit since the SML routine was already using efficiently the L2 cache, and the second level of pointers was enough to use the L3 cache adequately. On the Alpha 21264 the size used for data submatrices was 48×48. Then, a second level of pointers in the hypermatrix maps blocks of dimension 480.

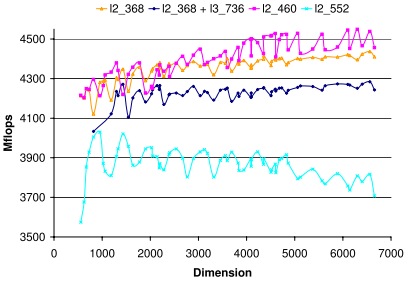


Fig. 4. Performance of dense matrix multiplication and Cholesky factorization on an Intel Itanium 2 processor

3.2 Orthogonal Blocks

In [14] a class of Multilevel Orthogonal Block forms was presented. In that class each level is orthogonal to the previous: they are constructed so that the directions of the blocks of adjacent levels are different. These algorithms exploit

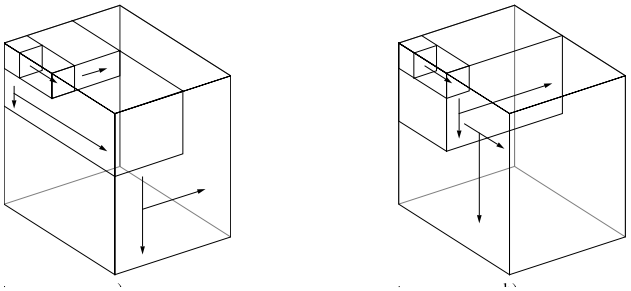


Fig. 5. Two examples of Multilevel Orthogonal Block forms

the data locality in linear algebra operations when executed in machines with several levels in the memory hierarchy. Figure 5 shows graphically the directions followed by two possible Multilevel Orthogonal Block (MOB) forms.

We have implemented Multilevel Orthogonal Blocks for the different levels in the hypermatrix structure. Figure 6 shows the performance obtained on a matrix multiplication performed on matrices of size 4507 on Itanium2 (left) and Alpha 21264 (right) processors for all combinations of loop orders for two level of pointers in a hypermatrix. All bars to the right of the dashed line correspond to orthogonal forms. Although we eventually use only 2 pointer levels, for hypermatrix multiplication there is an improvement in the performance obtained when the upper level is orthogonal to the lower. In this way the upper level cache is properly used. The performance improvement is modest, but results were always better than those corresponding to non-orthogonal block forms.

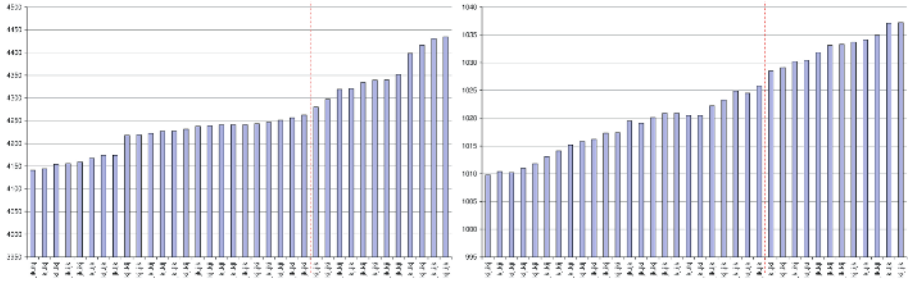


Fig. 6. Performance of HM dense matrix multiplication for several loop orders on Intel Itanium 2 (left) and Alpha 21264 processors (right)

4 Results

We have compared our dense hypermatrix multiplication and Cholesky factorization with ATLAS [15] DGEMM and DPOTRF routines. On the R10000 our code outperformed both the vendor and ATLAS DGEMM and DPOTRF routines. On

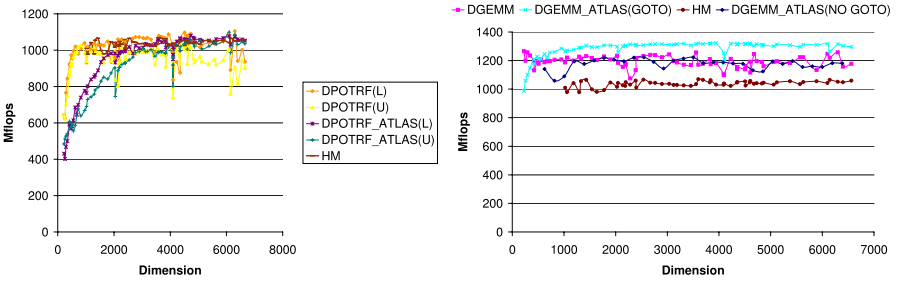


Fig. 7. Performance of dense matrix multiplication and Cholesky factorization on an Alpha 21264 processor

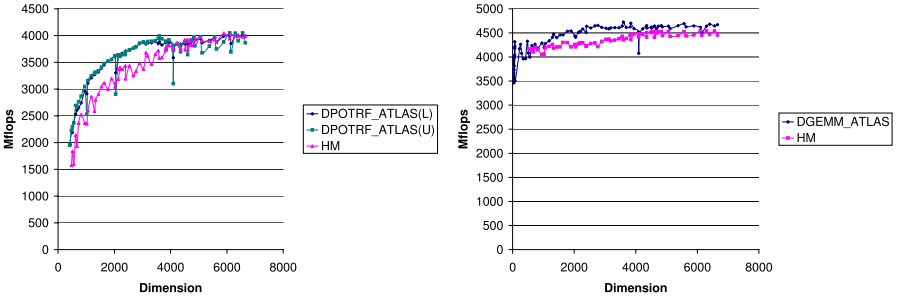


Fig. 8. Performance of dense matrix multiplication and Cholesky factorization on an Intel Itanium 2 processor

an ALPHA-21264A ATLAS' installation phase lets the user choose whether to install a hand made code specially designed for this platform (GOTO). In both cases, on this system, ATLAS outperforms our matrix multiplication code. One reason for this can be observed in figure 3. The peak performance of the matrix multiplication routine in our SML was far from the theoretical peak performance on this machine. However, we obtain the same performance as DPOTRF for large matrices (figure 7). On the Itanium2 our performance got close to ATLAS' both for DGEMM and DPOTRF. It was similar to ATLAS for large matrices (figure 8). We must note that, in spite of its name, the ATLAS project is often based on matrix multiplication kernels written in assembly code by hand.

5 Conclusions

It is possible to obtain high performance codes using a high level language and a good optimizing compiler. The inner kernel can target the first level cache. In some cases, it can also target the second level cache. This happens on processors with small level 1 caches and low latency for floating-point loads from the second level cache.

A hypermatrix data structure can be used to adapt the code to the underlying memory hierarchy. For the machines studied and working on dense matrices in-core, two levels of pointers were enough. Going out-of-core or working on machines with more levels of cache memory could benefit from the extension of this scheme to a larger number of levels in the hypermatrix. The use of Multilevel Orthogonal Block forms was always beneficial.

References

1. Fuchs, G., Roy, J., Schrem, E.: Hypermatrix solution of large sets of symmetric positive-definite linear equations. Comp. Meth. Appl. Mech. Eng. **1** (1972) 197–216
2. Noor, A., Voigt, S.: Hypermatrix scheme for the STAR–100 computer. Comp. & Struct. **5** (1975) 287–296
3. Ast, M., Fischer, R., Manz, H., Schulz, U.: PERMAS: User's reference manual, INTES publication no. 450, rev.d (1997)
4. Chatterjee, S., Jain, V.V., Lebeck, A.R., Mundhra, S., Thottethodi, M.: Nonlinear array layouts for hierarchical memory systems. In: Proceedings of the 13th international conference on Supercomputing, ACM Press (1999) 444–453
5. Frens, J.D., Wise, D.S.: Auto-blocking matrix multiplication, or tracking BLAS3 performance from source code. Proc. 6th ACM SIGPLAN Symp. on Principles and Practice of Parallel Program., SIGPLAN Not. **32** (1997) 206–216
6. Valsalam, V., Skjellum, A.: A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels. Concurrency and Computation: Practice and Experience **14** (2002) 805–839
7. Wise, D.S.: Ahnentafel indexing into Morton-ordered arrays, or matrix locality for free. In: Euro-Par 2000,LNCS1900. (2000) 774–783
8. Mellor-Crummey, J., Whalley, D., Kennedy, K.: Improving memory hierarchy performance for irregular applications. In: Proceedings of the 13th international conference on Supercomputing, ACM Press (1999) 425–433
9. Wise, D.S.: Representing matrices as quadtrees for parallel processors. Information Processing Letters **20** (1985) 195–199
10. Herrero, J.R., Navarro, J.J.: Automatic benchmarking and optimization of codes: an experience with numerical kernels. In: Proceedings of the 2003 International Conference on Software Engineering Research and Practice, CSREA Press (2003) 701–706
11. Herrero, J.R., Navarro, J.J.: Improving Performance of Hypermatrix Cholesky Factorization. In: Euro-Par 2003,LNCS2790, Springer-Verlag (2003) 461–469
12. Intel: Intel(R) Itanium(R) 2 processor reference manual for software development and optimization (2004)
13. Lam, M., Rothberg, E., Wolf, M.: The cache performance and optimizations of blocked algorithms. In: Proceedings of ASPLOS'91. (1991) 67–74
14. Navarro, J.J., Juan, A., Lang, T.: MOB forms: A class of Multilevel Block Algorithms for dense linear algebra operations. In: Proceedings of the 8th International Conference on Supercomputing, ACM Press (1994)
15. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: Supercomputing '98, IEEE Computer Society (1998) 211–217

Measuring the Scalability of Heterogeneous Parallel Systems

Alexey Kalinov

Institute for System Programming, Russian Academy of Sciences,
25 B. Kommunisticheskaya str., Moscow 109004, Russia

Abstract. A parallel algorithm cannot be evaluated apart from the architecture it is implemented on. So, we define a *parallel system* as the combination of a parallel algorithm and a parallel architecture. The paper is devoted to the extension of well-known isoefficiency scalability metrics to heterogeneous parallel systems. Based on this extension the scalability of SUMMA (Scalable Universal Matrix Multiplication Algorithm) on parallel architecture with homogeneous communication system supporting simultaneous point-to-point communications is evaluated. Two strategies of data distribution are considered: (i) homogeneous – data are distributed between processors evenly; (ii) data are distributed between processors according to their performance. It is shown that under some assumption both strategies ensure the same scalability of heterogeneous parallel system. This theoretical result is corroborated with experiment.

1 Introduction

A parallel algorithm cannot be evaluated apart from the architecture it is implemented on. So, following [1] we define a *parallel system* as the combination of a parallel algorithm and a parallel architecture. There are two important performance metrics of parallel systems: efficiency and scalability. A lot of research was conducted to improve the efficiency of heterogeneous parallel systems. But scalability of such systems was practically out of research scope.

The efficiency of homogeneous parallel system is introduced in the following way. The time taken by an algorithm to execute on a single processor is its sequential execution time T_s. The time taken by an algorithm to execute on p processors is its parallel execution times T_p. A parallel system's speedup S is the ratio of sequential execution time to parallel execution time: $S = T_s/T_p$. Its parallel efficiency E is the ratio of speedup to the number of processors used: $E = S/p$. The parallel efficiency of a parallel system usually increases as the problem size increases and decreases as the number of processors increases.

The scalability is an ability of a parallel system to increase speedup as the number of processors increases. There is no generally accepted metrics that measure scalability. In this paper we follow the most generally accepted metrics proposed in [1]. The metrics is based on the isoefficiency function that dictates how the problem size n must grow to maintain a fixed parallel efficiency as the number

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1066–1073, 2006.

of processors p increases. We extend the metrics to heterogeneous parallel systems case and apply it to scalability analysis of Scalable Universal Matrix Multiplication Algorithm (SUMMA) [2] on parallel architecture with homogeneous communication system supporting simultaneous point-to-point communications.

The paper has a number of contributions. The first one is the extension of isoefficiency metrics to heterogeneous parallel systems. The second is the analysis of influence of homogeneous and heterogeneous data distribution strategies to scalability of SUMMA. The third is the experimental evaluation of the scalability of a large heterogeneous parallel system.

The rest of the paper is organized as follows. Section 2 is devoted to the extension of isoefficiency scalability metrics to heterogeneous case. In section 3 we analyze scalability of the matrix-matrix multiplication algorithm SUMMA on heterogeneous parallel systems. Section 4 presents experimental results. In section 5 we briefly describe related work. Section 6 concludes the paper.

2 Extension of Isoefficiency to Heterogeneous Parallel Systems

Let the performance of each processor to be characterized by positive real number r. In this case in addition to number of processors p a parallel system is characterized by the set of processor performances $R = \{r_i\}, i \in [1, p]$. In this paper we consider the simplest case when r does not depend on problem size. We consider case of one process running per processor and do not differentiate processors from processes.

In [3] is proposed to define the parallel efficiency of heterogeneous parallel system as the ratio of the ideal execution time of parallel computation and the real one. For the problem size n, the number of processors p and the performance set R it can be written as:

$$E(n, p, R) = \frac{T_{ideal}(n, p, R)}{T_p(n, p, R)} \tag{1}$$

The ideal time can be computed in the following way. Let the sequential execution time $T_s(n, r_{seq})$ is in inverse proportion to performance r_{seq} of processor, which perform computation. In ideal we can consider a parallel system as one sequential computer with performance equal to the sum of performances of the processors constituting the parallel system $r_{sum} = \sum_{i \in [1,p]} r_i$ and write ideal time as follows:

$$T_{ideal}(n, p, R) = \frac{T_s(n, r_{seq}) r_{seq}}{r_{sum}} = \frac{r_{seq}}{r_{aver}} \cdot \frac{T_s(n, r_{seq})}{p},$$

where r_{aver}- average performance of processors of the parallel system. Then, equation (1) can be rewritten as

$$E(n, p, R) = \frac{r_{seq}}{r_{aver}} \cdot \frac{T_s(n, r_{seq})}{p T_p(n, p, R)}.$$

Let performance of the processor performing sequential computations is r_{aver}. Then, we write the expression for parallel efficiency as

$$E(n,p,R) = \frac{r_{seq}}{r_{aver}} \cdot \frac{T_s(n, r_{aver})}{pT_p(n,p,R)}. \quad (2)$$

For homogeneous parallel system $r_i = const$ and (2) is equal to usual parallel efficiency

$$E(n,p,R) = \frac{T_s(n)}{pT_p(n,p)}.$$

For homogeneous parallel system it is supposed that increase of system performance is in proportion to number of processes. For heterogeneous system increase of system performance depends on number of processors and their performance.

There is no generally accepted metrics that measure the scalability. In this paper we follow the most generally accepted metrics proposed in [1]. During execution, a parallel algorithm incurs overheads due to different reasons. The total time spent by all processors doing the work that is not done by sequential algorithm is the *total overhead*, $T - o$. The total time spent by all processors is $pT_p(n,p,R)$, and the total overhead is $pT_o(n,p,R)$, so

$$pT_p(n,p,R) = T_s(n, r_{aver}) + T_o(n,p,R)$$

and

$$E(n,p,R) = \frac{T_s(n, r_{aver})}{T_s(n, r_{aver}) + T_o(n,p,R)} = \frac{1}{1 + \frac{T_o(n,p,R)}{T_s(n, r_{aver})}}.$$

The *isoefficiency relation* that bounds problem size and number of processors to keep level of parallel efficiency is written as

$$T_s(n, r_{aver}) = KT_o(n,p,R).$$

Through algebraic manipulation this equation can be used to obtain the isoefficiency function.

Until now, we do not specify what we understand by size of problem being solved. In fact our extension does not depend on it. But to analyze particular parallel system we need to concretize the notion of problem size. There are two main approaches to expression of problem size: as volume of computation and as amount of primary memory. The first one is advocated, for example in [1], because it allows equating problem size and sequential time. The second one is advocated, for example in [4]. For a lot of algorithms it is assumed that data structures it manipulates fit in primary memory. The maximum problem size it can solve is limited by the amount of primary memory is available. This is the reason to treat space as the limiting factor for scalability analysis.

3 Scalability Analysis of Matrix Multiplication Algorithm SUMMA

Let us consider the Scalable Universal Matrix Multiplication Algorithm [2] executing on parallel architecture supporting simultaneous point-to-point

communications. The algorithm supposed that matrices are distributed over two-dimensional process grid in such way that rows of matrices are assigned to the same row of process grid and columns of the matrices are assigned to the same column of process grid. The algorithm is based on implementing the broadcast as passing a message around the logical ring that forms the row or column. This allows pipelining computations and communications. For homogeneous parallel systems in [2] it is demonstrated that the parallel efficiency depends on memory use per node $O(n^2)$and number of processes p only through their ratio. That is isoefficiency function is $N = O(n^2)$ and system is highly scalable.

For heterogeneous parallel system we analyze two strategies of data distribution:

- homogeneous – data are distributed between processors evenly;
- heterogeneous – data are distributed between processors according to their performance [8,9].

Let us consider square matrices $N \times N$ and square process grid. Let communication time to be approximated as $t = a + bL$, where a is start-up cost and b is cost of transfer of one unit of data. Let also $N \gg \sqrt{p}$.

3.1 Homogeneous Distribution of Data

In [2] it is shown that under such assumption time complexity of algorithm with homogeneous distribution of data on homogeneous system is

$$T_p(N^2, p) = \frac{N^3}{pr} + 2N(a + \frac{N}{\sqrt{p}}b),$$

where r is performance of the system processors.

On heterogeneous system time complexity is determined by time elapsed for local updates on the weakest processor with performance r_{min}. It can be written as

$$T_p(N^2, p, R) = \frac{N^3}{pr_{min}} + 2N(a + \frac{N}{\sqrt{p}}b). \tag{3}$$

Because $T_s(N^2, r_{aver}) = N^3/pr_{aver}$ the total overhead is

$$T_o(N^2, p, R) = pT_p(N^2, p, R) - T_s(N^2, r_{aver}) = p[\frac{N^3}{pr_{min}} + 2N(a + \frac{N}{\sqrt{p}}b)] - \frac{N^3}{pr_{aver}}$$

and the isoefficiency relations is

$$\frac{N^3}{pr_{aver}} = K\{p[\frac{N^3}{pr_{min}} + 2N(a + \frac{N}{\sqrt{p}}b)] - \frac{N^3}{pr_{aver}}\}.$$

One can see that in general the relation depends on r_{aver} and r_{min}, that is on performances of the add-on processors. If the number of processors is big enough then the average performance is a weakly changing value that can be approximated with constant. That is $r_{aver} \approx const$. Suppose that r_{min} is a constant also. In such supposition the isoefficiency function of heterogeneous parallel system with homogeneous distribution of data is the same as of homogeneous parallel system $N^2 = O(p)$.

3.2 Heterogeneous Distribution of Data

As shown in [6] time complexity for heterogeneous distribution of data can be expressed as

$$T_p(N^2, p, R) = \frac{N^3}{pr_{aver}} + 2N(a + \frac{HN}{\sqrt{p}}b), \tag{4}$$

where H is *level of network heterogeneity* (ratio of maximal and minimal processors performance). The total overhead is

$$T_o(N^2, p, R) = p[\frac{N^3}{pr_{aver}} + 2N(a + \frac{HN}{\sqrt{p}}b)] - \frac{N^3}{pr_{aver}} = p[2N(a + \frac{HN}{\sqrt{p}}b)]$$

and the isoefficiency relations is

$$\frac{N^3}{r_{aver}} = 2KpN(a + \frac{HN}{\sqrt{p}}b).$$

The relation depends on H. In supposition that H is a constant the isoefficiency function of heterogeneous parallel system with heterogeneous distribution of data is $N^2 = O(p)$ also.

Comparing equations 3 and 4 we can conclude that homogeneous distribution of data provide the better distribution of communications but heterogeneous distribution of data provide the better distribution of computations. Equating right-hand members of equations 3 and 4 we can derive a condition under which the both data distribution strategies have the same cost

$$\frac{N}{\sqrt{p}}(\frac{1}{r_{min}} - \frac{1}{r_{aver}}) = 2(H-1)b.$$

4 Experimental Results

We carried out some experiments to estimate the influence of different data distribution strategies to scalability of algorithm SUMMA on heterogeneous architecture. We used a parallel system consisting of 1,6 GHz and 2,2 GHz dual PowerPC 970 interconnected with Myrinet. The performance of the 1,6 GHz processor demonstrated on sequential matrix multiplication relates to that of the 2,2 GHz processor as 3:4. One processor ran one process of parallel program. Square processor grid and square matrices were used. The size of grid was varied from 2x2 to 13x13. To the best of our knowledge it is the first experimental evaluation of heterogeneous system of such size. The size of the matrices was selected as $5000\sqrt{p} \times 5000\sqrt{p}$. That is we maintained memory use per processor. The processors grid was formed the following way. Processors with the same performance were assigned to the same column of the grid. The first half of columns consist of 1,6 GHz processors and the second half of columns consist of 2,2 GHz processors. If size of processor grid $\sqrt{p}$ is odd then first $\sqrt{p}/2+1$ columns consist of 1,6 GHz processors. Thus we ensured that r_{min} and H are the constants and r_{aver} is approximately constant.

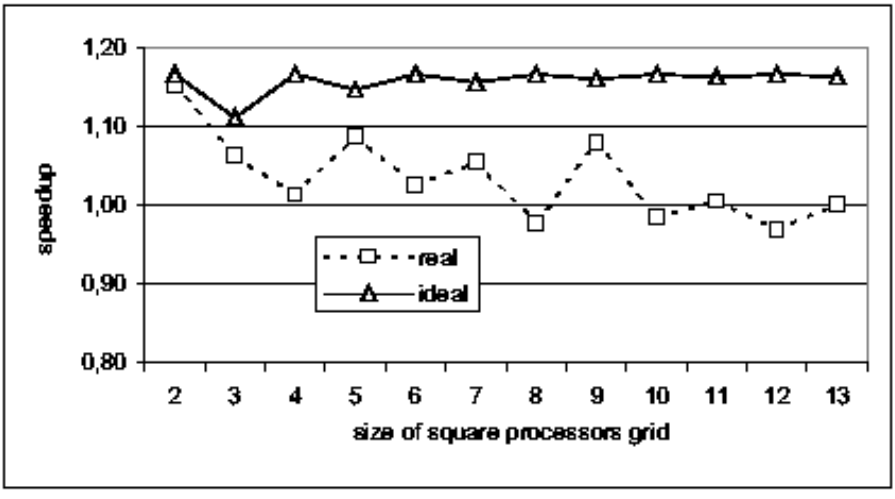


Fig. 1. Ideal and real speedups achieved using heterogeneous distribution of data on parallel system consisting of 1,6 GHz and 2,2 GHz dual PowerPC 970 interconnected with Myrinet. The square matrices $5000\sqrt{p} \times 5000\sqrt{p}$ were multiplied on square processors grid $\sqrt{p} \times \sqrt{p}$. The first half of processors grid columns consisted of 1,6 GHz processors. The second half of processors grid columns consisted of 2,2 GHz processors.

Figure 1 presents the ideal and real speedups achieved using heterogeneous distribution of data relative to homogeneous distribution of data. If we neglect the communication cost then the ideal speedup can be computed as r_{aver}/r_{min}. One can see that for small size of processors grid the real speedup is close to ideal one. For large size of processors grid the performances of the both data distribution strategies are approximately the same.

Figure 2 presents parallel efficiencies of parallel system with homogeneous and heterogeneous data distributions. One can see that starting from 4x4 processors grid the both parallel efficiencies are approximately constant. This confirm theoretical results that varying sizes of matrices and of processes grid in accordance with isoefficiency function $N^2 = O(p)$ keeps the parallel efficiencies of algorithm SUMMA constant for the both strategies of data distribution.

5 Related Work

We can point few papers considering scalability of heterogeneous parallel systems. Luis Pastor and Jose L. Bosque in [5] extend isoefficiency metrics for the case of problem size considered as volume of computations. The extension is based on the same definition of parallel efficiency as ratio of the ideal execution time of parallel computation and the real one as we use. They consider the case of workload divisible between processors *ad infinitum* only. In [6] the metrics proposed in [7] for scalability analysis of linear algebra algorithms was

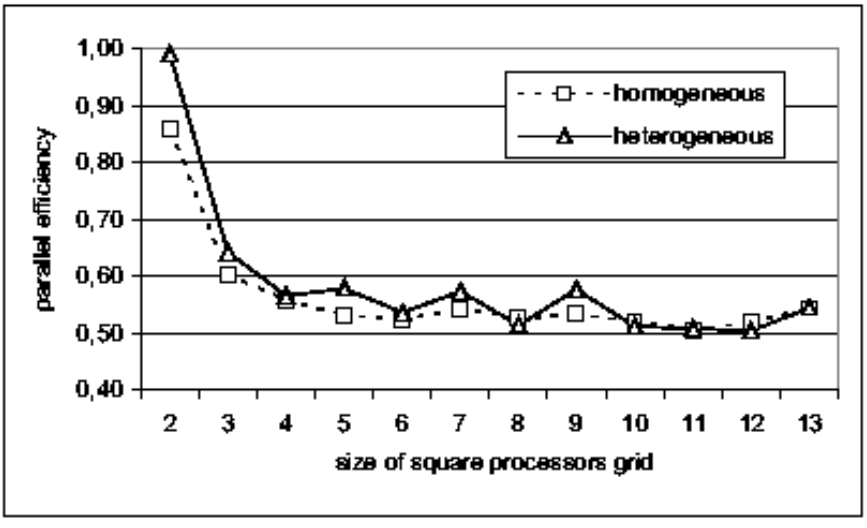


Fig. 2. Parallel efficiencies of parallel system with homogeneous and heterogeneous data distributions. The computing system consist of 1,6 GHz and 2,2 GHz dual PowerPC 970 interconnected with Myrinet. The square matrices $5000\sqrt{p} \times 5000\sqrt{p}$ were multiplied on square processors grid $\sqrt{p} \times \sqrt{p}$. The first half of processors grid columns consisted of 1,6 GHz processors. The second half of processors grid columns consisted of 2,2 GHz processors.

extended for heterogeneous parallel systems. The extended metrics was used to the analysis of influence of different strategies of heterogeneous distribution of computation on scalability of matrix multiplication algorithm SUMMA.

6 Conclusions

We extend the isoefficiency metrics for heterogeneous parallel systems without concretizing the problem size notion. We apply the extension to analysis of influence of different data distribution strategies on scalability of algorithm SUMMA that is scalable on homogeneous parallel platform. We show that under some assumptions the isoefficiency function for heterogeneous parallel systems is the same as for homogeneous parallel system with the both homogeneous and heterogeneous distribution of data. We experimentally corroborate this statement on the heterogeneous system consisting of 169 processors.

Heterogeneous data distribution improves the distribution of computations at the expense of deterioration of the communications distribution. So, for parallel computing systems consisting of hundred of processors appropriateness of heterogeneous distribution of computations should be estimated more precisely.

Acknowledgements

This research is partly supported by the program of the Presidium of the Russian Academy of Sciences "Mathematical modeling and intellectual systems".

The author thanks administration of the Joint Supercomputer Center, Moscow for possibility to conduct experiments and personally Vladilen Opalev, Svetlana Jagodkina, and Natalya Skrypnik for their help.

References

1. A. Grama, A. Gupta and V. Kumar. Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures, *IEEE Parallel & Distributed Technology*, 1993, Vol.1, No. 3, pp. 12-21
2. R. van de Geijn and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *Concurrency: Practice and Experience*, 9(4) 255-274 (1997)
3. A. Kalinov. Heterogeneous two-dimensional block-cyclic data distribution for solving linear algebra problems on heterogeneous networks of computers. *Programming and Computer Software*, Vol. 25, No. 2, 1999, pp. 3-11. Translated from Programmirovanie, Vol 25, No.2
4. M. Quinn. Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2004
5. L. Pastor and J. L. Bosque, An efficiency and scalability model for heterogeneous clusters. *Proceedings of Cluster 2001*, 8-11 October 2001, Newport Beach, CA, USA. IEEE Computer Society pp.427-434
6. A. Kalinov, Scalability Analysis of Matrix-Matrix Multiplication on Heterogeneous Clusters, *Proceedings of 3rd ISPDC/HeteroPar'04*, Cork, Ireland, July 05 - 07, 2004, IEEE CS Press, pp. 303-309
7. J. Dongarra, R. van de Geun, and D. Walker. Scalability Issues Affecting the Design of a Dense Linear Algebra Library, *Journal of Parallel and Distributed Computing*, 22, 523-537, 1994
8. Olivier Beaumont, Vincent Boudet, Antoine Petitet, Fabrice Rastello, and Yves Robert. A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). *IEEE Trans. Computers,* 2001, Vol.50, No.10, pp.1052-1070
9. E. Dovolnov, A. Kalinov, and S. Klimov, Natural Block Data Decomposition for Heterogeneous Clusters, *Proceedings of 17th International Parallel and Distributed Processing Symposium*, IEEE CS, Nice, France, April 2003, CD-ROM

A Variable Group Block Distribution Strategy for Dense Factorizations on Networks of Heterogeneous Computers

Alexey Lastovetsky and Ravi Reddy

Department of Computer Science, University College Dublin, Belfield, Dublin 4, Ireland
{alexey.lastovetsky, manumachu.reddy}@ucd.ie

Abstract. In this paper, we present a static data distribution strategy called Variable Group Block distribution to optimize the execution of factorization of a dense matrix on a network of heterogeneous computers. The distribution is based on a functional performance model of computers, which tries to capture different aspects of heterogeneity of the computers including the (multi-level) memory structure and paging effects.

1 Introduction

The paper presents a static data distribution strategy called Variable Group Block distribution to optimize the execution of factorization of a dense matrix on a network of heterogeneous computers. The Variable Group Block distribution strategy is a modification of Group Block distribution strategy, which was proposed in [1] for 1D parallel Cholesky factorization, developed into a more general 2D distribution strategy in [2] and applied to 1D LU factorization in [3], [4].

The Group Block distribution strategy is based on the performance model, which represents the speed of each processor by a constant positive number and computations are distributed amongst the processors such that their volume is proportional to this speed of the processor. However the single number model is efficient only if the relative speeds of the processors involved in the execution of the application are a constant function of the size of the problem and can be approximated by a single number. This is true mainly for homogeneous distributed memory systems where:

- The processors have almost the same size at each level of their memory hierarchies, and
- Each computational task assigned to a processor fits in its main memory.

But the model becomes inefficient in the following cases:

- The processors have significantly different memory structure with different sizes of memory at each level of memory hierarchy. Therefore, beginning from some problem size, the same task will still fit into the main memory of some processors and stop fitting into the main memory of others, causing the paging and visible degradation of the speed of these processors. This means that their relative speed will start significantly changing in favor of non-paging processors as soon as the problem size exceeds the critical value.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1074–1081, 2006.

– Even if the processors of different architectures have almost the same size at each level of the memory hierarchy, they may employ different paging algorithms resulting in different levels of speed degradation for the task of the same size, which again means the change of their relative speed as the problem size exceeds the threshold causing the paging.

Thus considering the effects of processor heterogeneity, memory heterogeneity, and the effects of paging significantly complicates the design of algorithms distributing computations in proportion with the relative speed of heterogeneous processors. One approach to this problem is to just avoid the paging as it is normally done in the case of parallel computing on homogeneous multi-processors. However avoiding paging in local and global heterogeneous networks may not make sense because in such networks it is likely to have one processor running in the presence of paging faster than other processors without paging. It is even more difficult to avoid paging in the case of distributed computing on global networks. There may not be a server available to solve the task of the size you need without paging.

Therefore, to achieve acceptable accuracy of distribution of computations across heterogeneous processors in the possible presence of paging, a more realistic performance model of a set of heterogeneous processors is needed. In [5], we suggested a functional performance model of computers that integrates some of the essential features underlying applications run on general-purpose common heterogeneous networks, such as the processor heterogeneity in terms of the speeds of the processors, the memory heterogeneity in terms of the number of memory levels of the memory hierarchy and the size of each level of the memory hierarchy, and the effects of paging. Under this model, the speed of each computer is represented by a continuous and relatively smooth function of problem size.

The Variable Group Block distribution strategy presented in this paper uses this functional performance model to optimize the execution of factorization of a dense square matrix on a network of heterogeneous computers.

The functional model does not take into account the effects on the performance of the processor caused by several users running heavy computational tasks simultaneously. It supposes only one user running heavy computational tasks and multiple users performing routine computations and communications, which are not heavy like email clients, browsers, audio applications, text editors etc.

The rest of the paper is organized as follows. In the next section, we present the Variable Group Block distribution strategy. We then show experimental results on a local network of heterogeneous computers to demonstrate the efficiency of the Variable Group Block Distribution strategy over the Group Block Distribution Strategy.

2 Variable Group Block Distribution

Before we present our Variable Group Block distribution strategy, we briefly explain the LU Factorization algorithm of a dense (**n**×**b**)×(**n**×**b**) square matrix A, one step of which is shown in Figure 1. **n** is the number of blocks of size **b**×**b** [6], [7]. On a homogeneous **p**-processor linear array, a CYCLIC(**b**) distribution of columns is used to distribute the matrix A. The cyclic distribution would assign columns of blocks with

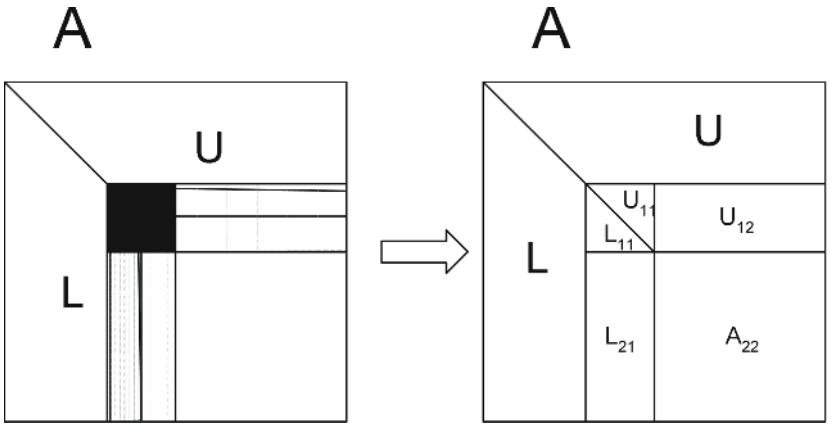


Fig. 1. One step of the LU factorization algorithm of a dense square matrix (**n**×**b**)×(**n**×**b**)

numbers 0,1,2,. . . ,**n**-1 to processors 0,1,2,. . . ,**p**-1,0,1,2,. . . ,**p**-1,0,. . . , respectively, for a **p**-processor linear array (**n**≫**p**), until all **n** columns of blocks are assigned. At each step of the algorithm, the processor that owns the pivot block factors it and broadcasts it to all the processors, which update their remaining blocks. At the next step, the next column of **b**×**b** blocks becomes the pivot panel, and the computation progresses. Figure 1 shows how the column panel, L_{11} and L_{21}, and the row panel, U_{11} and U_{12}, are computed and how the trailing submatrix A_{22} is updated. Because the largest fraction of the work takes place in the update of A_{22}, therefore, to obtain maximum parallelism all processors should participate in the updating. Since A_{22} reduces in size as the computation progresses, a cyclic distribution is used to ensure that at any stage A_{22} is evenly distributed over all processors, thus obtaining a balanced load.

Two load balancing algorithms, namely, Group Block algorithm and Dynamic Programming algorithm [7] have been proposed to obtain optimal static distribution over **p** heterogeneous processors arranged in a linear array. The Group Block distribution partitions the matrix into groups (or *generalized blocks* in terms of [2]), all of which have the same number of blocks. The number of blocks per group (size of the group) and the distribution of the blocks in the group amongst the processors are fixed and are determined based on speeds of the processors, which are represented by a single constant number. Same is the case with Dynamic Programming distribution except that the distribution of the blocks in the group amongst the processors is determined based on dynamic programming algorithm.

We propose a static distribution strategy called Variable Group Block distribution, which is a modification of the Group Block algorithm. It uses the functional model where absolute speed of the processor is represented by a function of a size of the problem. Since the Variable Group Block distribution uses the functional model where absolute speed of the processor is represented by a function of a size of the problem, the distribution uses absolute speeds at each step of the LU factorization that are based on the size of the problem solved at that step. That is at each step, the number of blocks per group and the distribution of the blocks in the group amongst the processors are

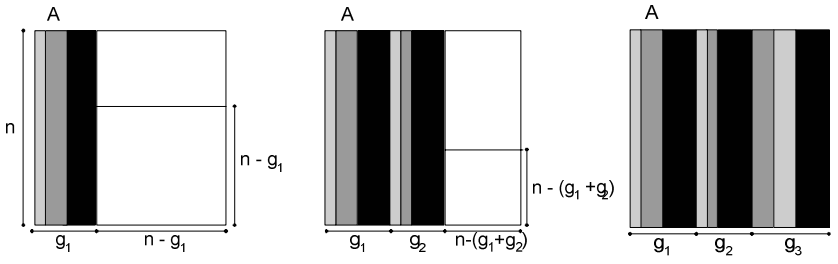


Fig. 2. The matrix *A* is partitioned using Variable Group Block distribution. The size of the matrix is shown in blocks of size **b**×**b**. This figure illustrates the distribution for **n**=18,**b**=32,**p**=3. The distribution inside groups G_1, G_2, and G_3 are {2,1,1,0,0,0}, {2,1,0,0,0}, and {2,2,1,1,0,0,0}. At each step of the distribution, the absolute speed of the processor is obtained based on the update of the trailing matrix. Since the Variable Group Block distribution uses the functional model where the absolute speed of the processor is represented by a function of the problem size, the distribution uses absolute speeds at each step that are based on the size of the problem solved at that step.

determined based on absolute speeds of the processors given by the functional model, which are based on solving the problem size at that step. Thus it takes into account the effects of (multi-level) memory structure and paging.

Figure 2 illustrates the Variable Group Block distribution of a dense square (**n**×**b**)×(**n**×**b**) matrix *A* over **p** heterogeneous processors. The Variable Group Block distribution is a static data distribution that vertically partitions the matrix into **m** groups of blocks of size **b** whose column sizes are $\mathbf{g}_1,\mathbf{g}_2,\ldots,\mathbf{g}_\mathbf{m}$ as shown in Figure 2.

The groups are non-square matrices of sizes (**n**×**b**)×($\mathbf{g}_1$×**b**), (**n**×**b**)×($\mathbf{g}_2$×**b**), ..., (**n**×**b**)×($\mathbf{g}_\mathbf{m}$×**b**) respectively. The steps involved in the distribution are:

1. The size $\mathbf{g}_1$ of the first group G_1 of blocks is calculated as follows:
   - Using the data partitioning algorithm [5], we obtain an optimal distribution of matrix *A* such that the number of blocks assigned to each processor is proportional to the speed of the processor. The optimal distribution derived is given by (x_i, s_i) ($0 \leq i \leq \mathbf{p}-1$), where x_i is the size of the subproblem such that $\sum_{i=0}^{\mathbf{p}-1} x_i = \mathbf{n}^2$ and s_i is the absolute speed of the processor used to compute the subproblem x_i for processor i. Calculate the load index $l_i = \frac{s_i}{\sum_{k=0}^{\mathbf{p}-1} s_k}$ ($0 \leq i \leq \mathbf{p}-1$).
   - The size of the group $\mathbf{g}_1$ is equal to $\lfloor 1/min(l_i) \rfloor$ ($0 \leq i \leq \mathbf{p}-1$). If $\mathbf{g}_1/\mathbf{p} < 2$, then $\mathbf{g}_1 = \lfloor 2/min(l_i) \rfloor$. This condition is imposed to ensure there is sufficient number of blocks in the group.
   - This group G_1 is now partitioned such that the number of blocks $g_{1,i}$ is proportional to the speeds of the processors s_i where $\sum_{i=0}^{\mathbf{p}-1} g_{1,i} = \mathbf{g}_1$ ($0 \leq i \leq \mathbf{p}-1$).
2. To calculate the size $\mathbf{g}_2$ of the second group, we repeat step 1 for the number of blocks equal to $(\mathbf{n}\text{-}\mathbf{g}_1)^2$ in matrix *A*. This is represented by the sub-matrix $\mathbf{A}_{\mathbf{n}\text{-}\mathbf{g}_1,\mathbf{n}\text{-}\mathbf{g}_1}$ shown in Figure 2. We recursively apply this procedure until we have fully vertically partitioned the matrix *A*.

Table 1. Specifications of the twelve computers. Paging is the size of the matrix beyond which point paging started happening.

Machine Name	Architecture	cpu MHz	Total Main Memory (kBytes)	Available Main Memory (kBytes)	Cache (kBytes)	Paging (LU)
X1	Linux 2.4.20-20.9 i686 Intel Pentium III	997	513304	363264	256	6000
X2	Linux 2.4.18-3 i686 Intel Pentium III	997	254576	65692	256	5000
X3	Linux 2.4.20-20.9bigmem Intel(R) Xeon(TM)	2783	7933500	2221436	512	11000
X4	Linux 2.4.20-20.9bigmem Intel(R) Xeon(TM)	2783	7933500	3073628	512	11000
X5	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	415904	512	8500
X6	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	364120	512	8500
X7	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	215752	512	8000
X8	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	134400	512	6500
X9	Linux 2.4.18-10smp Intel(R) XEON(TM)	1977	1030508	134400	512	6500
X10	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	409600	2048	5000
X11	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	418816	2048	5000
X12	SunOS 5.8 sun4u sparc SUNW,Ultra-5_10	440	524288	395264	2048	5000

3. For algorithms such as LU Factorization, only blocks below the pivot are updated. The global load balancing is guaranteed by the distribution in groups; however, for the group that holds the pivot it is not possible to balance the workload due to the lack of data. Therefore it is possible to reduce the processing time if the last blocks in each group are assigned to fastest processors, that is when there is not enough data to balance the workload then it should be the fastest processors doing the work. That is in each group, processors are reordered to start from the slowest processors to the fastest processors for load balance purposes.

In LU Factorization, the size of the matrix shrinks as the computation goes on. This means that the size of the problem to be solved shrinks with each step. Consider the first step. After the factorization of the first block of **b** columns, there remain **n**-1 blocks of **b** columns to be updated. At the second step, the number of blocks of **b** columns to update is only **n**-2. Thus the speeds of the processors to be used at each step should be based on the size of the problem solved at each step, which means that for the first step, the absolute speed of the processors calculated should be based on the update of **n**-1 blocks of **b** columns and for the second step, the absolute speed of the processors calculated should be based on the update of **n**-2 blocks of **b** columns. Since the Variable Group Block distribution uses the functional model where absolute speed of the processor is represented by a function of a size of the problem, the distribution uses absolute speeds at each step that are calculated based on the size of the problem solved at that step.

For two dimensional processor grids, the Variable Group Block algorithm is applied to columns and rows independently.

3 Experimental Results

A small heterogeneous local network of 12 different Solaris and Linux workstations shown in Table 1 is used in the experiments. The network is based on 100 Mbit Ethernet with a switch enabling parallel communications between the computers. The amount of memory, which is the difference between the main memory and free main memory shown in the tables, is used by the operating system processes and few other user application processes that perform routine computations and communications such as email clients, browsers, text editors, audio applications etc. These processes use a constant percentage of CPU.

For the parallel LU factorization application, the absolute speed of a processor must be obtained based on the execution of DGEMM routine on a dense non-square matrix of size $\mathbf{m}_1 \times \mathbf{m}_2$. The reason is that the computational cost of the application mainly falls into the update of the trailing submatrix, which is performed by this routine. Even though there are two parameters $\mathbf{m}_1$ and $\mathbf{m}_2$ representing the size of the problem, the parameter $\mathbf{m}_1$ is fixed and is equal to **n** during the application of the set partitioning algorithm [5].

To apply the set partitioning algorithm to determine the optimal data distribution for such an application, we need to extend it for problem size represented by two parameters, $\mathbf{m}_1$ and $\mathbf{m}_2$. The speed function of a processor is geometrically a surface when represented by a function of two parameters $\mathbf{s}=\mathbf{f}(\mathbf{m}_1,\mathbf{m}_2)$. However since the parameter $\mathbf{m}_1$ is fixed and is equal to **n**, the surface is reduced to a line $\mathbf{s}=\mathbf{f}(\mathbf{m}_1,\mathbf{m}_2)=\mathbf{f}(\mathbf{n},\mathbf{m}_2)$. The set partitioning algorithm can be extended here easily to obtain optimal solutions for problem spaces with two or more parameters representing the problem size. Each such problem space is reduced to a problem formulated using a geometric approach and tackled by extensions of our geometric set-partitioning algorithm. Con-sider for example the case of two parameters representing the problem size where neither of them is fixed. In this case, the speed functions of the processors are represented by surfaces. The optimal solution provided by a geometric algorithm would divide these surfaces to produce a set of rectangular partitions equal in number to the number of processors such that the number of elements in each partition (the area of the partition) is proportional to the speed of the processor.

The absolute speed of the processor in number of floating point operations per second is calculated using the formula $(2\times\mathbf{n}\times\mathbf{b}\times\mathbf{n}\times\mathbf{b})$/(execution time) where $\mathbf{n}\times\mathbf{b}$ is the size of the dense square matrix. The computer X6 exhibited the fastest speed of 130 MFlops for execution of DGEMM routine on a dense 8500×8500 matrix whereas the computer X1 exhibited the lowest speed of 19 MFlops for execution of DGEMM routine on a dense 4500×4500 matrix. The ratio $130/19 \approx 6.8$ suggests that the processor set is reasonably heterogeneous and it should also be noted that paging has not started happening at this problem size for both the computers.

We use a piece-wise linear function approximation to represent the speed function [5]. This approximation of speed function for a processor is built using a set of few experimentally obtained points. The block size **b** used in the experiments is 32, which is typical for cache-based workstations [3], [8].

Figure 3 shows the speedup of the LU Factorization application using the Variable Group Block distribution strategy over the application using the Group Block Distri-

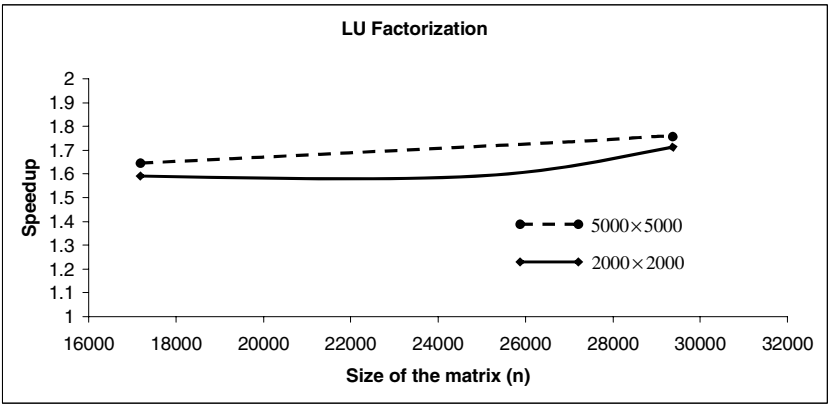


Fig. 3. Speedup of Variable Group Block Distribution over Group Block Distribution. For the Group Block Distribution, the single number speeds are obtained using DGEMM for a dense square matrix. For the solid lined curve, the matrix used is of size 2000×2000. For the dashed curve, the matrix used is of size 5000×5000.

bution strategy. The speedup calculated is the ratio of the execution time of the LU Factorization application using the Group Block distribution strategy over the execution time of the application using the Variable Group Block Distribution strategy.

4 Conclusions and Future Work

In this paper, we presented a static data distribution strategy called Variable Group Block distribution to optimize the execution of factorization of a dense matrix on a network of heterogeneous computers. The distribution is based on a functional performance model of computers, which integrates some of the essential features underlying applications run on general-purpose common heterogeneous networks, such as the processor heterogeneity in terms of the speeds of the processors, the memory heterogeneity in terms of the number of memory levels of the memory hierarchy and the size of each level of the memory hierarchy, and the effects of paging.

Future work would involve extension of Variable Group Block distribution strategy to optimize the execution of factorization of a dense matrix on a heterogeneous network of computers using a functional model that would incorporate communication cost parameters, namely, latency and the bandwidth of the communication links interconnecting the processors.

References

1. Arapov, D., Kalinov, A., Lastovetsky, A., Ledovskih, I.: Experiments with mpC: Efficient Solving Regular Problems on Heterogeneous Networks of Computers via Irregularization. Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel (IRREGULAR'98), Lecture Notes in Computer Science, Vol. 1457, (1998) 332–343

2. Kalinov, A., Lastovetsky, A.: Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers. Proceedings of the 7th International Conference on High Performance Computing and Networking Europe (HPCN Europe'99), Lecture Notes in Computer Science, Vol. 1593, (1999) 191–200
3. Barbosa, J., Tavares, J., Padilha, A.J.: Linear Algebra Algorithms in a Heterogeneous Cluster of Personal Computers. Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000) 147-159
4. Barbosa, J., Morais, C.N., Padilha, A.J.: Simulation of Data Distribution Strategies for LU Factorization on Heterogeneous Machines. Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)
5. Lastovetsky, A., Reddy, R.: Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)
6. Choi, J., Dongarra, J., Ostrouchov, L.S., Petitet, A.P., Walker, D.W., Whaley, R.C.: The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Rou-tines. Scientific Programming, Vol. 5, (1996) 173-184
7. Beaumont, O., Boudet, V., Petitet, A., Rastello, F., Robert, Y.: A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). IEEE Transactions on Computers, Vol. 50, (2001) 1052–1070
8. Blackford, L.S., Choi, J., Cleary, A., DAzevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users Guide. SIAM (1997)

Minimizing Associativity Conflicts in Morton Layout

Jeyarajan Thiyagalingam[1,2], Olav Beckmann[1], and Paul H.J. Kelly[1]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2AZ, United Kingdom
www.doc.ic.ac.uk/{~jeyan,~ob3,~phjk}
[2] Harrow School of Computer Science, University of Westminster,
Watford Road, Northwick Park, Harrow HA1 3TP, United Kingdom
t.jeyan@wmin.ac.uk

Abstract. Hierarchically-blocked non-linear storage layouts, such as the Morton ordering, have been shown to be a potentially attractive compromise between row-major and column-major for two-dimensional arrays. When combined with appropriate optimizations, Morton layout offers some spatial locality whether traversed row- or column-wise. However, for linear algebra routines with larger problem sizes, the layout shows diminishing returns. It is our hypothesis that associativity conflicts between Morton blocks cause this behavior and we show that carefully arranging the Morton blocks can minimize this effect. We explore one such arrangement and report our preliminary results.

1 Introduction

For dense multidimensional arrays, programming languages mandate one of the two canonical layouts — row-major and column-major. Traversing an array in its major order results in excellent spatial locality; however, traversing an array in opposite order to its major order can lead to an order-of-magnitude worse performance.

In our earlier work [1] we considered whether Morton layout can be an alternative storage layout to canonical layouts. Although Morton layout offers equal spatial locality both in row- and column-major order traversals, our early work in the area suggested that the performance of standard Morton layout may be disappointing. Following this observation, we proposed two optimization schemes for Morton arrays — unrolling and alignment. By exhaustively evaluating these optimizations, we demonstrated that unrolling combined with strength-reduction of the Morton index calculation and correct alignment of the base address of Morton arrays can lead to a significant improvement in performance [1]. A key remaining weakness which we address in this paper is that the performance of Morton layout tends to deteriorate with larger problem sizes.

Contributions of This Paper. In this paper we discuss how minimizing associativity conflicts in the two-dimensional Morton layout may lead to further improvements in performance. The main contributions of this paper are:

- We perform an in-depth analysis of associativity conflicts in Morton layout.
- We propose a hybrid layout scheme to minimize associativity conflicts in Morton arrays, and we discuss combining this scheme with a modest amount of padding.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1082–1088, 2006.

– We demonstrate the effectiveness of our proposed scheme through an experimental evaluation, using a suite of non-tiled micro-benchmarks.

This work differs from other work in this area, which mainly focused on optimizing for temporal locality through hierarchical tiling, by targeting spatial locality of non-tiled applications.

Structure of the Remainder of This Paper. In Section 2 we discuss related and previous work relevant to this paper. We illustrate the impact of associativity conflicts in Morton arrays and we propose a variant of Morton layout to address this problem in Section 3. Following this, we evaluate and report performance results for our proposed scheme in Section 4. Section 5 concludes the paper and discusses future work.

2 Previous Work

Many authors have studied recursive layouts in the context of performance optimization. Notably, Wise *et al.* [2], Chatterjee *et al.* [3,4], and Gustavson [5] pioneered these layouts, focusing on optimizing for temporal locality. Their implementations are either tiled or recursively formulated. In [5] Gustavson adopts a similar approach to Chatterjee [3, 4]; however, Gustavson's work is focused on deriving optimal layouts for particular problems rather than a generic solution.

Recursive layouts, by definition, require a complete decomposition up to the element level. Many authors [3, 4, 6] have identified that such a complete decomposition may lead to increased conflict misses between different blocks, and have proposed different variants to fully recursive layouts. Chatterjee *et al.* [4] propose a family of non-linear alternative layouts, called 4D layouts, which intermix recursive and linear layouts. In one such variant, Chatterjee *et al.* divide two-dimensional arrays into linearly arranged tiles, which are themselves blocked.

Drakenberg *et al.* [6] propose a semi-hierarchical layout (called HAT) and a linear-algebra framework to determine conflict misses at compile time. Their layout is very similar to one of the 4D layouts mentioned in [3, 4]. Their work, similar to ours, also considers non-tiled or non-recursive algorithms but does not discuss padding.

3 Conflict Misses in Morton Layout

Associativity Conflicts in the Standard Morton Layout. One of the advantages of using the standard Morton scheme, where blocking is applied recursively up to the element level, is its simplicity — there is no need to choose blocking factors. This is potentially useful for developing programs independently of underlying architectural features. However, different sub-blocks within a Morton array may conflict with each other. We demonstrate this effect in Figure 1(a): For a page-sized direct-mapped cache, every page-sized sub-block of a Morton array conflicts with every other sub-block. This appears to suggest that complete decomposition is sub-optimal.

In general, for a w-way associative cache with capacity C, addresses aligned at $(\frac{C}{w})$ bytes are mapped to the same set. If each word is l bytes, each way holds $\frac{C}{wl}$ words.

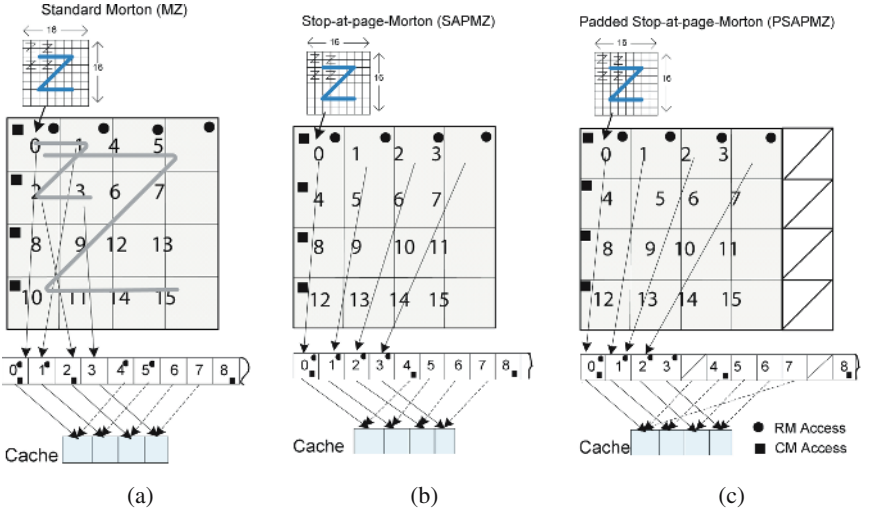


Fig. 1. Conflict Misses in Morton Layout. We show three different variants of the Morton scheme, where each location in the array represents a page-sized Morton block. In each case, we also show the mapping of pages into linear memory and into a direct-mapped 4-page cache. When accessing the standard Morton array (MZ) in row-major order, Morton pages 0 and 1 conflict with pages 4 and 5. Similarly, when accessing the MZ array in column-major order, pages 0 and 2 conflict with pages 8 and 10. This generates 2 misses per row or column for each traversal order. However, when accessing the Morton variant (SAPMZ) in row-major order, traversal of a single row is free from conflicts, but a column-major traversal suffers worse conflicts than the MZ layout. The diagram on the far right (PSAPMZ) shows how padding can be used to eliminate systematically recurring conflicts for column-major traversal. The same analysis is valid for caches with higher levels of associativity.

Note that C, w and l are typically powers-of-two, so $\frac{C}{wl}$ will also be a power-of-two. Two arbitrary word-addresses s and t collide if $|s-t| = \frac{C}{wl}$. With row-major layout, this happens with `A[i,s]` and `A[i,t]`, or for elements separated by a multiple of $|s-t|$. With Z-Morton layout this happens with `A[i,u]` and `A[i,v]` when $|u-v| = \sqrt{\frac{C}{lw}}$, if $\frac{C}{wl}$ is an even power-of-two (*i.e.*, if $\frac{C}{wl}$ words form a "square"). If $\frac{C}{wl}$ is an odd power-of-two, addresses `A[i,u]` and `A[i,v]` collide if $|u-v| = 2\sqrt{\frac{C}{2lw}}$.

Example. As a practical example, consider the ADI algorithm. In a slightly simplified form, this contains the loop shown in Figure 2. For good performance, this loop requires row-to-row re-use: the reference `A[i-1][j]` should come from cache, having been loaded on the previous iteration of the for-`i` loop. At what datasize will associativity conflicts prevent this in a standard Morton array? According to our analysis in the preceding paragraph, on a Pentium 4 processor with an 8-way set-associative 512KB L2 cache, and an array of 8-byte double words, addresses `A[i,u]` and `A[i,v]` will conflict ($\frac{C}{wl}$ is odd in this case) when $|u-v| = 2\sqrt{\frac{512\times 2^{10}}{2\times 8\times 8}} = 128$. With an 8-way set-associative cache, this means that we begin to lose row-to-row re-use at a row length of

```
1  for( int i = 1; i < sz; ++i )
2    for( int j = 0; j < sz; ++j )
3      A[i][j] += A[i-1][j];
```

Fig. 2. Simplified loop from the ADI algorithm, requiring row-to-row re-use for good performance

$128 \times 8 = 1024$. This observation is confirmed by our experimental results for the Adi algorithm on a Pentium 4 processor [1, 7] (see also Figure 3).

Stop-at-Page Morton Layout. A Hybrid Scheme. An alternative to standard Morton, as suggested by Chatterjee *et al.* [4] and Drakenberg *et al.* [6], is to divide the array into row-major or column-major ordered blocks and arranging the elements within each block in Z-Morton order. Choosing page-sized Morton blocks guarantees unbiased TLB behavior for both row-major and column-major traversals. We refer to this scheme as Stop-at-Page-Morton (since we stop the blocking at page-level). This enables us to utilize the compromise property of Morton layout and to minimize the effects of associativity conflicts among Morton blocks at the same time. The resulting effect on associativity conflicts is shown in Figures 1(b).

Notice, however, the increased dilation effect and the introduction of systematically repeating conflicts for column-major traversal of SAPMZ arrays. A simple method to avoid these systematically recurring conflicts is to pad each row of the array by a Morton block, whenever the number of Morton blocks in a row is even. We refer to this technique as Padded Stop-at-Page-Morton scheme (PSAPMZ). Although PSAPMZ does not improve spatial locality for column-major traversal, it does minimize associativity conflicts for column-major traversal. This is illustrated in Figure 1(c).

Implementation Issues. For standard Morton arrays, it is necessary to round up the array sizes of each dimension to the next power-of-two. For Stop-at-Page-Morton, padding is only necessary up to the next page size and will never be worse than the storage requirements for standard Morton, except for array sizes smaller than a page. The Padded-Stop-at-Page-Morton pads the row-length by an additional page-sized Morton block when the number of such blocks in a row would otherwise be even.

Page size may vary from architecture to architecture, and may not always correspond to an even power-of-two number of words. For most x86 variants (including the systems we used to test our hypothesis) the page size is 4KB, and we chose a 16×16 array of 8-byte doubles (half a page) as the largest Morton block.

4 Experimental Evaluation

Benchmark Kernels and Architectures. To test our proposed Stop-at-Page-Morton and Padded Stop-at-Page-Morton layouts, we have collected a suite of simple implementations of standard, non-tiled numerical kernels operating on two-dimensional arrays and carried out experiments on the Pentium 4 architecture. Table 1 summarizes the details of the architecture. The kernels used in our experiments are shown in Table 2. We carried out measurements over a full range of problem sizes and followed the experimental approach detailed in [7] to minimize the effects of external interference.

Table 1. Cache and CPU configuration used in the experiments. Compiler and compiler flags match those used by the vendor in their SPEC CFP2000 (base) benchmark reports [8].

Processor	Operating System	L1/L2/Memory Parameters	Compiler and Flags Used
Pentium 4 2.0 GHz	Linux 2.4.26	L1 D-cache: 4-way, 8KB, 64B cache line L2 cache: 8-way, 512KB, 128B cache line Page size: 4KB Main Memory: 512MB DDR-RAM	Intel C/C++ Compiler v8.1 -xW -ipo -O3 -static

Table 2. Numerical kernels used in our experimental evaluation [7]

MMijk	Matrix multiply, ijk loop nest order (usually poor due to large stride)
MMikj	Matrix multiply, ikj loop nest order (usually best due to unit stride)
Jacobi2D	Two-dimensional four-point stencil smoother
ADI	Alternating-direction implicit kernel, ij,ij order
Cholesky	K-variant (usually poor due to large stride)

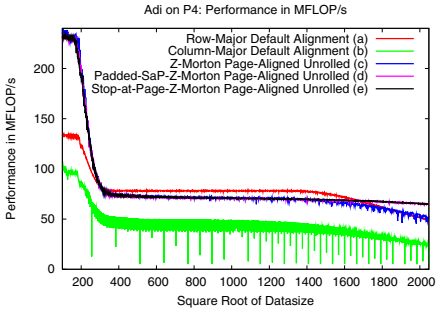
Performance Results. Figure 3 shows our results in detail, and we make some comments directly in the figure. For each experiment, we give a broad characterization of the performance of the different Morton schemes we tested.

Impact of the Padded and Non-padded Stop-at-Page-Morton Scheme. For Adi, MMijk and Cholesky-k kernels our theoretical conclusions from Section 3 are supported by our experimental data. For MMijk and Cholesky, the SAPMZ scheme does not offer an improvement over standard Morton; however, padding the Stop-at-Page-Morton scheme does help in these benchmarks. For the remaining benchmarks, Jacobi2D and MMikj, neither SAPMZ nor Padded SAPMZ offer an improvement over standard Morton.

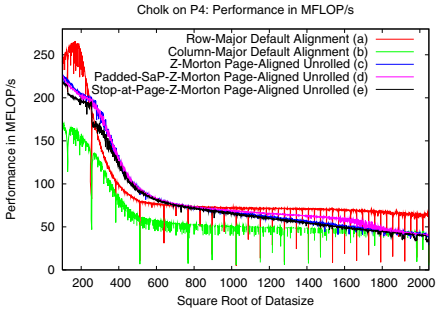
5 Conclusions and Future Work

Our hypothesis of how conflict misses in Morton layout can be minimized by the SAPMZ and Padded SAPMZ layouts is supported by some but not all experimental results. The padded SAPMZ scheme, in contrast to the SAPMZ scheme, has consistently improved performance. Further, the reduction in associativity conflicts offered by padded SAPMZ results in both row-major and column-major traversals of hierarchical arrays being yet closer in performance to row-major traversal of row-major arrays. There are number of interesting issues that remain to be addressed:

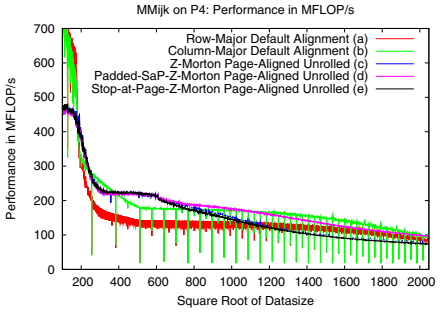
- Exploring large arrays. Our experimental results are limited to a range of problem sizes where the performance gain by SAPMZ/Padded SAPMZ occurs close to the upper limit of the problem sizes we consider. We would like to extend our work such that it covers a larger range of problem sizes.
- Hardware performance counters: Some features in the graphs may be explained with the help of hardware performance counters. We plan to explore this further by using one of the available hardware performance counter measurement tools.



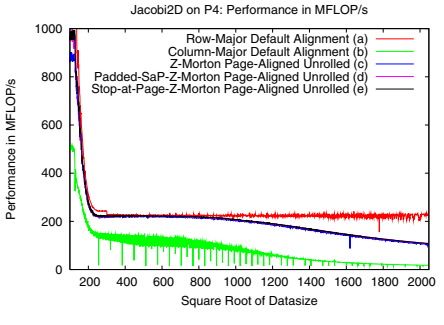
- Standard Morton performs almost as well as row-major.
- The performance of standard Morton begins to deteriorate around 1600×1600.
- Both SAPMZ and Padded SAPMZ sustain the performance for problem sizes larger than 1600×1600.

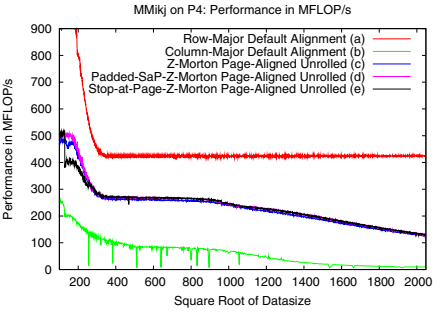


- Both SAPMZ and Padded SAPMZ never perform worse than standard Morton.
- SAPMZ does not result in a noticeable improvement over standard Morton.
- For a sub-range of problem sizes between 900×900 and 1900×1900, the padded SAPMZ scheme out-performs standard Morton and SAPMZ.



- Both SAPMZ and Padded SAPMZ never perform worse than standard Morton.
- SAPMZ does not result in a noticeable improvement over standard Morton.
- For problem sizes larger than about 512×512, Padded SAPMZ out-performs standard Morton and SAPMZ.





- For the Jacobi2D and MMikj benchmarks, SAPMZ and Padded SAPMZ scheme do not offer a performance improvement over standard Morton.

Fig. 3. Performance of different layouts using a suite of micro-benchmarks running on the Pentium 4 system described in Table 1

– Mixed Morton layouts: So far, we have only considered Z-Morton as the underlying layout. Mixing this with other members of the Morton layout family, which may have complementary effects, may lead to lower associativity conflicts.

References

1. Thiyagalingam, J., Beckmann, O., Kelly, P.H.J.: Improving the performance of Morton layout by array alignment and loop unrolling reducing the price of naivety. In Rauchwerger, L., ed.: Proceedings of 16th International Workshop on Languages and Compilers for Parallel Computing. Volume 2958 of LNCS., Springer-Verlag (2003) 241–257
2. Wise, D.S., Frens, J.D., Gu, Y., Alexander, G.A.: Language support for morton-order matrices. In: PPoPP '01: Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming. (2001) 24–33
3. Chatterjee, S., Lebeck, A.R., Patnala, P.K., Thottethodi, M.: Recursive array layouts and fast parallel matrix multiplication. In: SPAA '99: Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures. (1999) 222–231
4. Chatterjee, S., Jain, V.V., Lebeck, A.R., Mundhra, S., Thottethodi, M.: Nonlinear array layouts for hierarchical memory systems. In: ICS '99: Proceedings of the 13th International Conference on Supercomputing. (1999) 444–453
5. Gustavson, F.G.: New generalized data structures for matrices lead to a variety of high performance algorithms. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waniewski, J., eds.: PPAM 2001: Proceedings of the 4th International Conference on Parallel Processing and Applied Mathematics. Volume 2328 of LNCS., Springer-Verlag (2002) 418–436
6. Drakenberg, P., Lundevall, F., Lisper, B.: An efficient semi-hierarchical array layout. In: Proceedings of the Workshop on Interaction between Compilers and Computer Architectures, Kluwer (2001) Available via `www.mrtc.mdh.se`.
7. Thiyagalingam, J.: Alternative Array Storage Layouts for Regular Scientific Programs. PhD thesis, Department of Computing, Imperial College, London, U.K. (2005)
8. SPEC 2000 CPU Benchmarks: `http://www.specbench.org/` (2000)

A Parallel Algorithm for Solving the Reversal Median Problem

Matthias Bernt, Daniel Merkle, and Martin Middendorf

Department of Computer Science, University of Leipzig, Germany
{bernt, merkle, middendorf}@informatik.uni-leipzig.de

Abstract. We present a new algorithm called rEvoluzer II for solving the Reversal Median problem (RMP). Similar to its predecessor rEvoluzer I the new algorithm can preserve conserved intervals but it has the additional property that it is suitable for parallelization. For the parallel version of rEvoluzer II a master-slave parallelization scheme is used and several methods for reducing parallelization overheads have been incorporated. We show experimentally that rEvoluzer II achieves very good results compared to other methods for the RMP. It is also shown that the parallel version has good scaling behavior for a not too large number of processors.

1 Introduction

The genomes of species can be seen as arrangements of genes, rearranged during their evolution. Unichromosomal genomes can be described as signed permutations ("signed" is omitted in this paper). Reversals are rearrangement operations reversing the order of a subsequence of neighbored genes and the orientation of the involved genes. The Reversal Median Problem (RMP) is to find for three given genomes (permutations) a genome such that the number of reversals needed to transform it into the given genomes is minimal. The RMP is known to be NP-hard [7]. A solution method for the RMP is often used as a building block for solution methods for the Multiple Genome Rearrangement Problem (MGRP), where for a given set of permutations a phylogenetic tree with a minimal reversal distance sum over its edges has to be constructed.

Well known RMP Solvers are from Siepel et al. [9] and Caprara [7]. Both algorithms are exact, i.e. find a solution with minimal score, and are incorporated into the GRAPPA [10] software for solving the MGRP. Bourque [6] proposed a heuristic RMP solver which is the building block of the MGR algorithm for solving the MGRP. In MGR the RMP is solved iteratively by an evaluation of all possible reversals for each genome, in order to identify reversals that presumably bring the three genomes closer to each other until they meet in the proposed ancestral genome. Heuristics are used to determine the applied reversals.

The genome arrangements found in living species often have structures in the form of groups of genes that are related [11]. Some methods to describe such structures on the level of genome arrangements have been proposed recently (e.g., conserved intervals [4], common intervals [8], or gene teams [3]). Recently, the

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1089–1096, 2006.

first approaches considering structures for solving the MGRP have been proposed [1, 2, 5]. In [5] we describe a median solver, called rEvoluzer, that takes conserved intervals into account. In this paper we propose a new version of rEvoluzer, called rEvoluzer II. In contrast to the first version of rEvoluzer (rEvoluzer I) the new algorithm is suitable for parallelization. We present experimental results which show that rEvoluzer II is a very competitive RMP solver. Moreover, we present a parallel version of rEvoluzer II and study its speedup behavior.

In Section 2 the RMP is introduced formally. Section 3 describes rEvoluzer II. The parallel version of rEvoluzer II is described in Section 4. Experimental results are presented in Section 5.

2 The Reversal Median Problem

The reversal distance $d(G_1, G_2)$ between two permutations G_1 and G_2 is the minimal number of reversals needed to transform G_1 into G_2. It can be computed in time $O(n)$, where n is the number of elements each permutation has. The RMP is to find for three given permutations $G = \{G_1, G_2, G_3\}$ a permutation M such that $\sum_{i=1}^{3} d(M, G_i)$ is minimal. In this paper the additional constraint that possible gene groups should not be destroyed is considered. Therefore the concept of conserved intervals is employed. A conserved interval of a set of signed permutations is an interval of neighbored integers that is bounded by a pair of integers with the following properties: i) it occurs for all given permutations in the same order or in the reversed order if both integers have opposite signs and ii) the set of intermediate integers between the bounding pair of integers is the same in all given permutations. Reversals which preserve the conserved intervals are called preserving.

3 Algorithm rEvoluzer II

The basic strategy of Siepel's RMP solver, MGR, and rEvoluzer I is to apply iteratively reversals, which are selected by some criteria, on the given permutations G_i until they meet or some stopping criterion is met. As an exact algorithm Siepel's RMP solver tries to apply all reversals whereas MGR and rEvoluzer I select in every iteration one promising candidate reversal that is applied (in rEvoluzer I there exists also a backtracking mechanism). Because the set of all reversals becomes very large even for moderate sized problem instances, Siepel's RMP solver can only be applied to small instances. A possible disadvantage of the strategy to reduce the candidate set of reversals to only one reversal is that it can lead the algorithm to unfavorable regions of the search space.

The new strategy used in rEvoluzer II is to build up for each given permutation G_i a set of candidate permutations F_i, called front. Let $F = \cup F_i$. In order to organize the search efficiently all already considered (visited) permutations are stored in a tree data structure $\mathcal{V}$. Each iteration of rEvoluzer II consists of the following three parts: i) expansion of the fronts by applying reversals on the permutations in F, ii) insertion of the new front elements into $\mathcal{V}$, iii) reduction

of the size of the fronts if they become too large. In a final post processing step of rEvoluzer II the best found solutions are selected from $\mathcal{V}$. In the following the four different steps are described in more detail.

Expansion Step: In this step the permutations in every front F_i are moved towards (with respect to reversal distance) the permutations G_h and G_j, where $h, i, j \in \{1, 2, 3\}$ are pairwise different. Therefore a set of candidate reversals is determined for each front element. The candidate reversals are exactly the reversals acting on only one cycle (for the definition of cycle see, e.g., [9]). These reversals are known to be preserving [5]. These reversals are rated firstly by the achieved reversal distance reduction relative to G_h and G_j and secondly by the achieved distance enlargement to G_i. The best reversals are applied to obtain the candidate permutations for the new front.

Merge Step: The union of the candidate permutation sets is computed and permutations that have been visited already in a former iteration are deleted.

Cut Step: The expansion and merge steps may reduce or increase the size of the front. To prevent an exorbitant increase the front is reduced in this step if a front exceeds a size given by parameter $\overline{\tau}$. Then the front is reduced to size $\underline{\tau}$. In this case triples (f_1, f_2, f_3), $f_i \in F_i$ of permutations are selected that have a minimal score, i.e., a minimal value of $S = d(f_1, f_2)+d(f_1, f_3)+d(f_2, f_3)+\sum_i d(f_i, G_i)$. At most $\underline{\tau}$ elements from each front with the lowest scores are put into the reduced front. In order to bound the maximum runtime of this step at most r_{max} candidate triples are selected randomly from the fronts. We chose $r_{\max} := 5 \cdot \max_i |F_i|$. For this case it can be shown that the probability that a permutation is never chosen in a random triplet is $< 1/e^5$ ($\lesssim 0.68\%$).

Post Processing: In this step the final results are selected from $\mathcal{V}$. For each permutation in $\mathcal{V}$ the sum of the distances to G_1, G_2, and G_3 is calculated and all permutations with a minimal distance sum are put in the result set.

4 Parallelization of Algorithm rEvoluzer II

Algorithm rEvoluzer II can easily be adapted to different problem sizes. Increasing the maximal front size $\overline{\tau}$ leads to better results while using more computation time. To overcome that problem rEvoluzer II has been parallelized. The parallel version of rEvoluzer II (called P-rEvoluzer II) is based on a master/slave approach with a centralized load balancing mechanism. The master process manages the three fronts F_i and the data structure $\mathcal{V}$. During a run of P-rEvoluzer II extensive data decomposition operations are performed, which is a common technique used in parallelization when large data structures occur. Furthermore, process interaction latency is hidden using multiple threads in the master process and data locality techniques are incorporated.

Partitioning the Fronts: The expansion step is parallelized by data partitioning and self scheduling. Whenever a slave process is out of work the master sends

a number of front elements (a chunk) to that slave. For partitioning F either the same chunk sizes are used (each of the $p-1$ slaves gets $|F|/(p-1)$ front elements once) or alternatively a $1/l$-split technique as described in the following is used. The front elements are partitioned such that there are $p-1$ chunks of size $|F|/(l \cdot (p-1))$, which is $1/l$th of all front elements. The remaining front elements are partitioned again, such that $1/l$ of the rest of all elements are partitioned in $p-1$ chunks again. For sufficient large values of $|F|$ this partitioning technique leads to $p-1$ chunks of size $|F| \cdot (l-1)^{(s-1)}/(l^s \cdot (p-1))$, where s is the number of partitioning steps. This procedure is continued until a minimal chunk size $c_{\min}$ is reached. The chunks are sent to the slaves in order of decreasing sizes. There are two main advantages using chunk scheduling: i) Load-imbalances are compensated, ii) the master/slave communication operations of the expansion step within one iteration of P-rEvoluzer II will not all be initiated within a short time interval. In such a situation the master would become a bottleneck. Note, that a large value l leads to a communication overhead, as the number of chunks is increased.

Hiding Latency in the Master: To cope with the potential problem that the master process becomes a bottleneck chunk scheduling and other techniques are applied. i) All chunks are sent and received asynchronously and non-blocking from/to the slaves. ii) The master process uses two threads such that computation and communication overlap. One thread is used for calculating chunk sizes and for sending and receiving the chunks, the other thread is used to process the received data (store them in $\mathcal{V}$). When more than one slave is used, the order in which the results are received from the slaves is not deterministic. Since the results of the cut step depend on the order of elements in F, this can lead to different outcomes of the algorithm. Therefore, the elements in F are sorted in lexicographic order before the cutting for all test runs (and only those) that were done to compare the computation times for different numbers of slaves.

Data Locality Technique for Visited Permutations: A common technique to reduce process interaction overhead is maximizing data locality. Within an expansion step of P-rEvoluzer II all processes use their local data structure $\mathcal{V}_i, i = 1, \ldots, p$ to check whether a permutation has already been visited. If so, this reduces the communication overhead, as the permutation has not to be sent to the master. Furthermore, computation time is saved, as the master must not determine if that permutation is already in $\mathcal{V}$. $\mathcal{V}_i$ is reinitialized after each iteration of P-rEvoluzer II. $\mathcal{V}_1$ is used by the master during the merge step to reduce the number of accesses to $\mathcal{V}$.

Parallelization of the Cut Step: The cut step has been parallelized in a straight-forward manner. Suppose front F_i has to be cut. Then the master broadcasts F to all slaves and maximal r_{max}/p front element triples are generated on each processor. The best triples are chosen to determine $\lceil \underline{\tau}/p \rceil$ permutations. They are gathered by the master and merged to build the new front. Note, that the parallelization of the cut step leads to different outcomes of the algorithm,

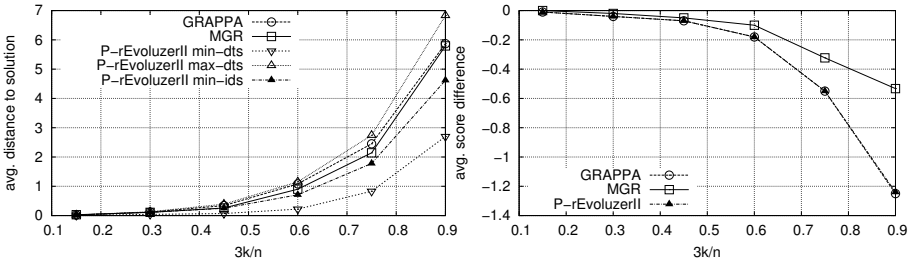


Fig. 1. Average distance to solution (left) and score difference (right) achieved by GRAPPA, MGR and P-rEvoluzer II; P-rEvoluzer II-{min-dts, max-dts, min-ids}, indicate the three different methods to select permutations among the permutations delivered by P-rEvoluzer II

as each processor has its own random number generator. Hence, we switch off the parallelization of this step when comparing computation times for different numbers of processors.

Parallelization of Post Processing: In the post processing phase the set of solutions in $\mathcal{V}$ is analyzed to find the best median. For this step, which is only done once, we use data partitioning. $\mathcal{V}$ is split in p parts and sent to the slaves. The sequences with the smallest score difference are sent back to the master. The master selects the best among these sequences.

5 Experimental Results

Setup: All experiments were conducted on a 32 dual-processor nodes Linux Cluster with AMD Opteron 248 processors that have 64-bit capabilities and 4GB main memory per node. P-rEvoluzer II was implemented in C++ using the MPICH v1.2.6 standard library for message-passing. Compilation was done with the Portland Group Inc. C++ compiler v6.0. For concurrency in the master process POSIX threads were used.

If not stated otherwise we used the following parameters in the experiments. The threshold for the size of a front F_i which initiates a cut step was set to $\overline{\tau} = 2000$. In a cut step the number of elements is reduced to $\underline{\tau} = 128$. The default self scheduling method was $1/l$-splitting with $l = 2$ and $c_{\min} = 2$. The master uses two threads to overlap communication and merging the results in $\mathcal{V}$. For time comparisons 20 RMP instances of size $n = 100$ have been solved. All results are average times for solving one instance.

Results: In order to evaluate the solution quality obtained with rEvoluzer II we compared it with other RMP solvers on 600 random instances of length $n = 100$. For each 100 instances the permutations were generated by applying $k \in \{5, 10, \ldots, 30\}$ random reversals to the identity permutation. The average distance to solution values and score differences are given in Figure 1 (further details in [6]). As the outcome of rEvoluzer II consists of a set of solutions, we

Table 1. P-rEvoluzer II: comparison of computation times using constant and dynamic chunk sizes for different number of processors; results in seconds

p	2	4	8	12	16	24	32	64
const.	456.62	184.22	95.40	72.18	61.84	54.24	47.45	44.78
1/2-splitting	458.68	170.42	81.81	57.31	46.62	37.86	34.37	31.68

used three methods to select permutations among the solutions with minimal scores differences: i) solutions with minimal interval distance sums, or ii) solutions with minimal distance to solution, or iii) solutions with maximal distance to solution. Note, that for real world instances the distance to solution is not known. Figure 1 shows that rEvoluzer II outperforms the other approaches on the test instances (due to limited space further results on this are not described).

In Table 1 the 1/2-split technique is compared with the method using constant size chunks. The results clearly indicate the superior performance of 1/2-splitting. For 24 processors it leads to an improvement of more than 30%. For a larger number of processors the improvement is smaller, as the master node becomes a bottleneck in both strategies. Note, that in the case of $p = 2$ processors only one slave is used. For $p = 1$ the computation times are identical (524.75 sec). As 1/2-splitting clearly outperforms the variant with constant chunk sizes only this technique is considered in the following. Figure 2 compares run times of

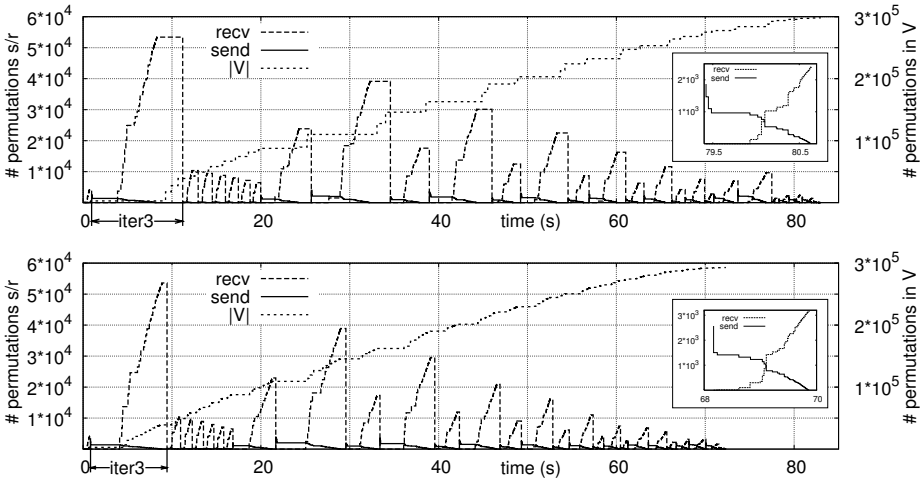


Fig. 2. P-rEvoluzer II with one thread (top) and two threads (bottom) in the master node; shown is the time-dependent behavior of: i) number of permutations in V, ii) number of permutations in the master which still have to be send within an iteration (*send*), iii) number of permutations which were already received by the master within one iteration (*recv*); typical run on one test instance with size $n = 100$; $p = 8$ processors

Table 2. Results for P-rEvoluzer II using different number of processors p; t_{exp}, t_{cut}, t_{post}: computation times for Expansion step, Cut step, Post processing step; c_{exp}, c_{post}: communication times for Expansion step and Post processing step; t_{comp}: overall computation time; E: efficiency

p	t_{exp}	c_{exp}	t_{cut}	t_{post}	c_{post}	t_{comp}	E
1	377.0	0.00	1.29	127.33	0.68	524.75	1.0
2	373.0	14.67	1.27	65.68	1.40	458.68	0.57
4	124.2	6.86	1.26	33.45	1.79	170.42	0.77
8	53.3	5.41	1.27	16.98	2.06	81.81	**0.80**
12	34.4	5.39	1.27	11.14	2.19	57.31	0.76
16	25.7	5.93	1.27	8.38	2.25	46.62	0.70
24	17.8	7.41	1.27	5.68	2.33	37.86	0.58
32	13.7	9.25	1.28	4.23	2.29	34.37	0.48
64	6.7	15.32	1.28	2.25	2.33	31.68	0.26

typical runs of the variant of P-rEvoluzer II that uses one thread in the master with the variant that uses two threads. 8 processors were used. The following three main differences can be observed between the two variants. i) The two-threaded variant merges the received permutations during sending and receiving. Hence, the time for merging the results in $\mathcal{V}$ is hidden. This can be seen when comparing the length of the time interval between receiving the last chunks of an expansion step and sending the first chunks of the next expansion step (e.g., in iteration 3 it is 2.89sec. in the one-threaded variant and 0.56sec. in the two-thread variant). ii) The increase of $|\mathcal{V}|$ is smoother for the two-thread variant, as permutations are merged by one thread as soon as they are received by the master. iii) The overall computation time is more than 12% smaller for the two-thread variant. Note, that the outcome of both variants can be slightly different (see Section 4). The small subfigures within Figure 2 show the behavior of *send* and *recv* in more detail for one iteration (iteration 27). It can be seen in both figures that 1/2-splitting has the effect that the expansion step is completed shortly after sending the last chunks to a slave.

Table 2 shows the computation and communication times for different numbers of processors for the expansion step, the cut step, and the post processing step. Because the serial version of the cut step was used for the comparison, there is no communication time for this step. Note, that the sum of all these times is less than the overall computation time since additional computation time is needed for sorting the front elements and a global synchronization operation before each expansion step. It can be seen that the overall computation time has a good scaling behavior up to about $p = 24$. For larger values of p the master node becomes a bottleneck. For $p = 64$ more than 50% of the overall computation time is used for communication, which leads to an efficiency of only 26%. For mid-range values of p P-rEvoluzer II has a very good efficiency. When using $p = 8$ processors an efficiency of 80% was achieved.

6 Conclusion

We presented the new algorithm rEvoluzer II for solving the Reversal Median Problem (RMP) which takes into account conserved intervals. The results of rEvoluzer II outperforms well-known approaches for the problem. In contrast to a former version of this algorithm (rEvoluzer I) it is suitable for parallelization. The parallel version (P-rEvoluzer II) uses a master-slave approach. Several techniques to reduce parallelization overheads are incorporated, namely multi-threading in the master node, chunk scheduling with decreasing chunk sizes and data locality techniques. An overall efficiency of more than 80% could be achieved for with 8 processors.

References

1. S. Bérard, A. Bergeron, and C. Chauve. Conservation of combinatorial structures in evolution scenarios. In *Proc. of the 2nd RECOMB Workshop on Comparative Genomics*, number 3388 in LNBI, pages 1–14. Springer Verlag, 2004.
2. A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve. Reconstructing ancestral gene orders using conserved intervals. In *Proc. of the 4th International Workshop on Algorithms in Bioinformatics (WABI 2004)*, number 3240 in LNCS, pages 14–45. Springer, 2004.
3. A. Bergeron, N. Luc, Raffinot M., and J. Risler. Gene teams: a new formalization of gene clusters for comparative genomics. *Computational Biology and Chemistry*, 27(1):59–67, 2003.
4. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proc. of the 9th Int. Conference on Computing and Combinatorics*, number 2697 in LNCS, pages 68–79. Springer, 2003.
5. M. Bernt, D. Merkle, and M. Middendorf. Genome rearrangement based on reversals that preserve conserved intervals. *IEEE Transactions on Bioinformatics*, 2005. Accepted.
6. G. Bourque and P. Pevzner. Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species. *Genome Res.*, 12(1):26–36, 2002.
7. A. Caprara. The reversal median problem. *INFORMS Journal on Computing*, 15(1):93–113, 2003.
8. S. Heber and J. Stoye. Finding all common intervals of k permutations. In *Proc. of the 12th Symposium on Combinatorial Pattern Matching (CPM 2001)*, number 2089 in LNCS, pages 207–218. Springer, 2001.
9. B. Moret and A. Siepel. Finding an optimal inversion median: Experimental results. In *Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI 2001)*, number 2149 in LNCS, pages 189–203. Springer, 2001.
10. B. Moret, S. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp. on Biocomputing (PSB 2001)*. World Scientific Pub.
11. D. Sankoff. Short inversions and conserved gene cluster. *Bioinformatics*, 18(10):1305–1308, 2002.

The Parallel Genetic Algorithm for Designing DNA Randomizations in a Combinatorial Protein Experiment*

Jacek Błażewicz[1,2], Beniamin Dziurdza[1,2],
Wojciech T. Markiewicz[2], and Ceyda Oğuz[3]

[1] Institute of Computing Science, Poznań University of Technology,
Piotrowo 3A, 60-965 Poznań, Poland
[2] Institute of Bioorganic Chemistry, Polish Academy of Sciences,
Noskowskiego 12/14, 61-704 Poznań, Poland
[3] Department of Industrial Engineering, Koç University,
Rumeli Feneri Yolu, 34450 Sariyer, Istanbul, Turkey
bdziurdza@cs.put.poznan.pl

Abstract. Evolutionary methods of protein engineering such as phage display have revolutionized drug design and the means of studying molecular binding. In order to obtain the highest experimental efficiency, the distributions of constructed combinatorial libraries should be carefully adjusted. The presented approach takes into account diversity–completeness trade–off and tries to maximize the number of new amino acid sequences generated in each cycle of the experiment. In the paper, the mathematical model is introduced and the parallel genetic algorithm for the defined optimization problem is described. Its implementation on the SunFire 6800 computer proves a high efficiency of the proposed approach.

1 Introduction

Evolutionary methods of protein engineering such as phage display have revolutionized drug design and the means of studying molecular binding. Usually, they involve repeating cycles of the combinatorial protein experiment. The biochemical experiment's cycle can be divided into three steps [1]: the generation of genetic diversity (randomizing DNA), the coupling of genotype and phenotype, and the identification of successful variants from the obtained protein libraries (the selection).

Protein libraries are often constructed by expressing a library of partially random gene sequences. This approach is used in phage display methods [2]. For instance, this method can be applied for searching new specific peptides or proteins with high affinities to given targets. The cycle of the phage display experiment is showed in Fig. 1, where a new peptide specific to a given acceptor molecule is searched for. Nucleotide distributions used at particular positions

* The research of the first three authors was partially supported by KBN grant No 3T11F00227.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1097–1105, 2006.

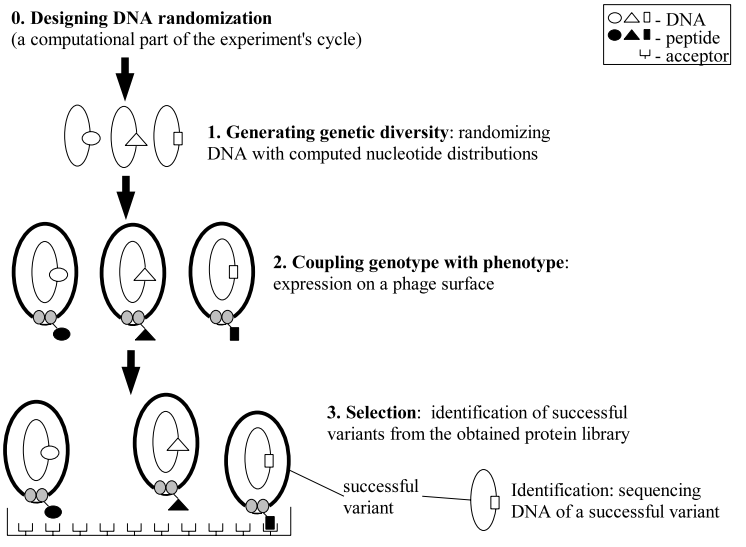


Fig. 1. Phage display experiment's cycle

of a randomized DNA sequence induce a distribution of amino acid sequences obtained in the expression step according to the genetic code. Thus, designing randomizations of DNA libraries is an important computational step of the experiment (step 0).

Various theoretical approaches were proposed to increase the efficiency of evolutionary methods. Many of them assume more or less clearly that "the likelihood of finding variants with improved properties in a given library is maximized when that library is maximally diversified" [3]. The main group of simple methods try to utilize properties of the genetic code like redundancy and degeneracy. In particular, simple equimolar mixtures of all four nucleotides for randomized codons can be replaced by more efficient distributions which are not so biased towards these amino acids with more entries in the genetic code table [1]. Additionally, non-equimolar mixtures can decrease the probability of stop codons, significantly improving the efficiency of experiments [1] [4] by increasing the number of complete sequences. In [4] the size of protein library is quantitavely described as the value of the sequence entropy, which is a measure of the effective number of sequences. Related approaches involve finding the minimal size of the library (the number of clones) that is required to obtain the desired number of distinct variants or all variants with a given probability [3]. Simple measures for library diversity were proposed in [5], where different combinatorial monomer mole fractions were taken into account.

In a typical experiment, the phage display experiment's cycles can be frequently repeated to explore new (not explored yet) parts of the amino acid sequence space. The common case is that one would like to obtain physically as many new (i.e. not obtained previously) peptides as possible in each cycle to

cover the sequence space quicker. Theoretically, it is even possible to obtain all amino acid sequences in the finite number of cycles using randomizations generating partial diversity only. Although the approaches discussed above provide means of assesing and adjusting library diversity and completness for a given library size and applied distributions, they do not take into account the memory of the experiment, i.e. information about the sequences that have been obtained in previous cycles [6][7].

In this article, a new approach is presented which takes into account diversity–completeness trade-off and tries to maximize the number of new amino acid sequences in each cycle of the experiment. The power of the approach is enhanced by the use of genetic algorithms with parallel evaluation of populations that were tested on the SunFire machine. The paper consists of the following sections: in Section 2 necessary phage display experiment details and its mathematical formulation, are presented. The optimized goal function is introduced as well there. In Section 3 the applied genetic algorithm for the defined Optimal Randomization problem is described, together with the parallel evaluation of population's solutions being its essential part. In Section 4 the obtained computational results are presented. Section 5 contains summary and future research directions.

2 Problem Formulation

In the first step of phage display experiment's cycle, the randomized DNA library is constructed by randomizing the selected N codons (not neccessarily consecutive) within available V codons in the open reading frame to achieve genetic diversity. (Consequently, as a result of the expression step, one obtains the peptide library containing amino acid sequences of length V, where N positions are randomized.) During a synthetization of the DNA library, particular nucleotides adenine (A), guanine (G), cytosine (C) and thymine (T) are added to DNA strands with the predetermined ratio $p(n_{i,j})$, $i = 1, \ldots, N$, where $j = 1, 2, 3$ denotes a nucleotide position in the i-th codon and $n = \mathrm{A}, \mathrm{G}, \mathrm{C}, \mathrm{T}$ denotes a nucleotide considered. (Please note that the i-th codon refers to the i-th randomized codon, and not to the i-th codon in a DNA sequence).

Since the probabilities (frequencies) of nucleotide appearances in the given codon $n_{i,1}n_{i,2}n_{i,3}$ are independent, the frequency of this codon can be calculated as follows: $p(n_{i,1}n_{i,2}n_{i,3}) = p(n_{i,1}) \cdot p(n_{i,2}) \cdot p(n_{i,3})$.

During the second step, the library of peptides displayed on a surface of bacteria is obtained. The distribution of amino acid sequences in the library depends on the nucleotide ratios applied during the construction of the DNA library. We assume, that the probability $p(a_i|n_{i,1}, n_{i,2}, n_{i,3})$ that amino acid a_i will occur as encoded by codon $n_{i,1}n_{i,2}n_{i,3}$ is equal to the frequency of $n_{i,1}n_{i,2}n_{i,3}$. To calculate the overall probability of a_i appearance, all codons that encode this amino acid should be considered:

$$p(a_i) = \sum_{n_{i,1}, n_{i,2}, n_{i,3}} p(a_i|n_{i,1}, n_{i,2}, n_{i,3})$$

$$= \sum_{n_{i,1},n_{i,2},n_{i,3}} p\left(n_{i,1}\right) \cdot p\left(n_{i,2}\right) \cdot p\left(n_{i,3}\right) \cdot \delta\left(a_i | n_{i,1}, n_{i,2}, n_{i,3}\right), \quad (1)$$

where:

$$\delta\left(a_i | n_{i,1}, n_{i,2}, n_{i,3}\right) = \begin{cases} 1 & \text{, if } n_{i,1}, n_{i,2}, n_{i,3} \text{ encodes } a_i, \\ 0 & \text{, otherwise.} \end{cases} \quad (2)$$

according to the genetic code.

Assuming that probabilities $p\left(a_1\right), p\left(a_2\right), \ldots p\left(a_N\right)$ are independent, the probability $p\left(s\right)$ that amino acid sequence s will occur with the given amino acids $a_1, a_2, \ldots, a_N$ on randomized codon positions, can be calculated as follows:

$$p\left(s\right) = \prod_{i=1}^{V} p\left(a_i\right) \ . \quad (3)$$

Let L be the DNA library size, i.e. the number of DNA entities (molecules) belonging to the constructed library. The number of such DNA entities (molecules) that each one encodes amino acid sequence s with the given amino acids $a_1, a_2, \ldots, a_N$ on randomized codon positions, can be calculated as follows:

$$E\left(s\right) = L \cdot p\left(s\right) \ . \quad (4)$$

Please note that if the DNA library size L or frequency $p\left(s\right)$ decreases, the number of DNA molecules encoding s decreases too and the chance of expressing s lowers. Thus, in our model, the requirement on the minimal number K of DNA molecules such that each of these molecules encodes s, is introduced to guarantee that s will be physically included into the library obtained and, consequently, s will be compared with other variants during the selection step. Formally, the following condition should be fulfilled:

$$E\left(s\right) = L \cdot p\left(s\right) = L \cdot \prod_{i=1}^{N} p\left(a_i\right) \geq K \ . \quad (5)$$

For given L, K and V and the nucleotide distribution, the real diversity of the expressed peptide library can be defined as the total number of amino acid sequences, where each amino acid sequence has to be encoded by at least K DNA molecules:

$$\text{Diversity} = \left| \left\{ s \in \left(\Sigma_{\text{aa}}\right)^V : L \cdot p\left(s\right) \geq K \right\} \right| \ , \quad (6)$$

where $\left(\Sigma_{\text{aa}}\right)^V$ denotes all possible words (sequences) of length V over alphabet Σ_{aa} of all amino acids. The above defined real diversity formula attempts to calculate the exact number of well presented amino acid sequences.

As we mentioned in Section 1, our approach also tries to take the history of an experiment into account, i.e. information about sequences that have been obtained in previous experiment's cycles. This allows to cover the sequence space quicker by generating as many new complete (not obtained previously) amino

acid sequences as possible. Here, "complete" means that a whole amino acid sequence is encoded only by such DNA sequences that do not contain stop codons in randomized positions. The total number of new amino acid sequences obtained in the current experiment's cycle can be calculated as follows:

$$|\text{NewAminoAcids}| = \left| s \in \left[\left(\Sigma_{\text{aa}}\right)^V \setminus H \right] : L \cdot p\left(s\right) \geq K \right|, \tag{7}$$

where NewAminoAcids is a set of new amino acid sequences and H is a set of previously obtained amino acid sequences. Our goal in "0-th" (computational) step of an experiment cycle is to maximize |NewAminoAcids| value by varying the nucleotide ratios applied during a synthetization of the DNA library.

To formally define the problem, let $p_w\left(n_{i,j}\right)$ be the nucleotide ratio applied in the w-th cycle of the experiment, where $w = 1, 2, \ldots, W$ and W is the number of executed cycles. Consequently, the frequency of each sequence s in w-th cycle is equal to $p_w\left(s\right) = \prod_{i=1}^{V} p_w\left(a_i\right)$. Analogously, L_w and K_w refer to the L and K values applied in the w-th cycle. (Please note that for the current cycle $L = L_{W+1}$ and $K = K_{W+1}$.) Now, one can calculate H as follows:

$$H = \bigcup_{w=1}^{W} \left\{ s \in \left(\Sigma_{\text{aa}}\right)^V : L_w \cdot p_w\left(s\right) \geq K_w \right\}. \tag{8}$$

Note that for every sequence s and its subsequence t consisting of *all randomized positions only*, $p\left(s\right) = p\left(t\right)$, so one can evaluate only the probability of subsequence of length N instead of V in order to compute $p\left(s\right)$. Furthermore, in each of the above definitions one can easily replace the whole sequence s of length V with subsequence t of length N and $\left(\Sigma_{\text{aa}}\right)^V$ with $\left(\Sigma_{\text{aa}}\right)^N$. In particular, |NewAminoAcids| value is always the same regardless of its definition; whether for the whole sequences or for its randomized subsequences. For simplicity, only a randomized subsequence of length N will be considered later on.

The Optimal Randomization problem can be defined as follows:
Optimal Randomization – search version:

Instance: Number $W \in \mathbb{N} \cup \{0\}$ of cycles executed, number N of randomized codons, library size $L_w \in \mathbb{N} \setminus \{0\}$ and minimal number $K_w \in \mathbb{N} \setminus \{0\}$ of DNA molecules encoding one amino acid sequence in the w-th cycle; ratios $p_w\left(n_{i,j}\right) \in [0, 1]$ of nucleotides in particular cycles for every $i \in \{1, 2, \ldots, N\}$, $j \in \{1, 2, 3\}$, $L \in \mathbb{N} \setminus \{0\}$ and $K \in \mathbb{N} \setminus \{0\}$ for the current cycle.

Answer: Randomization ratios $p\left(n_{i,j}\right) \in [0, 1]$ of nucleotides in the current cycle such that there is no other ratios with greater value of |NewAminoAcids|.

For $W > 0$, Optimal Randomization becomes strongly NP–hard [8]. Moreover, there is also not known any polynomial–time algorithm for the nondeterministic Turing machine solving its decision version.

3 The Parallel Genetic Algorithm

Taking into account the strong NP–hardness of the Optimal Randomization problem, there is no hope to find an exact polynomial–time algorithm. Thus, the

(meta)heuristic algorithms for the problem appear as a reasonable alternative to exact ones. Metaheuristic strategies like genetic algorithms [9] allow to search the explored solution space in a smart way and find good suboptimal solutions in an acceptable time.

The presented algorithm for Optimal Randomization uses the classic "simple" genetic algorithm (GA) scheme with non-overlapping populations as described in [9]. The score of a solution (an individual in GA terminology) is defined as |NewAminoAcids|. In each iteration (generation) the algorithm creates a new population of individuals by first selecting individuals from the previous population and then mating these selected individuals to produce the new offspring for the new population. Additionally, the best individual from each generation is carried over to the next generation. The array uniform crossover [10] was applied with a probability of mating 0.9. The mutation probability was 0.01.

A solution of length N is represented as an array genome (in a genetic algorithm sense) consisting of $12 \cdot N$ real numbers $z\left(n_{i,j}\right) \in [0.0, 1.0]$ corresponding to nucleotide ratios:

$$p\left(\mathrm{A}_{i,j}\right) = \frac{z\left(\mathrm{A}_{i,j}\right)}{Z}, p\left(\mathrm{G}_{i,j}\right) = \frac{z\left(\mathrm{G}_{i,j}\right)}{Z}, p\left(\mathrm{C}_{i,j}\right) = \frac{z\left(\mathrm{C}_{i,j}\right)}{Z}, p\left(\mathrm{T}_{i,j}\right) = \frac{z\left(\mathrm{T}_{i,j}\right)}{Z} \in [0,1] \tag{9}$$

where $Z = z\left(\mathrm{A}_{i,j}\right) + z\left(\mathrm{G}_{i,j}\right) + z\left(\mathrm{C}_{i,j}\right) + z\left(\mathrm{T}_{i,j}\right)$. The above encoding guarantees that every genome always represents a correct solution. The initial population is generated randomly in such way that:

$$p\left(n_{i,j}\right) \in \{0,1\} \text{ and } p\left(\mathrm{A}_{i,j}\right) + p\left(\mathrm{G}_{i,j}\right) + p\left(\mathrm{C}_{i,j}\right) + p\left(\mathrm{T}_{i,j}\right) = 1\ . \tag{10}$$

During the evaluation of a population, the scores of all individuals are evaluated. Unfortunately, there is not known any polynomial–time algorithm for computing |NewAminoAcids|. The applied algorithm needs to enumerate $O\left(20^N\right)$ amino acid sequences generated by an individual in order to compute its score value. Thus, the evaluation phase is the most time consuming step of the genetic algorithm for Optimal Randomization. To reduce the evaluation time, the parallel version of evaluation procedure was implemented as explained below.

Our evaluation procedure evaluates scores (|NewAminoAcids|) of population individuals simultanously in a master-slave configuration. The master controls the overall evolution (in the genetic algorithm sense): it initializes a population and then in each iteration of the genetic algorithm it generates a new population and calls the evaluation procedure until the given number of generations is executed. When the population has to be evaluated, the master sends individual's genomes (solutions) to free slaves, a genome to a slave, and waits for the answers. Each slave computes the |NewAminoAcids| of the solution received and then sends the evaluated score to the master and begins to wait for new genomes. The master continues sending genomes until the whole population is evaluated.

4 Results

The implemented genetic algorithm uses the GAlib - a C++ Library of Genetic Algorithm Components [10] and MPICH 1.2.4 [11]. The computational experiments were executed on multiprocessor SunFire 6800 located in Poznan Supercomputing and Networking Center.

The tested instances were obtained by the incremental procedure simulating the execution of experiment's cycles. The solution found for the given instance with W cycles has been added to this instance in order to create a new one with $W + 1$ cycles. The simulation procedure has been executed for initial instances with $K = 10^2$, $L = 10^8$, $W = 0$ and $N = 5, 8, 11, 14$.

Table 1. Results

	$N=5$					$N=8$					$N=11$					$N=14$				
W	P	T	BS	Q	SU	P	T	BS	Q	SU	P	T	BS	Q	SU	P	T	BS	Q	SU
0	17	4.22	356640	3.14	7.44	17	28.94	552960	720.94	10.25	17	27.51	552960	∞	11.01	17	28.02	655360	∞	11.01
0	9	7.44	356640	3.14	4.22	9	52.67	552960	720.94	5.63	9	46.00	552960	∞	6.58	9	45.96	655360	∞	6.71
0	5	12.02	356640	3.14	2.61	5	89.45	552960	720.94	3.32	5	83.80	552960	∞	3.61	5	83.40	655360	∞	3.70
0	3	19.44	356640	3.14	1.61	3	164.71	552960	720.94	1.80	3	155.69	552960	∞	1.95	3	159.69	655360	∞	1.93
0	1	31.38	356640	3.14	1.00	1	296.59	552960	720.94	1.00	1	302.87	552960	∞	1.00	1	308.48	655360	∞	1.00
10	17	3.96	114319	2.68	7.50	17	36.31	689144	899.67	9.86	17	40.85	774144	∞	10.68	17	46.11	589824	∞	10.71
10	9	6.98	114319	2.68	4.26	9	65.87	689144	899.67	5.44	9	65.31	774144	∞	6.68	9	74.36	589824	∞	6.64
10	5	11.30	114319	2.68	2.63	5	111.27	689144	899.67	3.22	5	117.86	774144	∞	3.70	5	133.23	589824	∞	3.71
10	3	18.42	114319	2.68	1.61	3	200.64	689144	899.67	1.78	3	221.32	774144	∞	1.97	3	251.57	589824	∞	1.96
10	1	29.70	114319	2.68	1.00	1	358.03	689144	899.67	1.00	1	436.32	774144	∞	1.00	1	493.86	589824	∞	1.00
25	17	5.36	66056	4.47	7.74	17	40.17	691392	902.60	9.92	17	46.76	688128	∞	10.39	17	44.56	514048	∞	9.50
25	9	9.49	66056	4.47	4.38	9	72.91	691392	902.60	5.47	9	74.68	688128	∞	6.50	9	67.52	514048	∞	6.27
25	5	15.44	66056	4.47	2.69	5	123.27	691392	902.60	3.23	5	134.04	688128	∞	3.62	5	117.68	514048	∞	3.60
25	3	25.11	66056	4.47	1.65	3	223.33	691392	902.60	1.78	3	252.33	688128	∞	1.93	3	221.73	514048	∞	1.91
25	1	41.51	66056	4.47	1.00	1	398.49	691392	902.60	1.00	1	485.75	688128	∞	1.00	1	423.33	514048	∞	1.00

To assess quality of the genetic algorithm, a simple *random algorithm*, generating solutions randomly, has been implemented. The algorithm generates random genotypes (see (9)) and evaluates their values in the loop which is terminated if the given time limit is exceeded. The time limit was set to the three maximal execution times of the genetic algorithm. The results of computational experiments for the chosen instances are presented in Table 1. Each entry in the table is the mean value for 3 runs of the program for varying seed values of a pseudorandom number generator that determines the behavior of the crossover and mutation operators. The number of generations (iterations) of the genetic algorithm was set to 100 and the number of individuals in a population was set to 64. The columns in Table 1 contain respectively: N - the number of randomized amino acid positions, W - the number of executed experiment's cycles, P - the number of processors utilized, T - execution time in seconds,

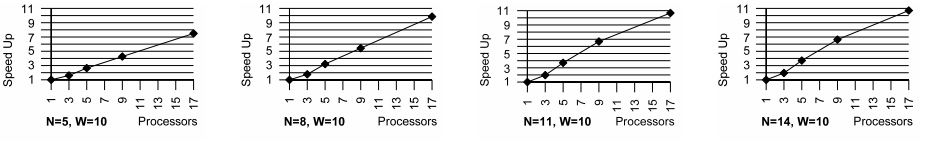


Fig. 2. Speed up charts

BS - the best score found, Q - the "quality" measure of the genetic algorithm computed as the ratio beetween the BS and the score value obtained by the random algorithm described above, and SU - the speed up. ∞ in Q column indicates that the random algorithm found solution with score equal to 0. The values in Q column show that solutions found by the genetic algorithm are significantly better than the ones found by the random algorithm. The obtained speed up values prove the high efficiency of the applied approach even though only the evaluation procedure is executed in parallel. The dependence beetwen speed up and the number of processors is almost perfectly linear for a wide range of problem's parameters (see Fig. 2).

5 Conclusions

In this paper, a new approach is presented which takes into account diversity-completness trade-off and tries to maximize the number of new amino acid sequences in each cycle of the phage display experiment. The corresponding mathematical model was defined and new measures Diversity and |NewAminoAcids| for assesing efficiency of biochemical experiment's libraries were introduced. For the defined Optimal Randomization problem, the genetic algorithm with evaluation procedure working in a master-slave configuration, was designed, implemented and tested on SunFire 6800 computer. Computational experiments proved the high efficiency of the approach proposed.

In the future more sophisiticated operators should be designed to improve the behavior of the genetic algorithm for hard instances. Alternatively, other genetic algorithm schemes combined with, for example, local search strategies, could be implemented.

References

1. B. Steipe (1999) Evolutionary Approaches to Protein Engineering. Steipe, B. (1999) Evolutionary Approaches to Protein Engineering. *Current Topics in Microbiology and Immunology: Combinatorial Chemistry in Biology (M. Famulok, E.-L. Winnacker, eds.)*, **243**, 56-81
2. Smith, G.P., Petrenko V.A. (1997) Phage Display. *Chem. Rev.*, **97**, 391-410
3. Patrick W.M., Firth A.E., Blackburn J.M. (2003) User-friendly algorithms for estimating completeness and diversity in randomized protein-encoding libraries. *Protein Eng*, **16**, 451-457
4. Wang, W., Saven, J. G. (2002). Wang, W., Saven, J. G. (2002). Designing gene libraries from protein profiles for combinatorial protein experiments. *Nucleic Acids Res.*, **30**, e120
5. Brian Ward and Thomas Juehne (1998) Combinatorial library diversity: probability assessment of library populations *Nucleic Acids Research*, **26**, 879-886
6. Dziurdza, B., Błażewicz J., Markiewicz W. T. (2003) Artificial evolution. Algorithms for searching for amino acid sequences. *RECOMB 2003. Currents in Computational Molecular Biology*, 151-152

7. Błażewicz J., Dziurdza, B., , Markiewicz W. T. (2003) Artificial evolution. An algorithm for an exploration of the amino acid sequences space. *Foundations of Computing and Decision Sciences*, **28**, 195-209
8. Dziurdza, B. Błażewicz J., Markiewicz W. T. (2004). Working papers in Institute of Computing Science, Poznan University of Technology
9. Goldberg, D. E. (1989) Genetic Algorithms in Search and Optimization, *Addison-Wesley Pub. Co., 1989*
10. Wall, M. (2005) GAlib C++ library. *http://lancet.mit.edu/ga/*
11. MPICH, free Message Passing Interface library, *http://www-unix.mcs.anl.gov/mpi/mpich/*

Introducing Dependencies into Alignment Analysis and Its Use for Local Structure Prediction in Proteins

Szymon Nowakowski[1], Krzysztof Fidelis[2], and Jerzy Tiuryn[1]

[1] Institute of Informatics, Warsaw University,
Banacha 2, 02-097 Warszawa, Poland
[2] Genome Center, University of California, Davis,
Genome and Biomedical Sciences Facility,
451 East Health Sciences Drive, Davis, CA 95616, USA
s.nowakowski@mimuw.edu.pl

Abstract. In this paper we explore several techniques of analysing sequence alignments. Their main idea is to generalize an alignment by means of a probability distribution. The *Dirichlet mixture method* is used as a reference to assess new techniques. They are compared based on a cross validation test with both synthetic and real data: we use them to identify sequence-structure relationships between target protein and possible local motifs. We show that the *Beta method* is almost as successful as the reference method, but it is much faster (up to 17 times). *MAP (Maximum a Posteriori) estimation for two PSSMs (Position Specific Score Matrices)* introduces dependencies between columns of an alignment. It is shown in our experiments to be much more successful than the reference method, but it is very computationally expensive. To this end we developed its parallel implementation.

1 Introduction

Motif discovery in the case of DNA or protein sequences has a wide range of applications in modern molecular biology: from modeling mechanisms of transcriptional regulation [4, 11] to prediction of protein structure [2]. In this work we address the latter problem. We developed a probabilistic approach of generalizing sequence alignments representing structural motifs of local neighborhoods in proteins. We show that we can identify a correct motif of a given protein fragment accurately using only sequence of the query fragment.

This problem has been extensively studied during the last few years. The simplest estimator of seeing an amino acid in an alignment, called *Maximum Likelihood* estimator [8], which only counts occurrences of amino acids, is of no use for alignments consisting of too few sequences, as it may happen that an amino acid *is not seen* in the alignment at all, but it *would be seen*, if the number of aligned sequences were greater. To this end various methods introducing prior knowledge were proposed. These methods range from the *zero-offset method* [10], through methods based on substitution matrices [3] or amino acid

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1106–1113, 2006.

feature alphabets [13], to the most advanced *Dirichlet mixture method* [6, 12] for which a mixture of Dirichlet distributions is supplied as prior knowledge. The *pseudocount method* [14] is a special kind of the Dirichlet mixture method, where only one Dirichlet distribution is used. It is shown in [10] that, when the columns of an alignment are independent, the Dirichlet mixture method is close to the theoretical optimum.

Modeling dependencies between columns in DNA sequence alignments has been recently studied in [1, 4, 7]. In [7] dependencies are modeled only between adjacent columns with the use of Hidden Markov Models. Methods exist to model dependencies between columns not adjacent in an alignment. Autors of [4] analyse a number of such methods, one of them being the *mixture of PSSMs (Position Specific Score Matrices)* method. They use the pseudocount method to introduce prior knowledge (modeled by a single Dirichlet distribution). We propose a new method of estimating distributions from alignments that is more suitable for protein sequence alignments. It models column dependencies with a mixture of PSSMs and uses a mixture of many Dirichlet distributions as prior knowledge. We discuss advantages of this choice in Section 5.

We performed two experiments comparing different techniques of generalizing alignments. The first one was performed with synthetic data, generated from known probability distributions, the second with real data from a database of protein motifs. In these experiments we compared our two new techniques with the Dirichlet mixture method. First of the techniques we propose is *MAP (Maximum a Posteriori) estimation for two PSSMs*. This technique can model column dependencies. We show that it gives much better results than the reference method. Second, the *Beta method*, gives results comparable with the Dirichlet mixture method, but is up to 17 times faster.

2 Methods

Results of our experiments were obtained by a cross validation test done with both synthetic and real data. We performed 3-fold cross validation with synthetic data and 5-fold cross validation with real data. Let us describe the *F-fold cross validation test*, where F is a positive integer, usually between 3 and 10. The dataset for the test consists of a number of alignments. The test itself is performed as follows: sequences from every alignment are randomly divided into F subsets. Each of them is treated as a test set T in one of F runs of the cross validation test. At the same time the remaining $F - 1$ subsets are treated as a training set (profiles are estimated from the union of $F - 1$ subsets, according to a selected profile estimation method, see below for details).

The log-probability is computed for every sequence from T and the alignment from which that sequence was removed. For each alignment we then compute its mean log-probability value for a given estimation method, averaged over all sequences in all runs of the cross validation procedure. Let us fix the alignment $\mathcal{A}$. To address the statistical significance of the difference in mean log-probability values for $\mathcal{A}$ between two methods, the paired t-test is performed. Following [4],

we call one method *better* than the other on $\mathcal{A}$ when the difference in mean log-probability values for $\mathcal{A}$ is positive, and *significantly better* when the associated paired t-test p-value is below the 0.05 treshold. It is called *worse* or *significantly worse* when the other method is, respectively, better or significantly better.

Additionally, in the case of synthetic data, which consists of alignments with the same number of columns, during the cross validation procedure the prediction test is performed as follows: for a given sequence from T, every alignment is scored as described below. The prediction is judged as successful if the correct alignment (i.e. the one, from which the test sequence was removed) is the one with the highest score.

Let us define a notion of PSSM. By PSSM we understand a sequence profile. Formally, *PSSM* of length l is a matrix (p_{ij}) of size $20 \times l$. We estimate PSSMs from alignments using one of the techniques described below. All those techniques accept a mixture of Dirichlet distributions as prior knowledge.

1. *Dirichlet mixture method.* This method, described in detail in [6, 12], is used as a reference method. In short, PME (Posterior Mean Estimator) is used with a prior being a mixture of Dirichlet distributions.
2. *Beta method.* Columns of all sequence alignments are first clustered by the similarity of their amino acid distributions. The clusters are built around center points computed from the supplied prior. After construction of the clusters the prior is discarded and new and much simpler priors are estimated for every cluster. Let us refer to a profile obtained from a column with the use of the Maximum Likelihood estimator as an ML-profile. For every amino acid x in the i-th cluster the new prior is the $Beta(a^i_x, b^i_x)$ distribution. The values of a^i_x, b^i_x are estimated from all ML-profiles in the i-th cluster.

   Let us fix a column c in an alignment. Suppose that c is in the i-th cluster. Let us consider any amino acid x. Suppose that x has the observed frequency $\frac{n_x}{n}$ in the ML-profile of c. A posterior estimator of the probability of x appearing in c is $\frac{n_x+a^i_x}{n+a^i_x+b^i_x}$. We repeat that for all 20 amino acids. After normalization by $N = \sum_x \frac{n_x+a^i_x}{n+a^i_x+b^i_x}$ we obtain a new profile for c.
3. *MAP estimation for two PSSMs.* Instead of PME estimation (as in the case of the Dirichlet mixture method) we use MAP estimation to obtain a mixture of two PSSMs together with the weights q_1, q_2 of these PSSMs. This representation takes into account column dependencies.

   The implementation is parallel since the estimation is computationally very demanding. MPI environment is used. We use a master-slave paradigm in which one of the processes distributes the tasks and collects the results, while the rest of the processes perform calculations. The calculations are done in two phases. In the first, called the *search phase*, an extensive sampling of the probability space is done to find good starting points for the second phase, which is called the *maximizing phase*. In the second phase a gradient descent as a part of EM (Expectation Maximization) algorithm is performed to locally maximize our goal function.

We use the following scoring procedure. Let us fix an alignment $\mathcal{A}$ of length l and a query sequence $S = s_1 s_2 s_3 \ldots s_l$. $\mathcal{A}$ is described by $P^{(1)}, \ldots, P^{(K)}$, PSSMs estimated according to one of the techniques described above, and by K positive weights of these PSSMs, $q_1, \ldots, q_K$, such that $\sum_{i=1}^{K} q_i = 1$. The value of K depends on the estimation technique and it equals 1 or 2 in this work. Every PSSM $P^{(k)} = (p_{ij}^{(k)})$ is a matrix of size $20 \times l$. The score of $\mathcal{A}$ is $\mathcal{M}(S, \mathcal{A}) = \sum_{k=1}^{K} q_k \cdot p_{s_1 1}^{(k)} \cdot p_{s_2 2}^{(k)} \cdot \ldots \cdot p_{s_l l}^{(k)}$. Logarithm of this score is used as the log-probability value for the alignment $\mathcal{A}$ in the cross validation procedure.

3 Experiment with Synthetic Data

We performed two tests with synthetic data: in the first one, alignments consisted of 200 sequences and in the second test of 600 sequences. This allowed us to assess the impact of the number of sequences in an alignment on the estimation accuracy.

For both tests we used the same prior distribution as in the case of the real data (i.e. a mixture of 54 Dirichlet distributions). We believe that it made our synthetic experiment more realistic. We generated 300 alignments of length 30 for both tests. The procedure to generate a single alignment was as follows: first, generate three PSSMs with 30 columns distributed according to the prior; then, from this mixture of PSSMs, generate 200 or 600 (depending on the test) sequences of length 30. This procedure was repeated 300 times to generate 300 alignments.

3-fold cross validation was performed as it was described in Section 2. Thus the estimation was done in fact on $\frac{2}{3}$ of the alignment. For the estimation we used the same prior that had been used to generate alignments.

In this experiment we tested the MAP estimation for two PSSMs and the Beta method. Their comparison to the reference method, the Dirichlet mixture method, presented in Table 1, shows the number of alignments, for which each method had higher mean log-probability value than that of the reference method. Table 1 also shows the percentage of successful predictions (evaluated as described in Section 2) depending on the profile estimation method used.

As seen in Table 1, for alignments comprising relatively small number of sequences the difference in prediction accuracy between the Beta method and the Dirichlet mixture method was larger. This is due to a much simpler character of the Beta method. The Beta method also had significantly worse mean log-probability values in almost all cases. In addition we can see that in the case of insufficient data the MAP estimation for two PSSMs performed very poorly. It had lower mean log-probability values in all cases and in the prediction accuracy test it was also much worse than both methods without dependencies. This is caused by a need to estimate a larger number of parameters.

However, for alignments with greater number of aligned sequences the Beta method performed almost as well as the Dirichlet mixture method (when we consider prediction accuracy) and MAP estimation for two PSSMs proved to be much more successful, having both much better prediction accuracy and higher

Table 1. Results of tests with synthetic data: the number of alignments with higher mean log-probability value than that of the reference method (i.e. better results), the number of significantly better results and the number of significantly worse results. The reference method row is presented in bold face. Column 6 shows the percentage of successful predictions. Columns 7 and 8 show the number of sequences in each alignment and an average number of sequences in the training set used for estimation in the 3-fold cross validation procedure.

Estimation method	PSSMs	Better	Sig. better	Sig. worse	Succ. pred.	Seqs	Tr. seqs
Dirichlet mixture	**1**	**0**	**0**	**0**	**18%**	**200**	**133**
Beta	1	0	0	293	16%	200	133
MAP estimation	2	0	0	300	11%	200	133
Dirichlet mixture	**1**	**0**	**0**	**0**	**22%**	**600**	**400**
Beta	1	63	5	128	22%	600	400
MAP estimation	2	285	274	7	29%	600	400

mean log-probability values in almost all cases, most of them being significantly better.

We can see that it is enough to estimate *two* PSSMs instead of one to greatly increase success rate, in spite of the fact, that data was generated from a mixture of *three* PSSMs.

4 Experiment with Real Protein Motifs

Our experiment was performed on a list of 117 sets of structurally aligned protein fragments. They were taken from proteins represented in ASTRAL 1.63 [5] and having less than 40% sequence identity to one another. The fragments were composed of several contiguous subfragments forming a local 3D neighborhood in a protein core, as described in [9].

Fragments were structurally aligned based on their 3D structure similarity measured by RMSD (Root Mean Square Deviation). After the structural alignment had been constructed, a corresponding sequence alignment was obtained

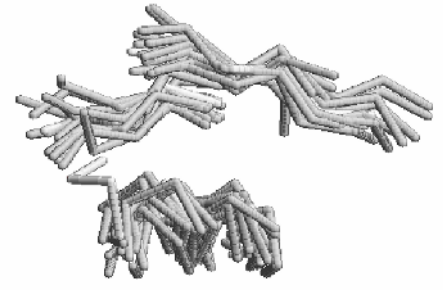


Fig. 1. Example of a set of structurally aligned fragments. There are 3 contiguous subfragments in every fragment in this set.

for each set of fragments. Figure 1 shows an example of such a structurally aligned set. The sets (and consequently alignments) contained from 508 to 6996 fragments (sequences), as we rejected alignments with less than 500 sequences (too few sequences to perform a reliable MAP estimation for two PSSMs).

To gather more sequential information for estimation methods, sequences of the subfragments were extended by 4 amino acids on both ends. Extensions in the set need not share common structure, as is the case for the original fragments.

Table 2 summarizes the results: it shows the number of alignments for which the considered method was better (significantly better, significantly worse) than the reference method (evaluated as decribed in Section 2) in the 5-fold cross validation procedure with a mixture of 54 Dirichlet distributions used as a prior.

Table 2. Results of the test: the number of alignments with higher mean log-probability value as compared to the reference method (i.e. better results), the number of significantly better results and the number of significantly worse results. The reference method row is presented in bold face. Last column shows computation time on a single processor on all data (without cross validation, averaged over 3 runs).

Estimation method	PSSMs	Better	Sig. better	Sig. worse	Comp. time
Dirichlet mixture	**1**	**0**	**0**	**0**	**34 sec.**
Beta	1	64	32	20	2 sec.
MAP estimation	2	83	77	26	no data

As seen in Table 2, the Beta method was more succesfull than the Dirichlet mixture method in 64 of 117 cases (but not significantly in half of them), while the estimation time in the former method was much shorter than in the case of the latter. To assess the speed of estimation, we performed an additional test (repeated 3 times) in which no cross validation was performed and both methods were used to estimate profiles from the data. The test was performed on a computer with Pentium4 2.80GHz processor and 512MB RAM. The estimation took 2 sec., 2 sec. and 2 sec. for the Beta method and 34 sec., 32 sec. and 37 sec. for the Dirichlet mixture method; on average the Beta method was 17 times faster.

Including dependencies in our method with two PSSMs greatly increased accuracy of estimation, making it better in 83 of 117 cases, most of them being significantly better. It was significantly worse only in 26 cases.

MAP estimation, although the most successful, is also computationally most demanding: the test described took about a week of computation time on a cluster of 16 two-processor computers (AMD Opteron 2GHz, 2GB RAM). To assess the impact of the number of processes on the computation time, additional test was performed on that cluster. Table 3 summarizes the results of running the computations without cross validation on a subset of alignments which consisted of less than 1000 sequences (48 such alignments were present among all 117 alignments under consideration). The computations were run with $2, 4, 8, 16$ and 32 processes. One of these processes, as described in Section 2, distributed the tasks and collected the results, the rest of the processes performed the calculations.

Table 3. Results of the scaling test for the MAP estimations for two PSSMs: the computation time as a function of the number of processes. The second column presents the number of processes doing the calculations. Columns 3 and 4 present the real computation time while two last columns present the computation time predicted by dividing the times from the row in bold face by the number of processes performing the calculations. The values agree very well.

Processes	Calc. processes	Computation time		Predicted time	
		Search phase	Max. phase	Search phase	Max. phase
2	**1**	**88704 sec.**	**961640 sec.**	**-**	**-**
4	3	29568 sec.	332843 sec.	29568 sec.	320547 sec.
8	7	12833 sec.	130843 sec.	12672 sec.	137377 sec.
16	15	5989 sec.	63409 sec.	5914 sec.	64109 sec.
32	31	2905 sec.	34648 sec.	2861 sec.	31021 sec.

There were, respectively, $1, 3, 7, 15$ and 31 processes doing the calculations. As seen in Table 3, the computations scale very well. The computation time in each case can be well predicted by dividing the time taken by one process doing the calculations by the number of calculating processes.

5 Conclusions

The success of MAP estimation for two PSSMs not only in the experiment with synthetic data, but also with real data, is caused, we believe, by three factors:

1. The way to include column dependencies, we introduce in our model (a mixture of PSSMs), has a strong biological background: it models the fact, that the structure of a protein motif can be stabilized in more than one way.
2. The dependent positions, which make the stabilization possible, are located on different subfragments, not adjacent in an alignment. When the models with dependencies between nonadjacent columns are considered, mixtures of PSSMs have relatively few parameters [4], which makes estimation more reliable.
3. The prior knowledge we use in our model (a mixture of many Dirichlet distributions) makes it possible to model many amino acid contexts in columns, in contrast to the pseudocount method used in [4], which models only one context: the background. Thus we can model hydrophobic columns, columns with big amino acids, columns with positive amino acids, etc.

Comparison of the results on synthetic and real data shows that when performing estimation with dependencies it is very important to include only alignments with enough sequences to make the estimation reliable. The results can be very poor when the dependencies are considered but not enough examples are provided for the estimation procedure.

Acknowledgments

This work was supported by Polish KBN grant 3 T11F 006 27. Our results were obtained with the use of computer resources of ICM Warsaw University, Poland.

References

1. Agarwal, P., Bafna, V.: Detecting Non-adjoining Correlations with Signals in DNA. in RECOMB'98 (1998) 2–8
2. Aloy, P., Stark, A., Hadley, C., Russell, R.B.: Predictions Without Templates: New Folds, Secondary Structure, and Contacts in CASP5. *Proteins: Struct. Funct. Genet.* **53** (2003) 436–456
3. Altschul, S.F.: Amino Acid Substitution Matrices from an Information Theoretic Perspective. *JMB* **219** (1991) 555–565
4. Barash, Y., Elidan, G., Friedman, N, Kaplan, T,: Modeling Dependencies in Protein-DNA Binding Sites, in RECOMB'03 (2003) 28–37
5. Brenner, S.E., Koehl P., Levitt M.: The ASTRAL Compendium for Sequence and Structure Analysis. *Nucleic Acids Research* **28** (2000) 254–256
6. Brown M. P., Hughey, R., Krogh, A., Mian, I. S., Sjölander, K., Haussler, D.: Using Dirichlet Mixture Priors to Derive Hidden Markov Models for Protein Families. In: Hunter, L., Searls, D., Shavlik J. (eds.) *ISMB-93*, Menlo Park, CA: AAAI/MIT Press. (1993) 47–55
7. Bulyk, M.L., Johnson, P.L., Church, G.M.: Nucleotides of Transcription Factor Binding Sites Exert Interdependent Effects On the Binding Affinities of Transcription Factors, *Nuc. Acids Res.*, **30** (2002) 1255–1261
8. Durbin, R., Eddy, S., Krogh, A, Mitchison, G.: Biological Sequence Analysis. Cambridge University Press (1998)
9. Hvidsten, R.H., Kryshtafovych, A., Komorowski, J., Fidelis, K.: A Novel Approach to Fold Recognition Using Sequence-Derived Properties From Sets of Structurally Similar Local Fragments of Proteins, *Bioinformatics*, **19** (2003) 81–91
10. Karplus, K.: Regularizers for Estimating Distributions of Amino Acids from Small Samples. Technical Report UCSC-CRL-95-11, University of California, Santa Cruz, CA, USA (1995), `ftp://ftp.cse.ucsc.edu/pub/tr/ucsc-crl-95-11.ps.Z`
11. Liu, X., Brutlag, D.L., Liu, J.S.: Bioprospector: Discovering Conserved DNA Motifs in Upstream Regulatory Regions of Co-expressed Genes. In *PSB'01* (2001)
12. Sjölander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I. S., Haussler, D.: Dirichlet Mixtures: a Method for Improved Detection of Weak but Significant Protein Sequence Homology, *Computer Applications in Biosciences* **12** (1996) 327–345
13. Smith, R.F., Smith, T.F.: Automatic Generation of Primary Sequence Patterns from Sets of Related Protein Sequences. *PNAS* **87** (1990) 118-122
14. Tatusov, R.L., Altschul, S.F., Koonin, E.V.: Detection of Conserved Segments in Proteins: Iterative Scanning of Sequence Databases with Alignment Blocks. *PNAS* **91** (1994) 12091-12095

Parallel Implementation of Logical Analysis of Data (LAD) for Discriminatory Analysis of Protein Mass Spectrometry Data

Krzysztof Puszyński

Silesian University of Technology

Abstract. A parallel implementation of proteomic ovarian cancer diagnosis system based on logical analysis of data is shown. The implementation is based on computational cluster elaborated in System Engineering Group at Silesian University of Technology. For verification of algorithm and software Ovarian Dataset 8-7-02 (which can be found at http://clinicalproteomics.steem.com) was used. This mass spectrometry data contains intensity levels of 15 154 peptides defined by their mass/charge ratios (m/z) in serum of 162 ovarian cancer and 91 control cases. A Seti-like and OpenMosix with PVM cluster technology was used to construct in LAD a fully reproducible models (1) using full range and (2) using only 700-12000 of m/z values of peptides and proved in multiple cross-validation leave-one-out tests to guarantee sensitivities and specificities of up to 100 %.

1 Introduction

The early detection of cancer is still one of the primary goals of cancer management. Since identification of peptides involved in tumorigenesis becomes possible the proteomic analysis of cancer is in the forefront of modern biomedical research. There are many computational methods applicable for analysis of proteomic data. Among them we mention clustering [1], discriminant analysis [1], Bayesian networks [1], decision trees [2], support vector machines [3], artificial neural networks [4], but as was shown in [5] the additional important results can be obtained by using the logical analysis of data (LAD) [6, 7]. Nevertheless, LAD methodology requires large number of calculations because of their specificity. It is impossible to implement this on a single desktop computer even with large RAM and very fast CPU. Approximate time for one iteration in leave-one-out validation on PIV 3,2GHz machine with 1 GB RAM is about 96 hour, because we have 253 samples and we need 24 288 hour for execution of complete validation, and we have no warranty that we receive correct ovarian cancer model. For this reason it becomes necessary to use of parallel calculations. At the begining a Seti-like cluster was build on eight System Engineering Group laboratory halls computers. When computational cluster with OpenMosix and PVM technology was started the computation became transferred on him.

R. Wyrzykowski et al. (Eds.): PPAM 2005, LNCS 3911, pp. 1114–1121, 2006.

2 Dataset Description and Methodology of LAD

2.1 Dataset

The data used in this paper are (as in [5]) Ovarian Dataset 8-7-02 which can be found at http://clinicalproteomics.steem.com. They include intensity levels of 15 154 peptides defined by their mass/charge ratios (m/z) in serum of 162 ovarian cancer and 91 control cases. This data has been obtained with SELDI-TOF-MS (Surface-Enhanced Laser Desorption/Ionization with Time-Of-Flight Mass Spectrometry) using WCX2 chip (weak cation exchange array with carboxylate funcionality protein chip array).

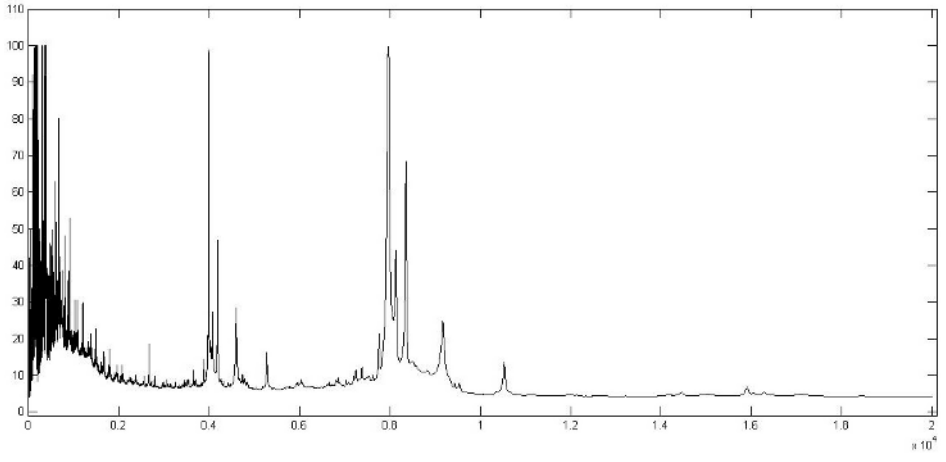


Fig. 1. SELDI-TOF-MS chromatogram with ion peaks in range of 0 20 000 Da. Notice that there are more m/z peaks between 0 and 10 000 Da than between 10 000 and 20 000 Da. On most cases accessible dataset contains range of 700 12 000 Da.

2.2 Methodology of LAD

The detailed description of methodology of LAD is beyond the scope of this paper. It can be found in [6, 7]. In this paper only short description of LAD is given. Logical analysis of data consists of 4 phases:

- binarization - because LAD works in boolean (0 and 1) space we need to convert our problem to this space. In this phase we find cut-points and then binarize our data according to them,
- support set finding - in this phase we build and solve a system of inequalities based on binary information of peptides intensity. As a result we receive a support set - a minimal set of peptides and its cut-points which are necessary to correct distinguishing ovarian cancer cases,

- pattern building - using peptides contained in support set we build positive (including only cancer cases) and negative (including only control cases) patterns. As a result of this phase we have large number of positive and negative patterns,
- model building - in this final phase we reduce number of patterns to receive ovarian cancer model,

The first two phases (binarization and support set finding) require a large computational power and therefore the application of the cluster is necessary.

2.2.1 Binarization

At the beginning of binarization phase we need to find a cut-points which are values of intensity levels between next cases counted according to formula:

$$cut - point(j) = intensity(i) + \frac{intensity(i+1) - intensity(i)}{2} \tag{1}$$

separately for each peptide. Then we carry out the proper binarization according to the rule:

$$\forall(i,j)\ if\ intensity(i) \geq cut - point(j)\ then\ b_{ij} = 1\ else\ b_{ij} = 0 \tag{2}$$

By making this for all peptides we receive a binary matrix B from about 3.5 million columns and 253 rows, where each column represents one cut-point and each row represents one (positive P or negative N) cases.

Table 1. Example of binary matrix B received in binarization phase of LAD. Each column represents one cut-point and each row represents one case. Notice that we have two types of cases: positive cases P (ovarian cancer cases) and negative cases N (control cases).

	cp1	cp2	cp3	...	cp(k)	...	cp(K)
Intensity P1	0	0	0	...	0	...	0
Intensity P2	1	0	0	...	0	...	0
Intensity Pi	1	1	1	...	0	...	0
...	...	...	...	...	...	...	...
Intensity PI	1	1	1	...	1	...	0
Intensity N1	1	1	0	...	0	...	0
Intensity Nj	1	1	1	...	0	...	0
...	...	...	...	...	...	...	...
Intensity NJ	1	1	1	...	1	...	1

2.2.2 Support Set Finding

In this phase we use binary matrix B received from binarization phase. For each pair of Pi and Nj we want to find this cp(k) which is different in Pi and Nj because this cp(k) discriminate cases Pi and Nj. We can say that to distinguish cases Pi and Nj we need at least one of this cp(k). This leads us to the following condition:

$$\forall(i,j,k)\ if\ b_{i,k} \neq b_{j,k}\ then\ a_{(i,j),k} = 1\ else\ a_{(i,j),k} = 0;\ b_{i,k} \in P \wedge b_{j,k} \in N \quad (3)$$

and finally to the following system of inequalities:

$$\begin{matrix}(1,1)\\(1,2)\\\dots\\(1,J)\\\dots\\(I,J)\end{matrix}\begin{pmatrix}1 & 1 & 0 & \dots & a_{(1,1),k} & \dots & 0\\1 & 1 & 1 & \dots & a_{(1,2),k} & \dots & 0\\\dots & \dots & \dots & \dots & \dots & \dots & \dots\\1 & 1 & 1 & \dots & a_{(1,J),k} & \dots & 1\\\dots & \dots & \dots & \dots & \dots & \dots & \dots\\0 & 0 & 0 & \dots & a_{(I,J),k} & \dots & 1\end{pmatrix}\begin{bmatrix}y_1\\y_2\\y_3\\\dots\\y_k\\\dots\\y_K\end{bmatrix} \geq \begin{bmatrix}1\\1\\1\\\dots\\1\\\dots\\1\end{bmatrix} \quad (4)$$

where we have I*J=162*91=14742 inequalities and about K=3.5 millions of variables. Because we want to have the smallest possible set of y=1 we treat this problem as a zero-one programming problem where minimalized objective function is:

$$F(x) = \sum_{k=1}^{K} y_k \quad (5)$$

and system of inequalities is a system of constraints. We solve this by modified Balas Algorithm and receive irredundant set of y's which are necessary to distinguish P and N cases. We call this set a support set.

2.3 Computational Effort

Because of its specificity proteomic data like microarray data need a special approach to correct cross validation tests. On this data we have a large number of proteins defined by their m/z charge values (15 154 peptides) and few number of cases (253 cases). In this case we need to repeat all procedure for each single case which we leave out (detailed explanation can be found at [8] and [9]). For this reason a full computation on single computer is impossible and we need to use a parallel computation. This was made by using a Seti-like cluster build on eight computers in the student lab then was transferred to the computational cluster with OpenMosix and PVM technology.

3 Parallel Implementation of LAD

3.1 Seti-Like Cluster

This kind of parallel computing has been well-known for quite a long time. It was invented on Berkeley University, and is used to analyze radio signals from space. Its idea is to divide calculations on many independent samples and to compute separately on different computers. Then results are sent to central computer. In my implementation a computation problem was divided on single units with one peptide being one unit. The first one and a half of second phase (inequalities building) were executed simultaneously. Seti-like cluster built on eight System Engineering Group laboratory halls computers and one personal computer as a

central cluster computer was applied. At the beginning a test unit was sent to all computers for its performance measuring so we can send a correct number of units to finish the computations in one, possibly the shortest time. Then the units was sent and computations begin. At the end results was sent back to the central computer where the rest of calculations was made. The use at Seti-like cluster permitted to accelerate calculations about six times.

3.2 Computational Cluster with OpenMosix and PVM Technology

Computational cluster with OpenMosix and PVM technology was elaborated in Research Laboratory of Computational Biology at System Engineering Group at Silesian University of Technology. In the laboratory we have four high-end servers which compose in total of 8 CPUs (dual processor systems based on Intel Xeon HyperThreading architecture), 16 GB of RAM and about 1,4 TB of disc space. The whole system is connected by low-latency local network (1 Gbit ethernet) to create High Performance Computing cluster, named Beowulf cluster one unified system to parallel computing. This cluster is based on Linux platform with OpenMosix kernel and software package based on message-passing paradigm (distributed memory virtual computers such as PVM Parallel Virtual Machine or MPI - Message Passing Interface). Users can create and execute programs written in C,C++ or FORTRAN programming language or can take the advantage of high developed scientific environments like Matlab, R or others. In this cluster, LAD calculations was distributed on all processors using PVM and OpenMosix technology.

Application of the computational cluster with OpenMosix and PVM technology permitted to accelerate calculations about ten times.

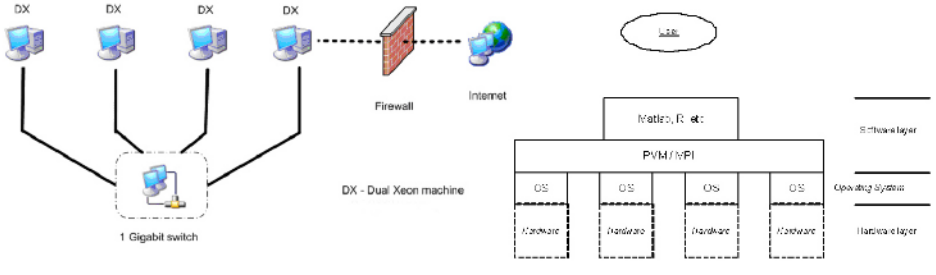


Fig. 2. Hardware and software computational cluster elaborated in Research Laboratory of Computational Biology at System Engineering Group architecture

4 Results

In this section two accurate models of ovarian cancer diagnostic system consisting on logical patterns are shown. The first contained peptides with m/z charge range of 0 – 20000 Da, and second with the most accessible dataset range of 700 – 12000 Da. Both have a sensitivity and specificity in multiple cross validation leave-one-out tests equal to 100%. In the tables below next columns contain: positive (P)

or negative (N) pattern number, cut-points values of peptides choosen to the support set and suitable relations on them describing given pattern and in the last column prelevance of this pattern. Notice (1) that for correct distinguishing ovarian cancer cases from control cases we need at least 3 peptides in the support set when we have a full range of m/z values (0 – 20000 Da) and at least 8 peptides when we have the most accessible range in dataset (700 - 12000 Da). Notice (2) the both models contained peptides with rather low m/z charge value (lower than the half of the maximum value). This indicates that peptides with lower m/z charge value have more importance than these with higher m/z charge values.

Table 2. First model of ovarian cancer diagnostic system build on peptides with m/z charge values of range 0 - 20 000Da

Patterns	2.2871	244.9524	435.0751	Prelevance (%)
P1		$<$52.8014	$\geq$28.0563	95
P2	$<$4.1559	$<$52.8014		34
P3	$<$4.1559		$\geq$28.0563	32
N1		$\geq$52.8014	$<$28.0563	89
N2	$\geq$4.1559	$\geq$52.8014		63
N3	$\geq$4.1559		$<$28.0563	58

5 Discussion

5.1 Discriminatory Analysis of Protein Mass Spectrometry Data

It has been seen that both models of ovarian cancer diagnostic system build by using LAD have sensitivities and specificities of 100% validated by multiple leave-one-out experiments. It justifies thesis that they can be used to distinguish ovarian cancer cases from control cases with high accuracy. As was shown in [5] it is possible to build model for the stage 1 ovarian cancer detection, that with low invasiveness of research leaning on serum of blood gives a perfect tool to early detection of tumors in which the patient's blood subjected the processing and analysis would give us answer about his membership to cancer or healthy cases. Also the peptides contained in the support set can be examined as a possible contributors or blockers of neoplasmic processes that in future can lead to individualization of therapy.

5.2 Parallel Computation

The parallel implementation of proteomic ovarian cancer diagnosis system based on logical analysis of data proved the strength of parallel processing. Both Seti-like cluster and OpenMosix with PVM cluster accelerated considerably the computational process necessary to build and to validate models of ovarian cancer diagnostic system. Since their hardware and software structure allow for simple extension by addition additional computers to the cluster as nodes it will be possible in future to increase further their computational power.

Table 3. Second model of ovarian cancer diagnostic system build on peptides with m/z charge values of range 700 - 12 000Da

P.	704.4728	704.9683	831.0739	2665.3973	3536.2919	4003.6449	4593.1246	6803.0344	(%)
P1			<22.55		≥7.4681	<30.0111			59
P2				<12.4736	≥7.4681	<30.0111			59
P3		≥37.7104				<30.0111		<8.1813	44
P4		≥37.7104			≥7.4681			<8.1813	34
P5		≥37.7104		<12.4736			<27.0132		33
P6			<22.55			<30.0111	<27.0132	<8.1813	28
P7		<37.7104	<22.55		≥7.4681		≥27.0132		19
P8		≥37.7104	<22.55		≥7.4681		<27.0132		18
P9			<22.55	<12.4736	≥7.4681			≥8.1813	11
P10			<22.55	<12.4736	<7.4681			<8.1813	11
P11	≥53.7858				≥7.4681		≥27.0132		9
N1		<37.7104	≥22.55	≥12.4736					43
N2				≥12.4736	<7.4681	≥30.0111			42
N3		<37.7104	≥22.55		<7.4681				38
N4		<37.7104		≥12.4736		≥30.0111	<27.0132		33
N5				≥12.4736	<7.4681			≥8.1813	32
N6			≥22.55			≥30.0111	<27.0132	≥8.1813	25
N7	<53.7858			≥12.4736	<7.4681		≥27.0132		23
N8					<7.4681		≥27.0132	≥8.1813	22
N9		<37.7104			≥7.4681	≥30.0111	<27.0132	<8.1813	5
N10		≥37.7104				≥30.0111	≥27.0132	≥8.1813	5

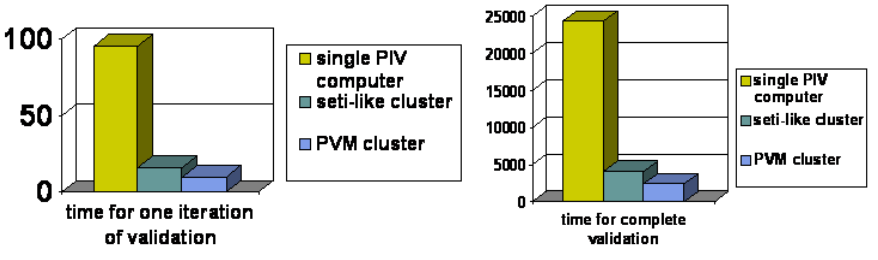


Fig. 3. Comparison of times of validation on single PIV computer, Seti-like cluster and OpenMosix with PVM cluster

Acknowledgment

It is my duty to thank prof. A. Polański from Department of Automatic Control at Silesian University of Technology and prof. P. Widłak from Institute of Oncology Branch Gliwice for their continuous advice and support in this research.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. John Wiley & Sons New York (2001)
2. Olshen, R.A., Stone, C.J., Breiman, L.: Classification and Regression Trees. Chapman & Hall New York (1984)

3. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and other Kernel-based Learning Methods. Cambridge University Press New York (2000)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press New York (1996)
5. Alexe, G., Alexe, S., Liotta, L.A., Petricoin, E., Reiss, M., Hammer, P.L.: Ovarian Cancer Detection by Logical Analysis of Proteomic Data. Proteomic **4** (2004) 766-783
6. Crama, Y., Hammer, P.L., Ibaraki, T.: Ann. Operations Res **16** (1988) 299-326
7. Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A. et al.: IEEE Transactions on Knowledge and Data Engineering **12** (2000) 292-306
8. Simon, R. et al.: Pitfalls in the analysis of DNA microarray data class prediction methods. J Natl Cancer Inst **95** (2003) 14-18
9. Simon, R. et al.: When is a genomic classifier ready for prime time? Nature Clinical Practice, Oncology **1** (2004) 4-5

Author Index