

Yunquan Zhang  
Kenli Li  
Zheng Xiao (Eds.)

Communications in Computer and Information Science

207

# High Performance Computing

8th CCF Conference, HPC 2012  
Zhangjiajie, China, October 2012  
Revised Selected Papers

 Springer



Editorial Board

Simone Diniz Junqueira Barbosa

*Pontifical Catholic University of Rio de Janeiro (PUC-Rio),  
Rio de Janeiro, Brazil*

Phoebe Chen

*La Trobe University, Melbourne, Australia*

Alfredo Cuzzocrea

*ICAR-CNR and University of Calabria, Italy*

Xiaoyong Du

*Renmin University of China, Beijing, China*

Joaquim Filipe

*Polytechnic Institute of Setúbal, Portugal*

Orhun Kara

*TÜBİTAK BİLGEM and Middle East Technical University, Turkey*

Igor Kotenko

*St. Petersburg Institute for Informatics and Automation  
of the Russian Academy of Sciences, Russia*

Krishna M. Sivalingam

*Indian Institute of Technology Madras, India*

Dominik Ślęzak

*University of Warsaw and Infobright, Poland*

Takashi Washio

*Osaka University, Japan*

Xiaokang Yang

*Shanghai Jiao Tong University, China*

Yunquan Zhang Kenli Li Zheng Xiao (Eds.)

# High Performance Computing

8th CCF Conference, HPC 2012  
Zhangjiajie, China, October 29-31, 2012  
Revised Selected Papers



Springer

## Volume Editors

Yunquan Zhang  
Chinese Academy of Sciences  
Beijing 100190, P.R. China  
E-mail: yunquan.cas@gmail.com

Kenli Li  
Hunan University  
Changsha 410082, P.R. China,  
E-mail: lk1510@263.net

Zheng Xiao  
Hunan University  
Changsha 410082, P.R. China  
E-mail: xiaozheng206@163.com

ISSN 1865-0929

e-ISSN 1865-0937

ISBN 978-3-642-41590-6

e-ISBN 978-3-642-41591-3

DOI 10.1007/978-3-642-41591-3

Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013951269

CR Subject Classification (1998): B.2.4, B.3.2, C.1-2, C.4, C.5.1

© Springer-Verlag Berlin Heidelberg 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

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

Printed on acid-free paper

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

# Preface

Computational science, the use of advanced computing capabilities to understand and verify complex problems, has become crucial to scientific research, economic competitiveness, and national security. Parallel computing is one of the most important fields in computational science, including parallel hardware system such as supercomputers and their large-scale parallel software. Currently, almost all of the fields of fundamental research in physics, chemistry, biology, astronomy, earth science, areas of national economy in the automotive, steel, aerospace, biomedicine, and new energy sectors, as well as related fields of national security such as information security and nuclear weapons are highly dependent on the development of computational science. Besides, the demand of computational science in the above-mentioned areas is increasing every day. Computational science plays an indispensable role in the process toward an innovation-oriented country. Except for theory and experiments, computational science has become the third most important tool for scientific exploration and engineering studies. Hence, the U.S. President's Information Technology Advisory Committee (PITAC) in the report submitted to President George W. Bush in June 2005, entitled "Computational Science: To Ensure the Competitive Advantage of the United States," suggested the U.S. government develop long-term plans and provide long-term funding for computational science, to ensure the competitive advantage and national security of the United States. The committee agrees that computational science is one of the most important technical fields of the twenty-first century, because it is indispensable to the progress of the entire society.

For a long time, China and the world's major developed countries attached great importance to the development and application of high-performance computing, and they made numerous achievements in this area. "TianHe 1A," ranked first in the world recently, and the world's second supercomputer system "Dawning Nebulae" are the representative achievements after years of investment of our country. Meanwhile, such high-performance domestic processors as ShenWei, Loongson, and FeiTeng are also beginning to take shape, and the petaflop supercomputer system "ShenWei BlueLight" that uses domestic processors was developed successfully, opening the way for the application of domestic processors in the field of supercomputers. In the next 5–10 years, the development of exascale supercomputer systems will be the common goal of China, the United States, Europe, Japan, and other major developed countries and regions.

So far, heterogeneous parallel computing that can take advantage of all the petaflop supercomputer processors and acceleration components has barely made any breakthroughs. The development of parallel application software faces bottlenecks and needs to progress.

The National Annual Conference on High-Performance Computing (HPC China 2012) is a national conference on all aspects of high-performance

computing. It serves as a forum to present current work by researchers from around the country as well as to highlight activities in China in the high-performance computing area. HPC China 2013 was held during October 29–31, 2013, at Guilin, Guangxi. The conference received more than 260 papers, and each paper was carefully reviewed by the Technical Program Committee members. Finally, fewer than 100 papers were selected. This volume comprises 14 excellent papers recommended by the Program Committee members, including parallel architecture, GPU computing, resource scheduling, parallel algorithm, and performance evaluation.

This was the first attempt for this conference to cooperate with the international academic journal and organization on the HPC's special issue. In the process of the organization, it is inevitable that omissions were made. We hope the readers provide valuable advice and suggestions so that we can improve future editions. On behalf of the Organizing Committee, we thank all the experts of the Program Committee for their work in reviewing the articles.

March 2013

Yunquan Zhang  
Kenli Li  
Zheng Xiao

# Conference Organization

The National Annual Conference on High-Performance Computing (HPC China 2012) was organized by The Specialty Association of Mathematical and Scientific Software of China Software Industry Association, Technical Committee of High-Performance Computing of China Computer Federation, National Supercomputing Center in Changsha, and Hunan University. It was partly funded by PARATERA, Intel, AMD, and Nvidia, among others.

## General Chair

|               |   |
|---------------|---|
| Guoliang Chen | University of Science and Technology of China |
| Xuejun Yang   | National University of Defense Technology     |

## Program Chair

|             |  |
|-------------|--|
| Ninghui Sun | Institute of Computing Technology, Chinese Academy of Sciences |
|-------------|--|

## Program Committee

|               |   |
|---------------|---|
| Fengbin Qi    | Jiangnan Institute of Computing Technology                            |
| Xuebin Chi    | Computer Network Information Center,<br>Chinese Academy of Sciences   |
| Yunquan Zhang | Institute of Software, Chinese Academy of Sciences                    |
| Qingfeng Hu   | National University of Defense Technology                             |
| Zeyao Mo      | Beijing Institute of Applied Physics and<br>Computational Mathematics |
| Renfa Li      | Hunan University  |

## Conference Executive Chair

|              |   |
|--------------|---|
| Jing Zhang   | Hunan University                          |
| Xiangke Liao | National University of Defense Technology |

## Organizing Committee

|            |   |
|------------|---|
| Liya Deng  | Hunan University                          |
| Nan Li     | National University of Defense Technology |
| Yaping Lin | Hunan University                          |
| Lijun Cai  | Hunan University                          |
| Kenli Li   | Hunan University                          |

## Sponsors





# Table of Contents

|  |     |
|--|-----|
| An Improvement to the OpenMP Version of BoomerAMG .....  | 1   |
| <i>Chunsheng Feng, Shi Shu, and Xiaoqiang Yue</i>  |     |
| Dynamic Partitioning of Scalable Cache Memory for SMT<br>Architectures .....                                     | 12  |
| <i>Wu Jun-Min, Zhu Xiao-Dong, Sui Xiu-Feng, Jin Ying-Qi, and<br/>Zhao Xiao-Yu</i>                                |     |
| Scheduling Model of Virtual Machine Base on Task Type in Multi-core<br>System .....                              | 26  |
| <i>Hui-Xing Chen, Kenli Li, and Lin Shi</i>  |     |
| Dynamic Pricing Strategy for Cloud Computing with Data Mining<br>Method .....                                    | 40  |
| <i>Xing Wu, Ji Hou, Shaojian Zhuo, and Wu Zhang</i>  |     |
| Detecting Communities and Corresponding Central Nodes in Large<br>Social Networks .....                          | 55  |
| <i>Shengyi Jiang and Meiling Wu</i>  |     |
| The Design and Prototype Implementation of a Pipelined Heterogeneous<br>Multi-core GPU .....                     | 66  |
| <i>Junyong Deng, Libo Chang, Guangxin Huang, Lingzhi Xiao,<br/>Tao Li, Lin Jiang, Jungang Han, and Huimin Du</i> |     |
| A Parallel Approach for Real-Time OLAP Based on Node Performance<br>Awareness .....                              | 75  |
| <i>Wei He and Lizhen Cui</i>   |     |
| A Parallel Multigrid Poisson PDE Solver for Gigapixel Image<br>Editing .....                                     | 89  |
| <i>Zhenlong Du, Xiaoli Li, Xiaojian Yang, and Kangkang Shen</i>  |     |
| Parallel Implementation and Optimization of Haze Removal Using<br>Dark Channel Prior Based on CUDA .....         | 99  |
| <i>Yungang Xue, Ju Ren, Huayou Su, Mei Wen, and Chunyuan Zhang</i>   |     |
| Research on the Solution of Heat Exchanger Network MINLP Problems<br>Based on GPU .....                          | 110 |
| <i>Mingxing Xia, Yuxing Ren, Yazhe Tang, Lixia Kang, and<br/>Yongzhong Liu</i>                                   |     |

|   |            |
|---|------------|
| MapReduce-Based Parallel Algorithm for Detecting and Resolving<br>of Firewall Policy Conflict . . . . .               | 118        |
| <i>Qi Xiao, Yunchuan Qin, and Kenli Li</i>  |            |
| DPA-Resistant Algorithms for Trusted Computing System . . . . .   | 132        |
| <i>Lang Li, Kenli Li, Yi Wang, YuMing Xu, Hui Liu, and Ge Jiao</i>  |            |
| Detection of KVM's Virtual Environment and Vulnerability . . . . .  | 140        |
| <i>Li Ruan, Yikai Sun, Limin Xiao, and Mingfa Zhu</i>   |            |
| Scalability Tests of a Finite Element Code on Hundreds of Thousands<br>Cores and Heterogeneous Architecture . . . . . | 151        |
| <i>Jiangyong Ren, ChaoWei Wang, Yingrui Wang, and Rong Tian</i>   |            |
| <b>Author Index . . . . .</b>   | <b>167</b> |

# An Improvement to the OpenMP Version of BoomerAMG

Chunsheng Feng<sup>1</sup>, Shi Shu<sup>2,\*</sup>, and Xiaoqiang Yue<sup>3</sup>

<sup>1</sup> School of Mathematics and Computational Science, Xiangtan University,  
Xiangtan 411105, China

<sup>2</sup> Hunan Key Laboratory for Computation and Simulation in Science and Engineering,  
Key Laboratory of Intelligent Computing and Information Processing of Ministry of Education,  
Xiangtan University, Xiangtan 411105, China

<sup>3</sup> School of Mathematics and Computational Science, Xiangtan University,  
Xiangtan 411105, China

{spring, shushi}@xtu.edu.cn, yuexq1111@163.com  
<http://math.xtu.edu.cn/myphp/math/personal/shushi/>

**Abstract.** Algebraic multigrid(AMG) method is one of the most efficient iterative methods for solving the linear systems which arising from discretizations of partial differential equations. Parallel AMG has been widely used in large-scale scientific and engineering computation. In this paper, considering a class of linear algebraic equations with sparse and banded coefficient matrices, we improve the OpenMP version of BoomerAMG by modifying its modules of the parallel interpolation and the parallel coarse grid operator. The improved version of BoomerAMG is applied to solve the Laplace equation and a class of two-dimensional three-temperature radiative diffusion equations. Numerical results demonstrate that the new method yields better scalability and efficiency.

**Keywords:** AMG method, parallel computing, OpenMP, HYPRE, radiation heat conduction.

## 1 Introduction

Algebraic multigrid(AMG) method is one of the most efficient iterative methods (or preconditioners) for solving the linear systems arising from discretizations of elliptic partial differential equations. AMG has been widely used in large-scale scientific and engineering computation fields. The AMG method was originally proposed by Brandt A et al [1] in the early 1980s. Recently, through the joint efforts of many scholars, AMG has been developed greatly. There are many new AMG methods, such as the energy minimum interpolation AMG method and adaptive AMG methods [2-4,11,12]. However, efficiency of AMG is usually depend on problem properties and discretizations.

---

\* Corresponding author.

Serial computer programming usually not sufficient due to ever increasing problem size and complexity in scientific and engineering computation. Parallel computing is the key to improve computing ability and efficiency. Parallel scalability is an important indicator which impact the computing performance of high-performance parallel computer system directly. Many scholars have conducted extensive research [6,13-15] in order to improve it. Currently, some solver packages based on the parallel AMG methods solver have made important progress. For example, HYPRE[8] (high performance parallel preconditioner library) is one of the most popular iterative solver (preconditioner) package, which is developed by the Lawrence Livermore National Laboratory. BoomerAMG [7] is the most common linear solver and preconditioner in HYPRE.

OpenMP is an application program interface that may be used to explicitly direct multicore (shared memory) parallelism[5]. It is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs. There are several difficulties for multithread implementation for AMG methods: One of the problems is that OpenMP programs sometimes require extra memory space than their corresponding sequential versions. When working with sparse matrices in compressed formats, like the Compressed Sparse Row(CSR) format, we sometimes need to introduce auxiliary memory space. This will become more heavy a burden as the number of OpenMP threads increases, and will lead to seriously reduce its parallel scalability. When the size of the linear algebra system and the number of threads are large, the problem becomes more severe. Therefore, how to take the advantage of sparse linear algebra system's characteristics to optimize BoomerAMG, such as reducing the auxiliary memory space allocated, is important for improving parallel scalability.

The radiation fluid dynamics equations is an important mathematical model to describe the implosion process of Inertial Confinement Fusion (ICF). The numerical solution of the two-dimensional three-temperature (2D3T) radiation diffusion equations takes the mainly time-consuming part in the numerical simulation of ICF (more than 70% of CPU wall time). The solution of a large-scale ICF simulation is challenging. The linear system resulting from the discretization of the radiation diffusion equations is usually large, sparse, highly nonsymmetric and ill-conditioned. The Krylov subspace methods are efficient iterative methods for these linear systems. In order to solve a linear algebraic system of equations efficiently a preconditioner, such as parallel AMG methods, is often necessary to accelerate the Krylov subspace method.

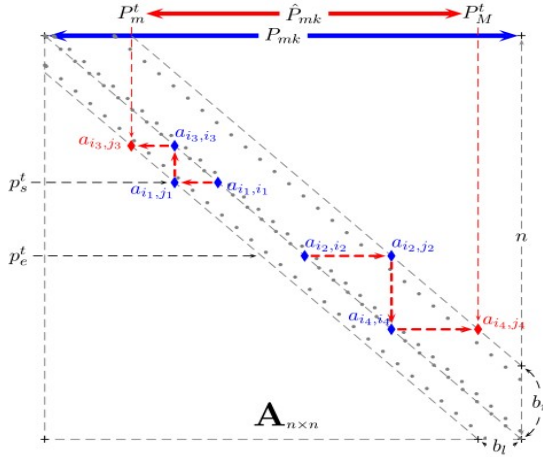
In this paper, we will carefully analyze the parallel interpolation and coarse grid operators in the setup phase of BoomerAMG based on the fact that the coefficient matrices  $A$  we consider are usually banded. We proposed the effective length of the auxiliary array  $P_{mk}$  in the parallel interpolation and auxiliary array  $A_{mk}$  and  $P_{mk}$  in the coarse grid operators on each thread, and the offset estimator. Then we obtained a new auxiliary array formula with less memory overhead, and forming the improved BoomerAMG solver named as BoomerAMG-OPT. The BoomerAMG-OPT is applied to solve the finite difference discrete system of the Poisson equation and the Symmetric Finite Volume Element (SFVE) discrete system[9] of a 2D3T radiation diffusion equation. The numerical results show that, as there is an obvious change in

the length of the auxiliary memory cost between the original and improvements BoomerAMG solver (for example, when the number of OpenMP thread is 128 and the grid size is 2048x2048, for the two-dimensional 5pt discrete Laplacian system, the size of auxiliary arrays in BoomerAMG is more than 100 times of the improved one). The new solver improved the solving ability of the original one, i.e. larger problem can be solved. This also makes BoomerAMG-OPT to obtain higher parallel efficiency.

The rest of the paper is organized as follows: In Section 2 and Section 3, we introduce the improved algorithms for parallel interpolation operator and parallel coarse grid operator generation for BoomerAMG, respectively. In Section 4, we report the results of our numerical experiments conducted in a typical multicore computing environment.

## 2 An Improved Parallel Interpolation Operator Algorithm

The coefficient matrix arises from the discrete system of PDEs and its coarse grid operator (matrix) in AMG methods typically have a sparse banded structure ( see Fig. 1). Where  $b_l$  and  $b_r$  are the half left- and right-bandwidth of the matrix  $A$ . We denote the bandwidth of the matrix  $A$  as  $b_n = b_l + b_r$ . We show that, if the bandwidth of the sparse coefficient matrix  $A$  is relatively small, then we can gain significantly economize in memory usage.



**Fig. 1.** A demo for banded sparse matrix

The default interpolation operator of BoomerAMG is the correction classic interpolation operator[10]. Corresponding to a program module name in HYPRE is `hypr_BoomerAMGBuildInterp`. For convenience of description, here, we only consider the case of two level grids in AMG. Suppose the current grid level of the coefficient matrix  $A = (a_{ij})_{n \times n}$  is symmetric. Let  $I_A = \{1, 2, \dots, n\}$  denote the line number

index set of  $A$ . Suppose we have split the index set  $I_A$  into a set  $C$  of the coarse-level vertices and a set  $F$  of fine-level vertices:

$$I_A = C \cup F, C \cap F = \emptyset,$$

and we denote  $n_c$  as the cardinality of  $C$ , i.e. the number of  $C$ -vertices. Assume that  $f_2c$  is the map from  $F$ -vertices to  $C$ -vertices. We denote  $P = (p_{ij_c})_{n \times n_c}$  as the standard interpolation matrix, where entry

$$p_{ij_c} = \begin{cases} -(a_{ij} + \sum_{k \in D_i^{F,s} \setminus F_i} a_{ik} \hat{a}_{kj} / \sum_{m \in D_i^{C,s}} \hat{a}_{km}) / (a_{ii} + \sum_{k \in D_i^w \cup F_i} a_{ik}) & i \in F, j \in D_i^{C,s}, j_c = f_2c[j] \\ 1.0 & i \in C, j_c = f_2c[i], \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where, Let  $\hat{a}_{ij} := 0$ , if  $a_{ij} > 0$ , and let  $\hat{a}_{ij} := a_{ij}$ , otherwise; let  $D_i^{F,s} = S_i(\varepsilon) \cap F$ ,

$D_i^{C,s} = S_i(\varepsilon) \cap C$ ,  $D_i^w = N_i \setminus (D_i^{C,s} \cup D_i^{F,s})$ ,  $N_i = \{j \in I_A : a_{ij} \neq 0, j \neq i\}$ , for  $\varepsilon \in [0,1)$ ,

$S_i(\varepsilon) = \{j \in N_i : -a_{ij} \geq \varepsilon \max_{k \neq i} (-a_{ik})\}$ ,  $F_i = \{j \in D_i^{F,s} : i \text{ and } j \text{ without the same depended}$

$C$ -vertices\}.

Since the matrix  $P$  is sparse and stored in the CSR format, we need to use an auxiliary integer marker called  $P_{mk}$  to locate the column index of each non-zero entry quickly in BoomerAMG of HYPRE. In fact, to generate the  $i$ -th row of  $P$ , we define that, for  $0 \leq j \leq n-1$

$$P_{mk}[j] = \begin{cases} J_{j_c} & j \in D_i^{C,s}, j_c = f_2c[j], \\ -2-i & j \in D_i^{F,s} \setminus F_i, \\ -1 & \text{otherwise,} \end{cases} \quad (2)$$

where  $J_{j_c}$  is the position of  $p_{ij_c}$  entry in the column index array of the CSR storage of  $P$ . In the OpenMP implementation, we have to allocate the marker  $P_{mk}$  for all OpenMP threads. The length of each  $P_{mk}$  is  $n$  and the total length of  $P_{mk}$  for all threads is then  $N_T \times n$  where  $N_T$  is the total number of OpenMP threads. When  $N_T$  is large, the memory cost for  $P_{mk}$  is considerable. If the bandwidth of the sparse coefficient matrix  $A$  is relatively small, and a certain conditions of the parallel partition for  $I_A$  are met, it can be found that the length of  $P_{mk}$  on each thread may be much less than  $n$ .

Denote  $P'_m$  and  $P'_M$  as the lower and upper column index of non-zero entries in  $A$  of the  $t$ -th OpenMP thread, respectively. Assume that  $p'_s$  and  $p'_e$  are respectively the lower and upper row index of  $I_A$  of the  $t$ -th OpenMP thread. Fig. 1 shows the

corresponding position of  $p'_s$ ,  $p'_e$ ,  $P'_m$  (corresponding non-zero entry  $a_{i_3, j_3}$  of  $A$ ) and  $P'_M$  (corresponding non-zero entry  $a_{i_4, j_4}$  of  $A$ ).

If  $I_A$  is assigned to each OpenMP thread in sequence and load balancing (i.e., the row numbers difference between each thread does not exceed one). Suppose  $p'_s \approx (t-1) \times n / N_T$ ,  $p'_e \approx t \times n / N_T$ , then we obtain the following theorem about length and offset estimates for  $P'_{mk}$ .

**Theorem 1.** If  $I_A$  is assigned to each OpenMP thread in sequence and load balancing, then the length and offset estimates of  $P'_{mk}$  is:

$$L'_p \leq \min(n, n / N_T + 2b_n), \quad P'_m \geq \max(0, (t-1) \times n / N_T - 2b_l)$$

**Proof:** By the equation (1) of the first branch, we know that the denominator  $(a_{ii} + \sum_{k \in D_i^r \cup F_i} a_{ik})$  associated minimum column number is  $k$ , and the molecule

$$\sum_{k \in D_i^{f,s} \setminus F_d} a_{dk} \hat{a}_{kj} \sum_{m \in D_i^{c,s}} \hat{a}_{km}$$

associated minimum column number is  $j$ .

Firstly, we consider column index  $k$ . Combing with Fig. 1 we can know that  $k$  ( $j_1$  in Fig. 1) is the minimum index of which is directly adjacent to vertices of  $I_A$  in the  $t$ -th OpenMP thread. There is

$$k \geq j_1 = i_1 - b_l = p'_s - b_l.$$

Then, we consider column index  $j$ . Combing with Fig. 1 we can know that  $j$  ( $j_3$  in Fig. 1) is the minimum index of which is indirect adjacent to vertices of  $I_A$  in the  $t$ -th OpenMP thread. There is

$$j \geq j_3 = i_3 - b_l = (i_1 - b_l) - b_l = i_1 - 2b_l = p'_s - 2b_l.$$

Combining the value of  $j$  and  $k$  with equation (2), we can obtain

$$P'_m \geq \max(0, \min(j, k)) = \max(0, p'_s - 2b_l) \approx \max(0, (t-1) \times n / N_T - 2b_l).$$

Similar to the derivation of  $P'_m$ , we can get

$$P'_M \leq \min(n-1, p'_e + 2b_r) \approx \min(n-1, t \times n / N_T + 2b_r)$$

Then, there is

$$L'_p = P'_M - P'_m + 1 \leq p'_e - p'_s + 2b_l + 2b_r + 1 \approx n / N_T + 2b_n.$$

In summary above,  $L'_p \leq \min(n, n / N_T + 2b_n)$ ,  $P'_m \geq \max(0, (t-1) \times n / N_T - 2b_l)$ .

### 3 An Improved Parallel Coarse Grid Operator Algorithm

The default coarse grid operator of BoomerAMG is generated by the Galerkin formula. In HYPRE, this operator is implemented as a program module named `hypre_BoomerAMGBuildCoarseOperator`. Similar to the previous section, we only consider the two-grid case.

Take the restriction operator  $R = P^T$ , and let  $I_R = \{1, 2, \dots, n_c\}$  denote the line number index set of  $R$ . Then, the coarse grid operator of multigrid methods can be built as  $A_c = (a_{ij}^c)_{n_c \times n_c} := RAP = P^T AP$ , where  $a_{ij}^c = \sum_k \sum_l p_{ki} a_{kl} p_{lj}$ ,  $i, j = 1, \dots, n_c$ . Assume that  $c_2 f$  is the map from C-vertices to F-vertices.

Similar to the implement of the parallel interpolation operator, we need to allocate two auxiliary arrays called  $A_{mk}$  and  $P_{mk}$  (see Fig. 2). The length of  $A_{mk}$  and  $P_{mk}$  are  $n$  and  $n_c$ , respectively.  $A_{mk}$  and  $P_{mk}$  can be defined as

$$A_{mk}[j] = \begin{cases} i, & w_{ij} \neq 0, \\ k < i, & \text{otherwise,} \end{cases} \quad P_{mk}[j_c] = \begin{cases} J_{j_c}, & a_{j_c}^c \neq 0, \\ k < A_c^t[i], & \text{otherwise,} \end{cases}$$

where  $W = (w_{ij})_{n_c \times n} := RA$ ,  $A_c^t$  is the row array of CSR storage for  $A_c$ ,  $J_{j_c}$  is the position of non-zero entry  $a_{j_c}^c$  in the column array  $A_c^t$ .

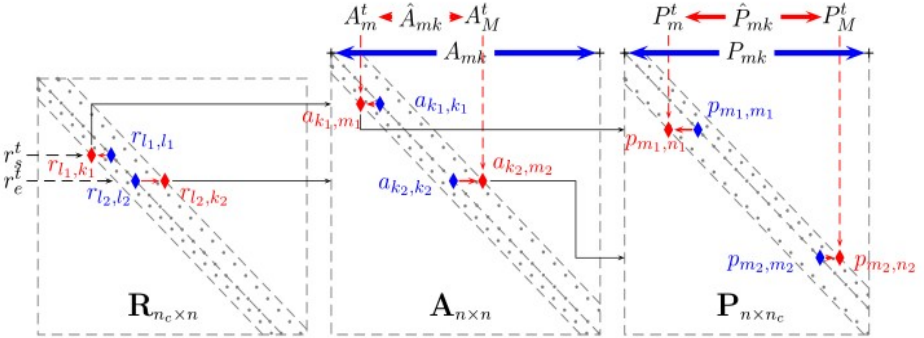


Fig. 2. A demo for  $A_c = RAP$

Denote  $A_m^t$  and  $A_M^t$  as the lower and upper column index of non-zero entries in  $A$  of the  $t$ -th OpenMP thread, respectively. Assume that  $P_m^t$  and  $P_M^t$  are respectively the lower and upper column index of non-zero entries in  $P$  of the  $t$ -th OpenMP thread. Here, we have

$$A_m^t = \min(V_t), \quad A_M^t = \max(V_t), \quad P_m^t = \min(\tilde{V}_t), \quad P_M^t = \max(\tilde{V}_t),$$



where  $V_t = \bigcup_{k=r'_s}^{r'_e} \{m \in N_l : c_2 f[k] \in N_l\}$ ,  $\tilde{V}_t = \bigcup_{k=A'_m}^{A'_M} \{j : c_2 f[j] \in N_k\}$ ,  $r'_s$  and  $r'_e$  are the lower and upper row index of  $I_R$  on the  $t$ -th OpenMP thread. Fig. 2 shows the corresponding position of  $r'_s$ ,  $r'_e$ ,  $A'_m$  (corresponding non-zero entry  $m_1$  of  $A$ ),  $A'_M$  (corresponding non-zero entry  $m_2$  of  $A$ ),  $P'_m$  (corresponding non-zero entry  $n_1$  of  $P$ ) and  $P'_M$  (corresponding non-zero entry  $n_2$  of  $P$ ).

By using a similar proof as in Theorem 1, we can get the following theorem about the length and the offset estimates for  $A_{mk}$  and  $P_{mk}$ .

**Theorem 2.** If  $c_2 f$  is monotonically increasing, then the length and the offset estimates of  $A_{mk}$  is:

$L'_A \leq \min(n, c_2 f[r'_e] - c_2 f[r'_s] + b_l + b_r + b_l^T + b_r^T + 1)$ ,  $A'_m \geq \max(0, c_2 f[r'_s] - b_l^T - b_r)$ ,  
the length and the offset estimates of  $A_{mk}$  is:

$$L'_p \leq f_2 c[n_1] - f_2 c[n_2] + 1, \quad P'_m \geq f_2 c[n_1],$$

where  $b_l^T$  and  $b_r^T$  are the left and right half-bandwidth of  $A^T$ ,

$$n_1 = \max(j \in [0, m_1 - b_l] : f_2 c[j] \geq 0), \quad n_2 = \min(j \in [m_2 + b_r, n] : f_2 c[j] \geq 0),$$

and

$$m_1 = \max(j \in [0, A'_m] : f_2 c[j] = -1), \quad m_2 = \min(j \in [A'_M, n] : f_2 c[j] = -1).$$

*Remark 1.* In BoomerAMG,  $c_2 f$  is monotonically increasing by default.

**Definition 1.** Denote  $a'_s$  and  $a'_e$  as the lower and upper row index of  $I_A$  of the  $t$ -th OpenMP thread, respectively. If all row indexes  $k \in I_R$  satisfy  $a'_s \leq c_2 f[k] \leq a'_e$ , then it's called the parallel partition of  $I_R$  is limited by the parallel partition of  $I_A$ .

**Corollary 1.** Assume that  $A$  is symmetric,  $I_A$  is assigned to each OpenMP thread in sequence and load balancing and the parallel partition of  $I_R$  is limited by the parallel partition of  $I_A$ , then the length and the offset estimates of  $A_{mk}$  is:

$$L'_A \leq \min(n, n / N_T + 2b_n), \quad A'_m \geq \max(0, t \times n / N_T - b_n).$$

If we do not consider the possibility that the bandwidth of  $A$  can be much smaller than  $n$ , then we will need two auxiliary arrays with length  $nN_T$ . However, as we noted above (see Theorem 1 and Corollary 1), we only need two arrays of length  $n + 2b_n N_T$ . When  $n \gg b_n$  and  $N_T$  is relatively large, we can save a lot of memory by using these improved estimates. In fact, this will reduce not only the storage cost but also the time needed to allocate and initialize memory.

When Theorem 1 and Corollary 1 are applied to BoomerAMG, then we obtain the improved BoomerAMG named as BoomerAMG-OPT. We name them corresponding Preconditioned Conjugate Gradient (PCG) algorithm as BoomerAMG-CG and BoomerAMG-OPT-CG, respectively. In the next section, we will introduce some numerical example results.

## 4 Numerical Experiments

In this section, we perform two numerical experiments and analyze the performance of BoomerAMG (in HYPRE) and BoomerAMG-OPT we proposed in Section 3. The first numerical example is for the Laplace equation, and then for a 2D3T radiative diffusion equation. We use a HP desktop PC which is equipped with two Intel Xeon(R) X5590 (3.33GHz, 8cores) and 24GB RAM. The experimental environment is CentOS 6.2 , GCC 4.4.6 and hypre-2.7.0b (with “-O2” optimization parameter). The stopping criteria is that the relative residual in the Euclidian norm is less than  $10^{-6}$ .

**Example 1.** considering the following Laplacian equation

$$\begin{cases} -\Delta u = 1, & \mathbf{x} \in \Omega, \\ u = 0, & \mathbf{x} \in \partial\Omega, \end{cases} \quad (7)$$

where  $\Omega := (0,1)^d, x \in R^d, d = 2, 3$ .

According to the linear equations which arise from the 2D-5pt and 3D-7pt discretizations of the Laplace equation, Table 1. show out the total length and the comparison of these two auxiliary arrays on the finest level of BoomerAMG and BoomerAMG-OPT, in which  $N_T$  is the OpenMP thread numbers, Size is the unknown numbers,  $P_1$  and  $P_2$  are the auxiliary arrays for interpolate and coarse grid operator, respectively,  $L_o^i$  and  $L_n^i$  are the length size in BoomerAMG and BoomerAMG-OPT, respectively.

**Table 1.** Length and relative ratio of the aux-arrays in BoomerAMG and BoomerAMG\_OPT

| $N_T$ | Size = 2048 × 2048 |         |                 |         |         |                 | Size = 256 × 256 × 256 |         |                 |         |         |                 |
|-------|--------------------|---------|-----------------|---------|---------|-----------------|------------------------|---------|-----------------|---------|---------|-----------------|
|       | $P_1$              |         | $P_2$           |         |         |                 | $P_1$                  |         | $P_2$           |         |         |                 |
|       | $L_o^1$            | $L_n^1$ | $L_o^1 / L_n^1$ | $L_o^2$ | $L_n^2$ | $L_o^2 / L_n^2$ | $L_o^1$                | $L_n^1$ | $L_o^1 / L_n^1$ | $L_o^2$ | $L_n^2$ | $L_o^2 / L_n^2$ |
| 2     | 32.0               | 16.0    | <b>2.0</b>      | 16.0    | 8.0     | <b>2.0</b>      | 16.0                   | 8.3     | <b>1.9</b>      | 8.0     | 4.2     | <b>1.9</b>      |
| 4     | 64.0               | 16.1    | <b>4.0</b>      | 32.0    | 8.1     | <b>4.0</b>      | 32.0                   | 8.8     | <b>3.7</b>      | 16.0    | 4.6     | <b>3.5</b>      |
| 8     | 128.0              | 16.2    | <b>7.9</b>      | 64.0    | 8.2     | <b>7.8</b>      | 64.0                   | 9.8     | <b>6.6</b>      | 32.0    | 5.3     | <b>6.0</b>      |
| 16    | 256.0              | 16.5    | <b>15.5</b>     | 128.0   | 8.4     | <b>15.3</b>     | 128.0                  | 11.8    | <b>10.9</b>     | 64.0    | 6.8     | <b>9.4</b>      |
| 32    | 512.0              | 17.0    | <b>30.2</b>     | 256.0   | 8.7     | <b>29.3</b>     | 256.0                  | 15.8    | <b>16.3</b>     | 128.0   | 9.8     | <b>13.0</b>     |
| 64    | 1024.0             | 18.0    | <b>57.0</b>     | 512.0   | 9.5     | <b>54.0</b>     | 512.0                  | 23.8    | <b>21.6</b>     | 256.0   | 15.8    | <b>16.3</b>     |
| 128   | 2048.0             | 19.9    | <b>102.9</b>    | 1024.0  | 11.0    | <b>93.3</b>     | 1024.0                 | 39.6    | <b>25.8</b>     | 512.0   | 27.6    | <b>18.5</b>     |

It can be seen from Table 1.: the total length of  $P_{mk}$  is linearly dependent on the number of OpenMP threads in BoomerAMG, but the total length of  $\hat{P}_{mk}$  is only weakly dependent on the number of OpenMP threads in BoomerAMG-OPT; The relative ratio of the total length of the auxiliary array in BoomerAMG and BoomerAMG-OPT monotonically increase as the number of OpenMP threads increasing; Specifically, when the size of the two-dimensional grid is 2048x2048 and

OpenMP threads is 128 (see the last line of data of Table 1.), the auxiliary memory demand of BoomerAMG reach 2GB but for BoomerAMG-OPT it is only 19.9 MB, namely the improved algorithm reduces the memory requirements is about 102.9 times. Due to the improvement on the length of the auxiliary memory of the new algorithm, making BoomerAMG-OPT can solve larger scale linear systems than the original one.

**Table 2.** Wall time and relative ratio of Setup in BoomerAMG and BoomerAMG\_OPT

| Size              | $N_T = 2$ |       |             | $N_T = 4$ |       |             | $N_T = 6$ |       |             | $N_T = 8$ |       |             |
|-------------------|-----------|-------|-------------|-----------|-------|-------------|-----------|-------|-------------|-----------|-------|-------------|
|                   | $T_o$     | $T_n$ | $T_o / T_n$ | $T_o$     | $T_n$ | $T_o / T_n$ | $T_o$     | $T_n$ | $T_o / T_n$ | $T_o$     | $T_n$ | $T_o / T_n$ |
| 512 <sup>2</sup>  | 0.19      | 0.19  | <b>1.00</b> | 0.16      | 0.15  | <b>1.07</b> | 0.15      | 0.14  | <b>1.07</b> | 0.15      | 0.14  | <b>1.07</b> |
| 1024 <sup>2</sup> | 0.87      | 0.82  | <b>1.06</b> | 0.73      | 0.69  | <b>1.06</b> | 0.68      | 0.64  | <b>1.06</b> | 0.66      | 0.62  | <b>1.06</b> |
| 2048 <sup>2</sup> | 3.87      | 3.49  | <b>1.11</b> | 3.08      | 2.85  | <b>1.08</b> | 2.99      | 2.64  | <b>1.13</b> | 2.85      | 2.57  | <b>1.11</b> |
| 64 <sup>3</sup>   | 0.55      | 0.52  | <b>1.06</b> | 0.43      | 0.41  | <b>1.05</b> | 0.38      | 0.35  | <b>1.09</b> | 0.34      | 0.33  | <b>1.03</b> |
| 128 <sup>3</sup>  | 5.49      | 5.24  | <b>1.05</b> | 4.02      | 3.70  | <b>1.09</b> | 3.56      | 3.20  | <b>1.11</b> | 3.35      | 2.95  | <b>1.14</b> |
| 256 <sup>3</sup>  | 50.06     | 47.16 | <b>1.06</b> | 37.54     | 33.65 | <b>1.12</b> | 33.36     | 29.33 | <b>1.14</b> | 31.45     | 27.00 | <b>1.16</b> |

Table 2. shows the CPU wall time for the SETUP phase of BoomerAMG and BoomerAMG-OPT, respectively. In which  $T_o$  is the CPU wall time of SETUP phase in BoomerAMG,  $T_n$  is the CPU wall time of SETUP phase in BoomerAMG-OPT.

It can be seen from Table 2.: the CPU wall time of BoomerAMG-OPT is less than BoomerAMG, and the time ratio is increasing with the size of the problem and the OpenMP threads increasing.

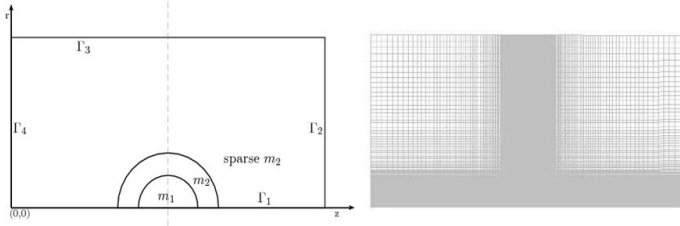
**Example 2.** considering the following 2D3T radiative diffusion equations

$$\begin{cases} c_e \rho \frac{\partial T_e}{\partial t} - \nabla \cdot (\kappa_e \nabla T_e) &= \rho^2 w_{ei} (T_i - T_e) + \rho^2 w_{er} (T_r - T_e), \\ c_i \rho \frac{\partial T_i}{\partial t} - \nabla \cdot (\kappa_i \nabla T_i) &= \rho^2 w_{ei} (T_e - T_i), \\ 4aT_r^3 \frac{\partial T_r}{\partial t} - \nabla \cdot (\kappa_r \nabla T_r) &= \rho^2 w_{er} (T_e - T_r), \end{cases} \quad (8)$$

where

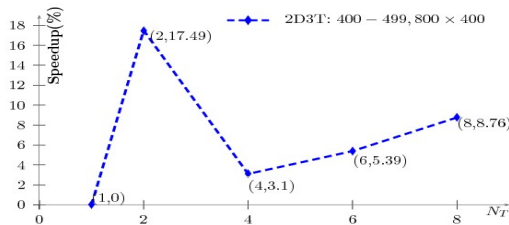
$$c_\zeta = \begin{cases} c_\zeta^1, & m_1, \kappa_\zeta = \begin{cases} A_{\kappa_\zeta}^1 T_\zeta^{\alpha_\zeta^1}, & m_1, \kappa_\zeta = \begin{cases} A_{\kappa_\zeta}^1 \rho_1^{n_1} T_\zeta^{3+\beta_1}, & m_1, \rho = \begin{cases} \rho_1, & m_1, w_{ei} = \begin{cases} A_{w_{ei}}^1 T_e^{\gamma_1}, & m_1, w_{er} = \begin{cases} A_{w_{er}}^1 T_e^{\gamma_2}, & m_1, \\ A_{w_{er}}^2 T_e^{\gamma_2}, & m_2 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \\ c_\zeta^2, & m_2, \kappa_\zeta = \begin{cases} A_{\kappa_\zeta}^2 T_\zeta^{\alpha_\zeta^2}, & m_2, \kappa_\zeta = \begin{cases} A_{\kappa_\zeta}^2 \rho_2^{n_2} T_\zeta^{3+\beta_2}, & m_2, \rho = \begin{cases} \rho_2, & m_2, w_{ei} = \begin{cases} A_{w_{ei}}^2 T_e^{\gamma_1}, & m_2, w_{er} = \begin{cases} A_{w_{er}}^2 T_e^{\gamma_2}, & m_2, \\ A_{w_{er}}^3 T_e^{\gamma_2}, & m_2 \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases} \end{cases}$$

$m_1$  and  $m_2$  are two different materials,  $c_\zeta^l, A_{\kappa_\zeta}^l, A_{\kappa_\zeta}^l, n_l, \beta_l, A_{ei}^l, A_{er}^l, \gamma_l, \alpha_\zeta^l, \rho_l, l=1,2$ ,  $\zeta=e,i$  and  $a$  are all given constants.



**Fig. 3.** Domain and mesh for equation (8)

For the given SFVE discretization linear system of the equation (8), the relative speedup of BoomerAMG-OPT-CG and BoomerAMG-CG is shown in Fig. 4, where the mesh size is 800x400 (See the right figure of Fig. 3), the time steps are from the 400th to 499th.



**Fig. 4.** Relative speedup for BoomerAMG\_OPT-CG and BoomerAMG-CG

It can be seen from Fig. 4: the parallel computational efficiency of BoomerAMG-OPT-CG is higher than BoomerAMG-CG, it increases upward trend with the OpenMP threads increasing (except OpenMP thread number is two, there may be some relation with the hardware construct).

Summary the numerical result of example one and example two, we can deduce the conclusion BoomerAMG-OPT is overtaken BoomerAMG on the memory save and parallel computational efficiency. At the same time , it also verified the correctness of Theorem 1, Theorem 2 and Corollary 1.

**Acknowledgments.** The authors would like to thank Dr. Chensong Zhang from NCMS, Academy of Mathematics and System Sciences of China for his helpful comments and suggestions. Feng is partially supported by the National Natural Science Foundation of China under Grant No.11201398 and Project of Scientific Research Fund of Hunan Provincial Education Department of China 11C1219 and 12A138. Shu is partially supported by the National Natural Science Foundation of China under Grant Nos. 91130002 and 11171281, Project of Scientific Research Fund of Hunan Provincial Science and Technology Department Grant No. 2012FJ4302 in China and Specialized research Fund for the Doctoral Program of Higher Education Grant No. 20124301110003 in China.

## References

1. Brandt, A., McCormick, S.F., Ruge, J.W.: Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations, Tech. Rep., Institute for Computational Studies, Colorado State University (1982)
2. Brandt, A.: General Highly Accurate Algebraic Coarsening. *Electron. Trans. Numer. Anal.* 10, 1–20 (2000)
3. Brandt, A.: Multiscale scientific computation: Review 2001. In: Barth, T.J., Chan, T.F., Haimes, R. (eds.) *Multiscale and Multiresolution Methods: Theory and Applications*, pp. 1–96. Springer, Heidelberg (2001)
4. Brezina, M., Falgout, R., Maclachlan, S., Manteuffel, T., McCormick, S., Ruge, J.: Adaptive smoothed aggregation. *SIAM J. Sci. Comput.* 25, 1896–1920 (2004)
5. Dagum, L., Menon, R.: OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5(1), 46–55 (1998)
6. Feng, C.S., Shu, S., Wang, J.X.: An Efficient Parallel Preconditioner For Solving H(Curl) Elliptic Problem And Parallel Implementation. *Journal of Numerical Methods and Computer Applicat.* 33(1), 48–58 (2012)
7. Henson, V.E., Yang, U.M.: BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics* 41, 155–177 (2002)
8. HYPRE: High performance preconditioner,  
<http://computation.llnl.gov/casc/hypre/>
9. Nie, C.Y., Shu, S., Sheng, Z.Q.: Symmetry-preserving Finite Volume Element Scheme on Unstructured Quadrilateral Grids. *Chinese Journal of Computational Physics* 26(2), 91–99 (2009)
10. Sterck, H.D., Yang, U.M., Heys, J.J.: Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM J. Mat. Anal. Appl.* 27(4), 1019–1039 (2006)
11. Wan, W.L., Chan, T.F., Smith, B.: An Energy-Minimizing Interpolation for Robust Multigrid Methods. *SIAM J. Sci. Comput.* 21, 1632–1649 (1999)
12. Xu, J., Zikatanov, L.: On An Energy-Minimizing Basis for Algebraic Multigrid Methods. *Comput. Vis. Sci.* 7, 121–127 (2004)
13. Xu, X.W., Mo, Z.Y., Cao, X.L.: Parallel scalability analysis for multigrid solvers in HYPRE. *Journal of Software* 20, 8–14 (2009)
14. Yuan, Q.B., Zhao, J.B., Chen, M.Y., Sun, N.H.: Performance Bottleneck Analysis and Solution of Shared Memory Operating System on a Multi-Core Platform. *Journal of Computer Research and Development* 48(12), 2268–2276 (2011)
15. Zhang, Y.Q., Sun, J.C., Chi, X.B., Tang, Z.M.: Memory Complexity Analysis on Numerical Programs. *Chinese Journal of Computers* 23(4), 363–373 (2000)

# Dynamic Partitioning of Scalable Cache Memory for SMT Architectures

Wu Jun-Min<sup>1,2</sup>, Zhu Xiao-Dong<sup>1,2</sup>, Sui Xiu-Feng<sup>3</sup>,  
Jin Ying-Qi<sup>1,2</sup>, and Zhao Xiao-Yu<sup>1,2</sup>

<sup>1</sup> School of Computer Science and Technology,  
University of Science and Technology of China, Hefei, 230027, China

<sup>2</sup> Suzhou Institute for Advanced Study,  
University of Science and Technology of China, Suzhou, 215123, China

<sup>3</sup> National Research Center for Intelligent Computing Systems,  
Institute of Computing Technology, Beijing 100091  
{jmwu, zhaoxy299}@ustc.edu.cn,  
{xdzhu001, sxf, yqjin}@mail.ustc.edu.cn

**Abstract.** The one-level data cache [1], which is optimized for bandwidth, eliminates the overhead to maintain containment and coherence. And it is suitable for future large-scale SMT processor. Although the design has good scalability, large-scale SMT architecture exacerbates the stress on cache, especially for the bank-interleaved data cache referred to in paper [1]. This paper proposes a dynamic partitioning method of scalable cache for large-scale SMT architectures. We extend the scheme proposed in [2] to multi-banking cache. Since memory reference characteristics of threads can change very quickly, our method collects the miss-rate characteristics of simultaneously executing threads at runtime, and partitions the cache among the executing threads. The partitioning scheme has been evaluated using a modified SMT simulator modeling the one-level data cache. The results show a relative improvement in the IPC of up to 18.94% over those generated by the non-partitioned cache using standard least recently used replacement policy.

**Keywords:** Cache Partitioning, Scalable Multi-banking Cache Memory, Bank Caching, Simultaneous Multithreading.

## 1 Introduction

Simultaneous multithreading (SMT) [3, 4] is a latency-tolerant architecture that executes multiple instructions from multiple threads each cycle. SMT works by converting thread-level parallelism into instruction-level parallelism, effectively feeding instructions from different threads into the functional units of a wide-issue, out-of-order superscalar processor [3]. SMT dynamically improves shared resources utilization, such as functional units and caches. SMT processor uses vertical and horizontal sharing to tolerate latencies, coping with branch

mispredictions, deeper pipelines, and the larger cache miss penalties. Intel Pentium 4 [5] and the proposed Alpha 21464 [6] both belong to SMT.

To implement higher-context and super-wide SMT processor, however, a lot of challenges have to be addressed, including the shared register file, the shared cache hierarchy, and the degree of sharing or partitioning of hardware resources. The problems of cache capacity and bandwidth become more serious as the number of threads increases [1]. However, current cache hierarchy can't solve these two problems. The cache hierarchy has to be redesigned and optimized in a large-scale SMT processor. [1] proposed a simple scalable single-level multi-banking cache design. It can optimize the bandwidth demand of large-scale SMT processors, while slightly increasing the latency of primary data cache access.

Furthermore, SMT exacerbates the stress on the cache hierarchy, especially since the LRU replacement scheme treats all references in the same way. Thus, a single thread can easily "pollute" the cache with its data, causing higher miss rates for other threads, and resulting in low overall performance [2]. To solve this problem, [2] presented a dynamic cache partitioning algorithms that minimizes the overall cache miss rate for SMT system. Rather than relying on the standard LRU cache replacement policy, the algorithm dynamically allocates parts of the cache to the most needy threads using on-line estimates of individual thread miss rates and the partition granularity is block or way.

The cache design proposed in [1] can support multiple contexts, but the data cache may still be not large enough to hold all of the working sets of the simultaneously executing threads. Workloads have become much larger and diverse, multimedia programs such as video or audio processing software often consume hundreds of MB and many SPEC CPU2000 benchmarks now have memory footprints larger than 100 MB [7].

This paper improves the dynamic cache partitioning algorithm and applies it to the large-scale SMT architectures. It partitions the single-level multi-banking data cache at "bank" granularity. The partitioning scheme will only allocate a new cache bank to a thread if its current allocation is below its limit. To implement this scheme, we require counters to provide on-line estimates of individual thread miss rates. Based on these counters we can augment non-partitioned multi-banking cache to better allocate cache banks to threads. In order to actually control the allocation to each thread, we use bank caching mechanism (derived from column caching [8, 9]), which allows threads to be assigned to the corresponding banks. Simulation shows that the partitioning algorithm can improve the instructions per cycle (IPC) of the overall workloads.

The remainder of this paper is organized as follows. Section 2 discusses the related work, including the scalable SMT architecture and the partitioning algorithm. In section 3, we first study the optimal cache partitioning problem for the multi-banking cache. Then we extend the definition of marginal gain to the multi-banking cache, and discuss implementation details in Section 4. Section 5 describes methodology and Section 6 presents results. Finally, Section 7 concludes the paper.

## 2 Related Work

The conventional cache hierarchy is only an added overhead and complexity because of the principle of containment and cache coherence across the different levels. The most prominent feature of the architecture proposed in [1] is the elimination of the cache hierarchy. The design only preserves primary instruction and data caches and scales them according to requirement. The large primary data caches, with a large number of ports and banks, increase the overall capacity and the bandwidth.

G. Edward Suh et al. [2] studied one method to reduce cache interface among simultaneously executing threads. The on-line cache partitioning algorithm estimates the miss characteristics of each thread at runtime, and dynamically partitions the cache amongst the threads that are executing simultaneously. The algorithm estimates the marginal gains as a function of cache size and uses a search algorithm to find the partition that minimizes the total number of misses. Moreover, the hardware overheads for the algorithm are minimal.

Our partition work differs from the previous efforts. It works for multi-banking caches with multiple threads, whereas Suh [2] only focused on set-associative with way granularity.

## 3 Dynamic Bank Partitioning Algorithm

This section presents an analytical analysis of bank partitioning which extended from the analysis in [2]. First, we define the optimal bank partition that minimizes the total number of misses for simultaneous threads. Second, to design a partitioning algorithm, we still use the term “marginal gains” proposed in [2] as a way of determining the usefulness of cache space for a thread, but we have to redefine it based on the multi-banking cache.

### 3.1 Optimal Bank Partitioning

Given  $N$  executing threads sharing a multi-banking cache of  $B$  banks with partitioning on a bank granularity, the problem is partition the cache into  $N$  disjoint subsets of banks to minimize the overall misses. Obviously, it isn't reasonable to repartition the cache every load/store instruction, so the partition remains fixed over a time period,  $\pi$ , which is long enough to amortize the repartitioning cost. Let  $b_i$  represent the number of banks assigned to the  $i$ -th thread over the time period. A bank partition is specified by the number of cache banks assigned to each thread, that is  $\{b_1, b_2, \dots, b_N\}$ .

Define  $m_i(c)$  as the number of cache miser for the  $i$ -th thread over a time period  $\pi$  as a function of partition size ( $c$ ) [2]. Let  $D$  and  $S$  represent the number of ways and sets of each cache bank respectively. Then the optimal partition for the period  $\pi$  is the set of values  $\{b_1, b_2, \dots, b_N\}$ , that minimizes the following expression:



$$\text{Total misses over time period } \pi = \sum_{i=1}^N m_i(D \cdot S \cdot b_i) \quad (1)$$

Under the constrain that  $\sum_{i=1}^N b_i = B$ .  $B$  is the total number of banks in the cache.

### 3.2 Marginal Gains

In order to find the optimal partition, [2] used marginal gains of each competing thread to guide the partition. And [2] gave the definition of the marginal gain of a thread at a given cache space  $c$ , namely  $g_i(c)$ .

$$g_i(c) = m_i(c) - m_i(c+1) \quad (2)$$

Then the definition was extended to a partition chunk (one way of the cache in [2])  $g_{\text{way}}(k)$ .

$$g_{\text{way}}(k) = \sum_{c=k \cdot S}^{(k+1) \cdot S - 1} g(c) \quad (3)$$

where  $S$  is the number of sets in the cache.

To partition the cache in bank granularity, we use a partition block that's the same size with one bank ( $D \cdot S$  blocks). The cache allocation to each thread can only be multiples of  $D \cdot S$  blocks. For this purpose, we define marginal gains of a bank  $g_{\text{bank}}(b)$  as follows:

$$g_{\text{bank}}(b) = \sum_{k=b \cdot D}^{(b+1) \cdot D - 1} g_{\text{way}}(k) = \sum_{c=b \cdot D \cdot S}^{(b+1) \cdot D \cdot S - 1} g(c) \quad (4)$$

The meanings of  $D$  and  $S$  are given in subsection 3.1.

If we expand equation (4), we can get the following result:

$$g_{\text{bank}}(b) = m(b \cdot D \cdot S) - m((b+1) \cdot D \cdot S) \quad (5)$$

The marginal gain at a given bank means the number of cache misses that will be reduced by having one more cache bank. Thus, it indicates the benefit of increasing the cache allocation from  $b$  to  $b+1$  banks for a thread.

Meanwhile, Stone et al. [10] and G. Edward Suh et al. [2] gave the well-known, simple greedy algorithms to result in an optimal partition, for both the case where the marginal gain for each thread is a monotonically decreasing function of cache space and a non-monotonically decreasing function.

## 4 Implementation

The previous section discussed some concepts used in our partitioning algorithm. Now, we consider how to implement bank partitioning in multi-banking cache. Our partitioning scheme consists of three parts: marginal gain counters, bank

caching, and a partition controller. First, we use a set of counters to estimate the marginal gains of executing threads. Second, we adopt a mechanism to actually control the allocation to each thread. Finally, the controller determines the best partition based on the information from counters.

#### 4.1 Marginal-Gain Counters

To perform dynamic bank partitioning, the marginal gains of having one more cache bank should be estimated on-line. As discussed in previous work [2],  $g_{way,i}(k)$  is the number of additional hits that the  $i$ -th thread can obtain by having  $k+1$  cache ways compared to the case when it has  $k$  ways. Assuming the LRU replacement policy is used,  $g_{way,i}(0)$  represents the number of hits on the most recently used cache way of the  $i$ -th thread,  $g_{way,i}(1)$  represents the number of hits on the second most recently used cache way of the  $i$ -th thread, and so on.

In multi-banking caches, a set of counters, one for each bank, is maintained per thread. On every cache hit, the corresponding counter is increased. Although we can only estimate marginal gains of having each bank, not each cache way, this is often enough for partitioning if the cache has reasonable amount of banks. In the scalable single-level multi-banking cache design, the primary data cache usually has a great deal of banks, so we can obtain adequate information to guide the partition. Let  $BA$  represents  $B/D$ . It is reasonable to assume that  $BA$  is always greater than 1, because the number of banks is much more than the set associativity in the scalable cache memory.

*Bank-counters for a multi-banking cache:* There is one counter for each bank of the cache. A hit to the MRU blocks of a certain bank whose bank number is between 0 and  $(D-1)$  updates  $counter(0)$ . A hit to the LRU blocks of a certain bank whose bank number is between  $(B-D)$  and  $(B-1)$  updates  $counter(B-1)$ . That is to say a hit to the  $\left(\left\lfloor \frac{i}{BA} \right\rfloor + 1\right)$ -th most recently used blocks of a certain bank whose bank number is between  $\left[\left(i\%BA\right) \cdot D\right]$  and  $\left[\left(\left(i\%BA\right) + 1\right) \cdot D - 1\right]$  updates  $counter(i)$ .

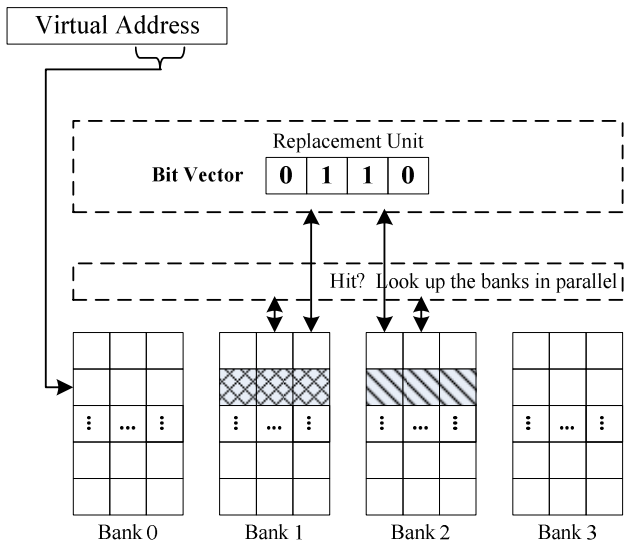
Assuming that cache accesses are well distributed over banks and sets, we can approximate the marginal gain of having additional  $D$  ways from the bank-counters as follows:

$$counter(b) = \sum_{k=b-D}^{(b+1)D-1} g_{way}(k) = \sum_{c=b-D}^{(b+1)D, S-1} g(c) \quad (6)$$

Note that the values of the counters may not monotonically decrease. For example,  $counter(0), counter(1), \dots, counter(BA-1)$  record the number of hits to the MRU way of some banks according to the foregoing definition, since which bank to be accessed is dynamically determined in the decoding stage and every thread can access any cache bank, it's possible that some banks are hotter than others. But the LRU ordering is still partly recorded by the counters.

## 4.2 Bank Caching

Our unit of partition granularity is that of a bank of the multi-banking cache, hence the name “bank caching”. A standard multi-banking cache chooses one bank from all banks to replace. As a result, a thread’s data can occupy any cache bank. Bank caching, on the other hand, restricts the replacement to a subset of banks, which essentially partitions the cache.



**Fig. 1.** Basic Bank Caching. Replacement and hit units are modified to partition the cache via bit vector.

Bank caching specifies replacement candidacy using a bit vector in which a bit indicates if the corresponding bank is a candidate for replacement. Bank caching is implemented by small modifications to a conventional multi-banking cache (Figure 1). The LRU replacement unit is modified so that it replaces the LRU cache block from the candidate banks specified by a bit vector. When the cache is accessed, all banks belonging to the thread are searched in parallel. If there are no hit signals returned, a bank is chosen randomly from the subset indicated by the bit vector, and the LRU cache block of the corresponding set specified by the virtual address is replaced. Since the cache has been partitioned into disjoint subsets, the probability that different threads access the same bank is rather small. That is bank caching won’t result in additional bank interference.

## 4.3 Partitioning Controller

The previous two subsections discussed two mechanisms to enable cache partitioning: marginal gain counters and bank caching. In our implementation, the

partitioning controller (or the operating system) controls these mechanisms to partition cache. The partitioning controller has two main functions. First, it determines a proper cache assignment based on the marginal gain counters. Second, it appropriately updates the counters to reflect dynamic changes in workload behavior.

Every  $\pi$  cycles, the partitioning controller interrupts a running thread and starts to partition the cache. It first reads marginal gain counters to update its data structure for marginal gains. Based on the new marginal gains, it decides a proper partition for each thread and modifies the bit vectors. At last, it clears the counters and restarts a thread.

### Cache Assignment

The previous work [2] assumes that we can control cache allocation at a cache way granularity, and we know marginal gains also at a way granularity. However, with bank-counters for multi-banking cache, we can't accurately estimate marginal gains at a cache way or block granularity. Again, it's very difficult to control the cache assignment at a way granularity. So we have to assign multiples of cache ways at a time, referred as a partition way.

Using bank counters, we obtain marginal gains at a bank granularity. Thus, we use a partition way which is the same size with one bank of the cache ( $D$  ways). The cache allocation to each thread can only be multiples of  $D$  ways. To achieve this goal, we can adopt the definition  $g_{bank}(b)$  specified by (4).

For the case when the marginal gain is not a monotonically decreased function, we randomly choose an initial allocation and use a greedy algorithm to decide a partition. After computing a new partition, we compare it with the previous partition and choose the better one to be a partition for the next partition period. We use the similar algorithm proposed in [2], and the whole algorithm is as follows:

1. Randomly initialize  $\{b_1, b_2, \dots, b_N\}$ .
2. Find a thread that will get the most benefit by having one more bank (index  $i$  for which  $g_{bank,i}(b_i)$  is largest), and the thread that will lose the least by giving up one bank (index  $j (\neq i)$  for which  $g_{bank,j}(b_j-1)$  is smallest).
3. If  $g_{bank,i}(b_i) > g_{bank,j}(b_j-1)$  increase  $b_i$  and decrease  $b_j$ .
4. Repeat step 3 and 4 until  $g_{bank,i}(b_i) \leq g_{bank,j}(b_j-1)$  (maximum  $B$  times)
5. Compare the new partition with the previous one, and choose the better one.

Consider that  $N$  (the number of simultaneous threads) and  $B$  (the number of partition banks) are given, the time complexity of foregoing algorithm is  $O(B \cdot N)$ . To be conservative, for the case when we have eight threads sharing a sixteen-bank data cache, it is reasonable to set the overhead for computing a new partition for every partition period to be 80000 cycles (see also paper [2]).

### Counter Update

There is no doubt that characteristics of thread change dynamically, so the estimation of  $g_{bank}(x)$  must reflect the changes. But our method of estimating  $g_{bank}(x)$  is not quite precise, moreover, we also would like to maintain some history of the memory reference characteristics of a thread. We can achieve both objectives, by giving more weight to the counter value measured in more recent time periods.

When a process starts running for the first time, all marginal gains are set to zero. The partition controller updates the marginal gains ( $g_{bank}(b)$ ) every partition period by giving more weight to the new counter values:

$$g_{bank}(b) = \mu \cdot g_{bank}(b) + counter(b) \quad (7)$$

Here  $\mu$  is between 0 and 1. As a result, the effect of hits in the previous time period exponentially decays.

### 4.4 Optimization — Cache Line Buffer

A small but important optimization is to hold cache data inside the processor load/store unit, allowing a same-line load to be satisfied from the line buffer, instead of from the cache. Think of line buffer as a small multi-ported fully-associated level-zero cache with a FIFO replacement policy. The line buffer can mainly reduce the conflicts to the same cache block, caused by consecutive load instructions, due to spatial locality of reference [11]. The main policies used in the line buffer are as follows:

- A hit on a load in a line buffer is served from the buffer.
- A hit on a store updates the line buffer as well as the corresponding data cache bank.
- A miss on a load is served from a data cache bank, and forwards a copy of the cache block to the requiring line buffer.
- A miss on a store only writes the corresponding data cache bank, but no block transfer occurs.

## 5 Simulation Methodology

We use a modified version of a SMT simulator [12] to emulate the scalable SMT architecture proposed in [1]. It was built on top of the SimpleScalar [13] using PISA instruction set. We simulate a 4-context and 8-context SMT processor. The simulation parameters are summarized in the following table (taking an 8-context SMT processor as an example):

In the scalable SMT architecture, an unavoidable price is the overhead of the interconnection, which increases in complexity with the number of cache banks [1]. This interconnect can be a crossbar, a multi-stage network with uniform data cache bank access. Whatever it might be, the interconnect increases the access

**Table 1.** Architectural parameters used in simulations

|                |   |
|----------------|---|
| I-Cache        | 4 independent I-Cache banks<br>Each is 128KB,8-way associative,<br>64-byte lines,1 cycle access latency                         |
| D-Cache        | 16 banks<br>Each is 128KB, 8-way associative,<br>64-byte lines, total capacity:2MB,<br>7 cycle access latency                   |
| Line Buffers   | 8 lines per thread  |
| L2 Cache       | None  |
| Memory         | 100 cycles latency  |
| Function Units | 24 int-alu,8 fp-alu,4 int-mult,4 fp-mult  |
| Pipeline       | Fetch width=32,Decode width=32,<br>Issue width=32, Commit width=32,<br>RUU 128 entries per thread,<br>LSQ 64 entries per thread |

delay to the data cache from one to several clock cycles. Therefore the data cache delays in table 1 can be divided into three parts:

- Forward the address form one input port to the corresponding data cache banks through the interconnection network.
- Cache bank access.
- Forward the data to the corresponding physical destination register.

For benchmarks, we choose a subset of eight programs from SPEC2000 [7] to run as independent threads: 4 integer benchmarks (176.gcc, 197.parser, 175.vpr, 181.mcf) and 4 floating-point benchmarks (188.amp, 183.equake, 177.mesa, 179.art). For the simulation of the SMT-4 configuration, one workload composed of 4 benchmarks (2 integer and 2 floating-point) are used. For the simulation of the SMT-8 configuration, a single workload composed of the eight programs is used. Table 2 summarizes the composition of the workloads. The eight benchmarks are run in parallel until one of them terminates.

**Table 2.** Multiprogrammed workloads

|       |  |
|-------|--|
| SMT-4 | 175.vpr, 181.mcf, 188.amp, 179.art   |
| SMT-8 | 176.gcc, 197.parser, 175.vpr, 181.mcf,<br>188.amp, 183.equake, 177.mesa, 179.art |

## 6 Experimental Results

This section presents the simulation results in order to understand the quantitative effects of our partitioning scheme. First, we discuss how long the partition period should be. Then, we evaluate our partitioning scheme by running a set of eight benchmarks on a large-scale SMT system sharing a multi-banking data cache.

### 6.1 Partition Period

In this subsection, we research how long the period should be. Figure 2 illustrates the effects of the partition period on the IPC. Each curve represents a thread mix with the particular primary data cache.

As shown in the figure, the performance degrades when the partition period is either too short or too long. Short partition periods decrease the performance because of two reasons. First of all, there is an overhead to compute a new partition every partition period. Second, short partition period is bad for the estimation of marginal gains. We age our marginal gains every partition period by multiplying them by  $\mu$ . If the partition period is too short, the marginal gains will lose the past history very quickly.

On the other hand, if the partition period is too long, a partition can not reflect the dynamic changes in the program behavior as quickly as possible. It must lead to poor performance. However, program period does not change very quickly in our experiments. As a result, any partition period between 5000000 cycles and 100000000 cycles showed the best possible performance. If we have a partition period of fifty million cycles, the overhead of computing a new partition is less than 0.16%.

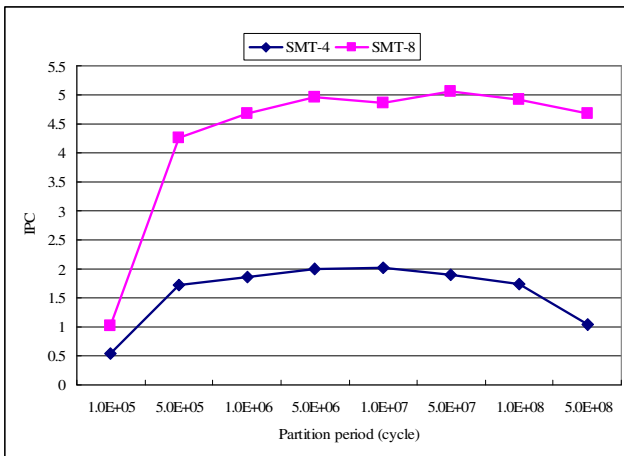
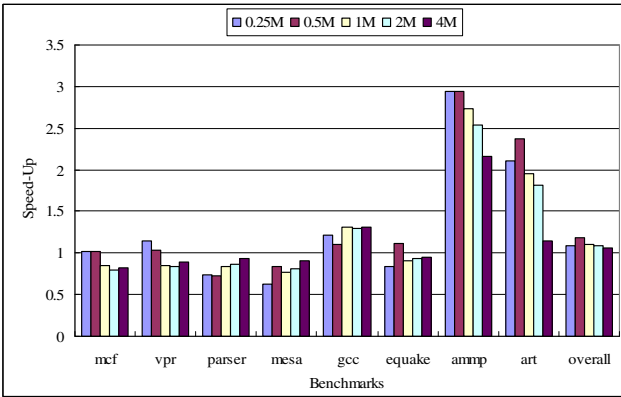


Fig. 2. The IPC as a function of the partition period

## 6.2 Effect of Partitioning on IPC

Now we study the performance of our cache partition scheme. For the simulation, we simulate a mix of eight benchmarks from SPEC CPU2000. The simulation compares the IPC of conventional policy (non-partitioned) and the IPC of our partitioning scheme. In the simulation, eight threads are run sharing the 16-bank data cache. The partition period  $\pi$  is set as fifty million cycles, and the weighting factor is  $\mu=0.5$ . There is an 80000 cycle overhead modeling the computation of a partition every period.

Figure 3 illustrates the speed-up of our partition scheme over the non-partitioned multi-banking data cache for both the individual performance and the overall performance. The results are shown for various data cache sizes, which range from 0.25MB to 4MB.



**Fig. 3.** The speed-up of the partitioning scheme over the non-partitioned multi-banking cache. The number of data cache banks is 16.  $\pi = 50000000$ ,  $\mu=0.5$ .

The simulation results demonstrate that the cache partitioning can improve the total IPC more or less. For a 0.5-MB 16-banked data cache running eight benchmarks, the speed-up of partitioning can arrive at 1.19. Our partitioning algorithm assigns more cache banks to the thread with a larger marginal gain, however, non-partitioned multi-banking cache blindly allocates the banks according to the cache misses. In addition, we can note that the individual performance of each benchmark may not get benefits from our partitioning scheme. The reason lies in our partitioning algorithm concentrates on the overall performance of the large-scale SMT system. At first, all the benchmarks share the whole data cache, but some of them will get less cache space after partitioning.

Figure 3 also illustrates the relationship between the cache size and the effectiveness of partitioning. For small caches, partitioning only helps rather little since the size of total workloads is too large compared to the cache size. However, in this case, changing the cache allocation can improve individual performance of some benchmarks significantly (such as ammp and art). At the other extreme,



partitioning can improve the cache performance slightly if the cache is large enough to hold most of the workloads, e.g. under the condition of a 4MB data cache. The range of cache sizes for which partitioning can improve performance depends on both the number of simultaneous threads and the characteristics of the threads. In our experiments, cache partitioning improves the performance in a large range of cache size.

Table 3 summarizes more detailed approximated IPC information for an 8-context SMT with a 16-bank data cache managed by the conventional policy and our partitioning algorithm respectively. The overall IPC is just the sum of the eight threads. The improvement is relative to the IPC of the conventional non-partitioned multi-banking cache. The table shows that partitioning algorithm improves IPC for all cache sizes up to 18.94%.

The experiment results also show that SMT should manage cache carefully. In the case of eight threads with a 2MB cache, SMT can achieve the overall IPC of 4.9095 from table 3. However, if you only consider one thread (gcc), its IPC is only 0.3850 whereas it can achieve an IPC of more than 1 alone [2]. The performance of a single thread is significantly degraded by sharing banks. Furthermore, the performance degradation by cache bank interference will become severe as the latency to the main memory increases. This problem can be solved by partitioning the banks of cache memory for some cases.

**Table 3.** Detailed comparison of IPCs between the non-partitioned and partitioned cache in an 8-context SMT

|                                   | bench-<br>marks | Cache (MB) |         |         |         |
|-----------------------------------|-----------------|------------|---------|---------|---------|
|                                   |                 | 0.25       | 0.5     | 1       | 2       |
| Non-partition IPC / Partition IPC | mcf             | 0.3539/    | 0.4021/ | 0.4293/ | 0.4584/ |
|                                   |                 | 0.3583     | 0.4098  | 0.3631  | 0.3637  |
|                                   | vpr             | 0.6919/    | 0.8213/ | 0.9016/ | 0.9144/ |
|                                   |                 | 0.7885     | 0.8426  | 0.7706  | 0.7672  |
|                                   | parser          | 0.9551/    | 0.8808/ | 0.8337/ | 0.8035/ |
|                                   |                 | 0.7041     | 0.6370  | 0.6953  | 0.6947  |
|                                   | mesa            | 0.7380/    | 0.6579/ | 0.5886/ | 0.5517/ |
|                                   |                 | 0.4592     | 0.5547  | 0.4495  | 0.4491  |
|                                   | gcc             | 0.3233/    | 0.3136/ | 0.2943/ | 0.2975/ |
|                                   |                 | 0.3903     | 0.3461  | 0.3852  | 0.3850  |
| equake                            | 0.7338/         | 0.6979/    | 0.6782/ | 0.6632/ |         |
|                                   | 0.6158          | 0.7795     | 0.6144  | 0.6167  |         |
| ammp                              | 0.1996/         | 0.2103/    | 0.2223/ | 0.2423/ |         |
|                                   | 0.5872          | 0.6187     | 0.6068  | 0.6157  |         |
| art                               | 0.4616/         | 0.4636/    | 0.5251/ | 0.5604/ |         |
|                                   | 0.9736          | 1.1013     | 1.0243  | 1.0175  |         |
| overall                           | 4.4573/         | 4.4474/    | 4.4731/ | 4.4916/ |         |
|                                   | 4.8771          | 5.2897     | 4.9092  | 4.9095  |         |
| Improve                           |                 | 9.42%      | 18.94%  | 9.75%   | 9.30%   |

## 7 Conclusion

Low IPC can be caused by long memory latency. We have found that SMT only exacerbates the problem when multiple executing threads share a multi-banking cache.

We have studied one method to reduce cache interference among simultaneously executing threads. Our on-line cache partitioning algorithm estimates the miss-rate characteristics of each thread at run-time, and dynamically partitions the cache among the threads at a cache bank granularity. We give the definition of marginal gains in multi-banking cache as a function of cache size and use bank counters to record their values approximately. Then a greedy search algorithm is used to find a proper partition that can reduce the total number of misses at a certain extent.

The hardware overheads for the partition proposed in this paper are small. A small number of extra counters are required. The counters are updated on cache hits. To actually partition the cache, we can use bank caching which require a small amount of bit vectors. In an 8-context SMT with a 16-banked data cache, sixteen 32-bit counters and one 16-bit vector are required for each thread. The total overheads are only 528 bytes.

The simulation results have shown that our partitioning algorithm can solve the problem of thread bank interference in multi-banking caches for a range of cache size. But threads that executing simultaneously should be selected carefully considering their memory reference behavior.

Moreover, our algorithm only allows the allocation of a whole bank a time, and there is no banks shared among the simultaneously executing threads. However, sharing a bank is essential to achieve high performance with bank granularity partitioning. For example, in large-scale SMT architectures, when the number of threads is close to the number of data cache banks. It is obvious that threads must share banks in order to use the cache more effectively. In this paper, we only consider the no sharing case, and it is our future work to consider the sharing case.

**Acknowledgment.** This work is supported by the National Natural Science Foundation of China under Grant No. 61272132, and the Fundamental Research Funds for the Central Universities of China under Grant No. WK011000020.

## References

1. Mudawar, M.F.: Scalable cache memory design for large-scale SMT architectures. In: Proceedings of the 3rd Workshop on Memory Performance Issues: In Conjunction with the 31st International Symposium on Computer Architecture, June 20, pp. 65–71 (2004)
2. Suh, G.E., et al.: Dynamic partitioning of shared cache memory. *Journal of Supercomputing* 28(1) (2004)
3. Tullsen, D.M., Eggers, S.J., Levy, H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism. In: Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy (June 1995)

4. Eggers, S.J., Emer, J.S., Levy, H.M., Lo, J.L., Stamm, R.L., Tullsen, D.M.: Simultaneous Multithreading: A Platform for Next-generation Processors. *IEEE Micro*, 12–18 (September/October 1997)
5. Marr, D.T., Binns, F., Hill, D.L., et al.: Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal* 6(1), 4–15 (2002)
6. Preston, R., et al.: Design of an 8-wide superscalar risc microprocessor with simultaneous multithreading. In: *IEEE International Solid-State Circuits Conference*, p. 344 (2002)
7. Henning, J.L.: SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer* (July 2000)
8. Chiou, D.T.: Extending the reach of microprocessors: Column and curious caching. Ph.D. Thesis, Massachusetts Institute of Technology (1999)
9. Chiou, D., Rudolph, L., Devadas, S., et al.: Dynamic Cache Partitioning via Columnization. CSG Memo 430. MIT
10. Stone, H.S., Turek, J., Wolf, J.L.: Optimal partitioning of cache memory. *IEEE Transactions on Computers* 41(9) (1992)
11. Wilson, K.M., Olukotun, K., Rosenblum, M.: Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors. In: *Proc. 23rd Ann. Int’l Symp. Computer Architecture*, pp. 147–157 (May 1996)
12. Gonçalves, R., Ayguadé, E., Valero, M., Navaux, P.: A Simulator for SMT Architectures: Evaluating Instruction Cache Topologies. In: *12th Symposium on Computer Architecture and High Performance Computing*, pp. 279–286 (October 2000)
13. Austin, T., Larson, E.: SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer* 35(2), 59–67 (2002)

# Scheduling Model of Virtual Machine Base on Task Type in Multi-core System

Hui-Xing Chen, Kenli Li, and Lin Shi

School of Information Science and Engineering, Hunan University,  
Changsha 410082, China  
{celine,lkl,shilin}@hnu.edu.cn  
<http://www.hnu.edu.cn>

**Abstract.** The traditional virtual machine scheduling algorithm does not fully consider the execution efficiency of parallel applications. When multiple virtual machines cooperate to execute the parallel computing tasks, the virtual machine monitor still allocates the physical CPUs by the time-division multiplexing method. That will lead the parallel tasks to be serialized and the efficiency degraded greatly. The modern chip multiprocessors platform involves several available computing cores, to meet the need of the concurrent execution of multiple virtual machines. In this paper, we proposed a dynamic scheduling strategy –CON-Credit scheduler, which helps to speed up the parallel applications in virtual environment with multi-cores or many cores system. The main feature of CON-Credit is to map the virtual CPU to the physical CPU directly, so the virtual machines involves parallel tasks can take fully advantage of the underlying hardware resources. More precisely, the CON-Credit algorithm dynamically allocated processor cores to the virtual domains according to the type of the application. For the parallel applications, CON-Credit chooses to schedule a bulk of physical CPUs at the same time to avoid the extra makespan of discrete dispatch in traditional virtual machine scheduling algorithm. The experimental results show that the CON-Credit algorithm improved the execution efficiency of the parallel application and optimized the overall performance of the virtual machine system.

**Keywords:** multi-core, scheduler, parallel, VMM, MapReduce.

## 1 Introduction

System virtualization technology once had a great vogue in the mainframe era. But it fades out as the descent of the computer hardware price and the popular use of the Personal Computer. At the beginning of the 21st century, the prosperity of server consolidation market and the proposition of cloud computing concept offer the opportunity of resuscitation to system virtualization technology. Nowadays the system virtualization technology has become an important infrastructure of cloud computing platforms, and there are more than one hundred affiliated companies and platforms, for instance, KVM, XEN, VMware,

Virtual PC, Hyper-V, and a large number of mature virtual machine platforms have been widely deployed and applied.

Just as the role of process scheduling algorithm in the operating system, the virtual machine scheduling algorithm is a key factor which affecting the overall performance of system virtualization. The virtual machine scheduling is implemented by the VMM(Virtual Machine Manager).At present, the popular VMM scheduling algorithm directly absorbed the essential idea of the operating system scheduler. It cant reflect the characteristics of the virtual machine platform: the virtual machine is a set of hardware and software, and it is far greater in granularity than the traditional process or thread on scheduling. The virtual machine contains an additional scheduling layer, thus forms a double-layer and two-dimensional scheduling architecture. Because the OS of virtual machine may be heterogeneous, uncooperative, and unfriendly to virtualization, the scheduling optimization only in the virtual machine manager layer may be not enough. Therefore, it is important to design a multi-dimension scheduling algorithm for virtualized multi-core processor system according to the very varied types of application. And the most important factor is to design the scheduling algorithm. In the single-core computer system, VMM, Domain0 and Domain U all run in a processor core. But the multi-core platform has the capability to simultaneously run multiple virtual machines.The traditional time-division multiplexing scheduling algorithm should transit to the hybrid schedule mode which combines both the time-division multiplexing and space-division multiplexing.All in all, traditional VMM scheduling algorithms mainly focus on the fairness of processor resources among the virtual machines, but havent considered multipath parallel features of the multi-core processors. Therefore, it leads to the decline in the execution efficiency of the parallel tasks, when request tasks contains some parallel tasks, the default scheduling algorithm in VMM cant fit to the characteristic of the underlying hardware, and fully utilize the hardware.

In this work, we take the virtual machine monitor XEN as an example, design a virtual machine schedule algorithmCON-Credit for the parallel environment. The CON-Credit exploits the virtual machine requests-assign mechanism, and dynamically adjusts the resource allocation pattern; the CPU resources are allocated to the different tasks, according to the current resource status and task characteristics. Without enforce a serious impact on other tasks, it ensures the distribution execution in space and concurrent execution in time of the parallel tasks, and improves the overall performance of the multi-core platform.

## 2 Background

### 2.1 Xen's Credit Scheduler

Xen[1] is a virtual machine monitor based on the open source Linux kernel, it comprises two parts: the hypervisor which located in the highest privilege level management monitor and the frontend/backend driver in the virtual domain. On recent versions of Xen, the Credit scheduler[2] is used by default. For this

scheduler, each domain has two properties associated with it, a weight and a cap. The weight determines the share of the PCPU time that the domain gets, whereas the cap represents the maximum that the domain can get. In contrast, the cap is an absolute value, representing a proportion of the total CPU that can be used. The credit scheduler manages multiple PCPUs and distributes CPUs time to each VCPU equally, which in order to realize the load balance. However, it focus mainly on the fairness, ignores the specific characteristics of different tasks, which leads to the inefficiency for the collaborative workload in the multi-core system.

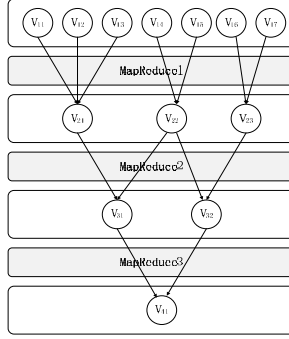
## 2.2 MapReduce

MapReduce is a distributed computing model by Google. It is simply represented in two functions: Map and Reduce. Map function was written by the user, processes a key-value pair to generate a set of intermediate key/value pairs. Reduce function was written by the user too, which merges all intermediate values associated with the same intermediate key. The Map usually contains independent operation adapt to the characteristics of the large-scale parallel and distributed computation. In MapReduce clusters consist of a large number of nodes for redundancy and fault tolerance, each node in the cluster is assigned a certain number of data chunks, which are in turn duplicated on several nodes based on a distributed file system[3][4]. In the Map and Reduce operations, the Map is a highly parallel, and Reduce is dependent on the result of Map operations. The parallelism and balance of the Map operation determine the overall efficiency of the MapReduce program. The MapReduce is also widely used in the system-level virtual machine platform, such as CLOUDLET [5] and Cloud BLAST [6] and other projects. They combined the flexibility of the virtual machine and the distributed parallelism of MapReduce framework, so the MapReduce can easily be deployed in the cloud platform. It is important for cloud platform to support this kind of parallel computing.

## 3 Scheduling Strategies and Analysis

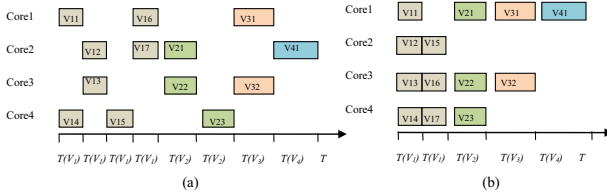
### 3.1 Motivations and Modeling

In this paper, we take the *MapReduce* structure model for an example, which is shown in the figure 1, where nodes  $V = V_{11}, V_{12}, \dots, V_{21}, V_{22}, \dots, V_{i1}, \dots, V_{ij}$  denote the set of different tasks. Assume that the computation time of all the nodes at the same level is the same. The processing time of the  $i$ -th level task is  $T(V_i), T(V_i) = T(V_{i1}) = T(V_{i2}) = \dots = T(V_{ij}), i = 1, 2, 3, \dots, I$ . Assume there are 4 cores available to execute this task. If a node  $V_{ij}$  has already started in the core, we define the node can occupy the processor core resources until its processing completed. The target of scheduling is to find out a dispatch scheme to guarantee the minimum of total execution time.



**Fig. 1.** The Structure model of *MapReduce*

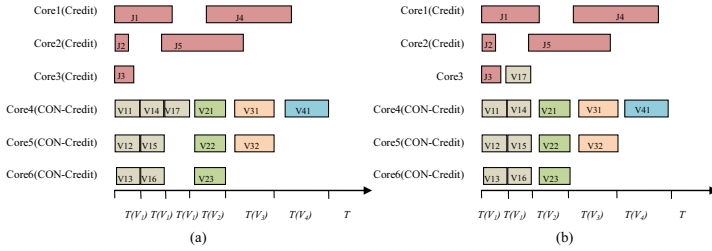
The main drawbacks of the existing default credit scheduler when applied to *MapReduce* workloads are as follows. Figure 2(a) shows the scheduling time of the nodes on the 4 cores physical machine under *MapReduce* workloads. The total execute time is  $T = 4 \cdot T(V_1) + 2 \cdot T(V_2) + T(V_3) + T(V_4)$ .



**Fig. 2.** (a) A possible order distribution of the *MapReduce* workloads with Credit algorithm. (b) A possible order distribution of the *MapReduce* workloads with concurrent scheduling algorithm.

The concurrent scheduling algorithm uses a novel mechanism to optimize the Credit algorithm. For a node, the tasks of next level must wait for all the tasks of the previous level were disposed, before it is implemented. To make full use of processor resources, a core can be adopted to maximize the parallel process the various tasks of the same level. The total time  $T = 2 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$ . A order distribution of the *MapReduce* workloads with concurrent scheduling algorithm base on this idea is as shown in the figure 2(b).

Further we will discuss the situation that hybrid types of tasks run simultaneously. Assuming that the *MapReduce* model above and the other five common tasks send request at the same time, there are six physical processor cores available and fix 3 processor cores to each type of task before execute. The total time  $T = 3 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$ , a possible schedule order distribution of the hybrid tasks as shown in the figure 3(a).



**Fig. 3.** (a) A possible order distribution of the *MapReduce* workloads and common tasks with Credit schedule. (b) A possible order distribution of the *MapReduce* workloads and common tasks with concurrent scheduling algorithm.

If we design a mechanism to adjust the allocation of resources dynamically according to the number of tasks for the different types of tasks, this allows us to make full use of the resources, improve the overall real-time and throughput, and reduce the total execution time. At this point, due to the *core3* has been idle after performing task  $J_3$ , if its resources assigned to  $V_{17}$  dynamically, the total time  $T = 2 \cdot T(V_1) + T(V_2) + T(V_3) + T(V_4)$ . Now, a schedule order distribution of the hybrid tasks as shown in the figure 3(b).

From the above scheduling situation, we can see that the default credit algorithm for parallel tasks has obvious defects, but the concurrent scheduling algorithm in multi-core system show better performance. In addition, the analysis of the hybrid tasks shows that dynamic scheduling give more conducive to improve the resource utilization and application performance than static scheduling, thereby enhancing the throughput of the entire system, reducing the minimum total execution time the event(*MAKESPAN*).

### 3.2 Scheduling Framework

Based on the above analysis, this paper proposes CON-Credit scheduling model(concurrent credit scheduler) that according to the types of the tasks to divide multi-core processor core dynamically. The task status monitoring and scheduling decision-making module are added in this model on the basis of the original VMM scheduler structure. The former collect time and task status information in the process of device access by reading the data in the event channel, while the latter process the information that have collected in real-time, and dispatch cores resources to each VCPU dynamically to deal with the tasks of corresponding types, make a scheduling decision when some conditions are satisfied.

The four main parts to realize this scheduling model are as follows:1) Cores are divided into the COM-Core and CON-Core dynamically, COM-Core corresponding to ordinary task and CON-Core corresponding parallel tasks; 2) The VMM scheduler dispatch tasks according to the application type in the different sets; 3) Task status monitoring and real-time decision-making. 4) Scheduling virtual machine in parallel and collaboratively. The separation of CON-core and



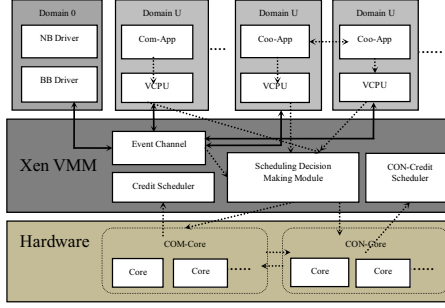


Fig. 4. The overall structure of the scheduling strategy model in VMM

COM-Core is used to reduce the time of the physical CPU resource allocation and to respond events request quickly. The task of condition monitoring and dynamic division of decision-making are used to reduce the performance penalty of context switching of the different task types. Parallel and collaboration of the virtual machine are conducive to the separation between the parallel tasks, and the implementation of the synchronous collaboration to achieve the task parallel processing. The overall structure of the scheduling model was shown as figure 4.

### 3.3 CON-Credit Strategy Analysis

Taking into account  $|P|$  processor cores system, use  $P = P_1, P_2, \dots, P_{|P|}$  to represent all the multi-core processors. Let  $J = J_1, J_2, \dots, J_n$  means be the set of the common tasks which have  $n$  tasks. For each task  $J_i \in J$ ,  $T(J_i)$  means the processing time of task  $i$ . Besides, with  $V = V_{11}, V_{12}, \dots, V_{21}, V_{22}, \dots, V_{i1}, \dots, V_{ij}$  denotes the set of the nodes of parallel tasks, and  $m$  nodes.  $T(V_{ij})$  denotes the process time of node  $i$ , and let  $W = J \cup V_i$ . Assume the set of the  $|D|$  VC-PUs is  $D = D_1, D_2, \dots, D_{|D|}$ . For a running time, there is  $|W_R| \leq |D_R| \leq |P_R|$ , and  $|W_R|, |D_R|, |P_R|$  denotes the number of tasks are running at the moment, VC-PUs and cores respectively. In the multi-core virtual machine monitor system, the task scheduling includes allocation of the tasks in VCPUs to the *domainU*, denoted by  $D_k = \lambda(W_i)$ , and the VMM allocate the cores to the each *domainU*, denoted by  $P_j = \mu(D_k)$ . So  $P_j = \mu(\lambda(W_i))$  denotes that allocate the processor core  $j$  to the node  $i$ . ((((((())) We make the weight of PCPU process the common types of task is  $\varpi(P_J)$ , and the parallel types of task is  $\varpi(P_V)$  respectively, then  $\varpi(P_J) + \varpi(P_V) = 1$ . Of course, the cores that perform two types of tasks can interconversion according to the value of  $\varpi(P_J), \varpi(P_V)$ .

When the event request, allocate processor core resources to the various VC-PUs dynamically depending on the value of  $\varpi(P_J), \varpi(P_V)$  to. Among the  $|P|$  processor cores system, we assume that each half of the processor cores are assigned to the two types of tasks. In operation, record the number of two types of tasks and the processing time required to be processed within a time unit in real-time by state monitoring mechanism. We assume the number of common tasks is

$C_1$ , then the total processing time can express as  $\sum_1^{C_1} T(J_{K_1}), k = 1, 2, 3, \dots, C_1$ . At the same time, we assume the numbers of parallel tasks type are  $C_2$ , then the total processing time can expressed as  $\sum_1^{C_2} T(V_{K_2}), k = 1, 2, 3, \dots, C_2$ . So, the ratio of the task duration is denoted by  $R = \frac{\sum_1^{C_1} T(J_{K_1})}{\sum_1^{C_2} T(V_{K_2})}$ , and let

$\frac{\varpi(P_J)}{\varpi(P_V)} = R = \frac{\sum_1^{C_1} T(J_{K_1})}{\sum_1^{C_2} T(V_{K_2})}$ ,  $\varpi(P_J) + \varpi(P_V) = 1$ . As a result, the number of processor cores are assigned to the type of the common tasks can be defined as:

$R_J = \lfloor \varpi(P_J) \cdot |P| \rfloor = \left\lfloor \frac{\sum_1^{C_1} T(J_{K_1})}{\sum_1^{C_1} T(J_{K_1}) + \sum_1^{C_2} T(V_{K_2})} \cdot |P| \right\rfloor$ . The number of processor cores are assigned to the type of the parallel tasks can be defined as:

$R_V = \lceil \varpi(P_V) \cdot |P| \rceil = \left\lceil \frac{\sum_1^{C_2} T(V_{K_2})}{\sum_1^{C_1} T(J_{K_1}) + \sum_1^{C_2} T(V_{K_2})} \cdot |P| \right\rceil$ . Here  $|P| = R_J + R_V$ .

In the CON-Credit algorithm, when a task node  $W_i$  has begun executed in the processor core, it will occupy the core until it finished. The question is how to find out a task scheduling strategy that has minimum total execution time for the request event so as to improve the real-time performance and throughput. When an event sends a request, each task nodes are assigned to each processor core according to  $\mu(\lambda(W_i))$ ,  $T(\mu_{W_i}^{P_j})$  denote the process time of the task  $W_i$  that assigned to the processor core  $P_j$ . If the total number of tasks that has already completed is  $K$  within the time  $T$ , then the throughput is:  $\varphi = \frac{K}{T}$ , and:

$$T = \max \left\{ \sum_J T(J_i), \sum_V T(V_i) \right\} + \sum_{i=1}^{n+m} T(\mu_{W_i}^{P_j}). \quad (1)$$

When the parallel task comes, each node is scheduled according to their level or their sequential dependencies. The tasks at the same level are allocated to each virtual machine in accordance with the  $\lambda(V_i)$ , and corresponds to the processor core for concurrent processing in accordance with the  $\mu(\lambda(V_i))$ . Then allocate the appropriate number of processors to process the task nodes at each level according to the value of  $\varpi(P_J), \varpi(P_V)$ , and disposed of these tasks in a period of time as much as possible to maximize concurrent processing of parallel tasks. When  $R_V < |V_i|$ , the processor cores resources are not enough. At this time, assign the  $R_V$  processor cores to  $R_V$  nodes in accordance with the function  $\mu(\lambda(V_i))$ , and detect dynamically if the task on the virtual machine completed or not. If finished, transfer to the other node at the same level that waiting for processing. When all the nodes at the same level have finished, then execute the tasks of each node of the next level. The events the total execution time  $T(V)$  is as follow:  $T(V) = \sum_{i=1}^I k_i \cdot T(V_i) + \sum_{i=1}^I \sum_j T(\mu_{V_{ij}}^{P_n}) = \sum_{i=1}^I \left\lceil \frac{j_i}{R_V} \right\rceil \cdot T(V_i) + \sum_{i=1}^I \sum_j T(\mu_{V_{ij}}^{P_n})$ ; Where  $k_i = \left\lceil \frac{j_i}{R_V} \right\rceil$ ,  $k_i$  denotes the task allocate to the  $i$ -th level, times of  $T(V_i)$  time period on a processor cores is needed to complete all of the tasks in the  $i$ -th level. So the total execution time for such events is as follows:  $\min T(V) = \min \sum_{i=1}^I \left\lceil \frac{j_i}{R_V} \right\rceil \cdot T(V_i) + \min \sum_{i=1}^I \sum_j T(\mu_{V_{ij}}^{P_n})$ .

Therefore, when the hybrid tasks request, the objective function is to minimize the execution time:

$$\min T = \max \left( \min T(V), \sum_J T(J_i) + \min \sum_{i=1}^n T \left( \mu_{J_i}^{P_n} \right) \right). \quad (2)$$

Subject to: The time that task node  $V_{ij}$  obtained is decided by  $\varpi(V_{ij})$ , the  $k$ -level task nodes scheduled before the  $r$ -level task nodes; here  $k < r$ , and inequality  $|W_R| \leq |D_R| \leq |P_R|$  must be established.

### 3.4 The Algorithm Flow

The CON-Credit algorithm flow chart was shown as figure 5. In the multi-core processors virtual environment monitor system, the algorithm take VCPU for scheduling unit, each VCPU associated with the corresponding virtual machine. At the initial stage of this scheduling algorithm, it check the target VCPUs running state and the queue position, judge the request event type and obtain the proportion of the  $R$  value through the state decision module, and assign the core number of various types that needed according to the  $R$  value. During the operation, the state decision module monitor the ratio of the time required for each task type in dynamically, and generate the corresponding distribution value, reduce or increase the core number of the concurrent events needed according to the value to make more common tasks or concurrent events can be operated. In operation, if a new task request, the function  $\mu(\lambda(W_i))$  will generate a new allocated value  $R_j, R_v$ , reallocate processor resources to each virtual CPU. Meanwhile, the scheduling decision module should judge the type of event and select scheduling strategy. For the concurrent event, the CON-Core is selected. Otherwise, the default COM-Core is assigned.

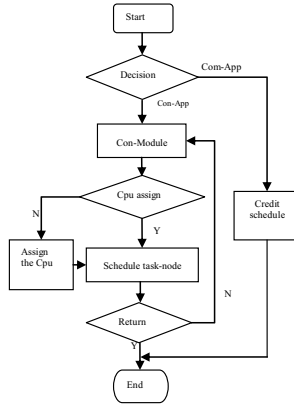


Fig. 5. The model flow chart

## 4 Performance Evaluation

### 4.1 Experiment Environment

The hardware configuration of the experimental platform is Intel eight-core Xeon 7550 processors, the Seagate 1TB IDE hard disk, DDRII-800 8GB memory, and the RTL 8139D 200Mbps Ethernet card. All experiments used Xen 4.1.2 on the Fedora16 operating system with Linux 64bit 3.10.17 kernel. All virtual machines are running with Fedora 16 with kernel 3.10.17. At the same time we build a executable parallel programming environment with the virtual machines. To study the performance comprehensively, we analyze the performance of CON-Credit scheduling algorithm in three ways:Efficiency: What is the performance profit CON-Credit algorithm can get comparing with the traditional Credit algorithms.Adaptability: How the CON-Credit algorithm work in the hybrid task mode.Scalability: How the number of virtual machine will give an influence on the CON-Credit scheduling performance.

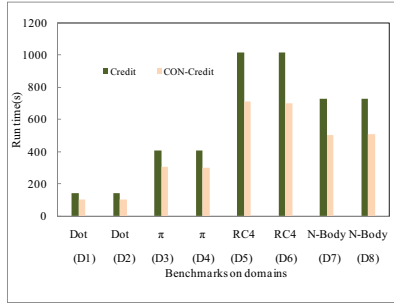
### 4.2 Benchmarks

Now, there are four *MapReduce* benchmarks(*Dot product*,  $\pi$  *Computation*, *RC4 Key Search* and *N – body problem*)[7]have been constructed to demonstrate the applicability of CON-Credit schedule framework.*Dot product*:We perform the multiplications in the Map function and additions in the Reduce function. $\pi$  *Computation*: The classic Monte Carlo simulation is used to approximate the value of  $\pi$ . For  $N$  paths, the output of the Map function is a stream of binary values $\in 0, 1$ . The Reduce function is the addition and  $\pi$  is computed by multiplying the reduced value by  $\frac{4}{N}$  on the host computer.*RC4 Key Search*: The Map function input is an index indicating the position to start the search. The reduce function, implemented on the host, checks the return value and outputs the correct key if found.*N – body*:In our test, the input to the Map function is the current information for the  $n$  particles and the particle index. It is the reduce functions responsibility to fresh each particle new state according to the formula associated[8].

### 4.3 Performance Measurement and Analysis

Now, we take the above 4 typical*MapReduce* programs and serial program (such as *Rank* sorting algorithm and *gcc* compile test in SPEC CPU2000) as the example to verify our proposed virtual machine scheduling algorithm and compare its performance with traditional one.

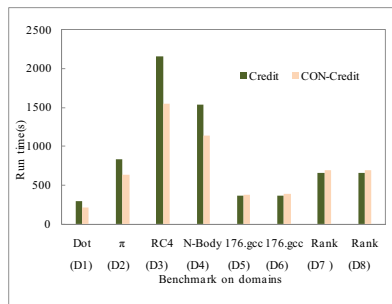
The first experiment tests per the formance of the distributed *MapReduce* applications which ran in eight virtual machines. Each domain is configured with 4 VCPUs and only one *MapReduce* task runs on each of them. The result is depicted in Figure 6. Compare with the traditional scheduler the total execution time of four benchmarks reduced by 28.17%, 25.86%, 30.47%, 30.39% respectively under the CON-Credit algorithm.



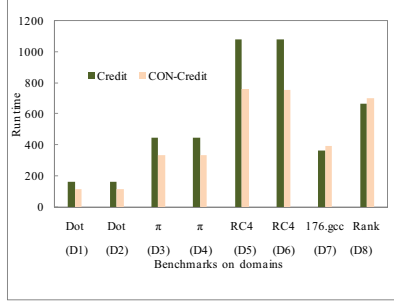
**Fig. 6.** The *MapReduce* benchmarks run simultaneously on 8 domains, respectively

The second experiment measures the overall performance of CON-Credit with hybrid workload, server the parallel and serial task at the same time. In this test, eight domains are created on XEN, and each domain is configured with 4 VCPUs. The figure 7 show the case then 4 domains run the *MapReduce* programs while the others involve only serial programs. According to Figure 9, the proposed CON-Credit algorithm show better performance than traditional algorithm, the execution time of dot product is decreased by 25.61%,  $\pi$  Computation is decreased by 23.06%, *RC4 Key Search* is decreased by 28.60%, *N-body* is decreased by 26.27%. However, the execution time of serial programs *Rank* is increased by 5.33%, and *176.gcc* is increased by 9.07%. Because the improved VMM scheduler has added a scheduling decision module, all the tasks need the scheduling decision module to allocate physical processor resources. That leads to the time of serial tasks has a slight increase. This is acceptable in the large environment.

The figure 8 show the another case, up to 6 domains are configured to run *MapReduce* programs and the other 2 domains severd for the serial programs. The overall execution time of the benchmarks with two schedulers is depicted. According to Figure 10, the CON-Credit algorithm speeds up the *dot product*,



**Fig. 7.** Benchmarks run simultaneously on 8domains respectively, 4domains are used to *MapReduce* programs and the others are used to serial programs, and the iterations are set to  $10^9$



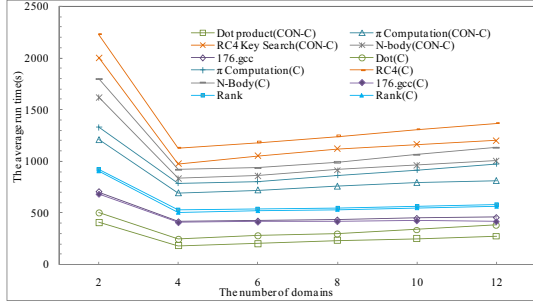
**Fig. 8.** The benchmarks run simultaneously on  $8domains$  respectively,  $6domains$  are used to *MapReduce* programs and the other  $2domains$  are used to serial programs, and the iterations are set to  $10^9$

$\pi$  *Computation*, and *RC4 Key Search* by 26.23%, 24.94%, 29.70% respectively. However, the execution time of *Rank* serial programs is increased by 5.25%, and *176.gcc* is increased by 6.20%. It is also acceptable in the large environment as the same reason of before said.

In the 3rd experiment, we create several virtual machines to stimulate the cluster environment which be used to test the hybrid scheduling performance, and the *MapReduce* parallel tasks and ordinary serial tasks are scheduled simultaneously. In order to find out how the number of domains influences the overall performance of scheduling algorithm, we test the result with different cores and domains. Each domain is equipped two VCPU, and the test cases are set to the 4 *MapReduce* benchmarks and two serial tasks. In this test, the CON-Credit algorithm show better performance than the traditional scheduling algorithms for parallel tasks, and adapt to the hybrid mode which involves both the parallel and serial tasks.

The results of the experiment show that the total execution time changed with the growing number of domains, and the average time of all tasks with the corresponding number of domains are all shown on figure 9. The experiment demonstrates the relative advantage of CON-Credit algorithm for *MapRrduce* tasks than the default scheduling algorithm. With the number of virtual machines increased, the execution time of *dot product* is decreased by 9.18%, 18.03%, 13.62%, 12.75%, 13.08%, 14.81%; the execution time of  $\pi$  *Computation* is decreased by 9.31%, 16.82%, 9.98%, 12.18%, 13.43%, 15.43%; the execution time of *RC4 Key Search* is decreased by 10.08%, 13.55%, 10.74%, 9.81%, 10.71%, 11.99%; the execution time of *N – body* is decreased by 9.68%, 14.01%, 9.81%, 9.79%, 10.55%, 11.40%. However, the execution time of *rank serial programs* increased by 1.19%, 5.45%, 4.42%, 3.62%, 3.37%, 2.94% in proper order, and *176.gcc* increased by 2.79%, 1.97%, 2.43%, 3.60%, 5.18%, 7.91% in proper order.

As the Figure 9 shows the whole system get the maximum performance when the number of domains is configured to 4. The reason is there are only 8 physical cores in the testbed, when 4 domains are running on top of the system there are eight VCPUs on XEN in total because two VCPUs are set to each domain.



**Fig. 9.** 4 *MapReduce* benchmarks and 2 common tasks are run simultaneously on domains, and the *MapReduce* benchmarks iterations set to  $10^9$

The communication overhead among domains in the Reduce process is smaller relative to more domains. When the number of VCPUs is equal to the number of physical processor cores, the execution time of Map process is reduced, and system reach a balance point. When the number of VCPUs is great than 8, the extra communication overhead will increase dynamically and fade out the benefit of more VCPUs. Besides, with the increase of domains, the increase velocity of the time of CON-Credit is gentle and trends to a steady value, but the default algorithm still keep in an increasing trend. For serial task, when four VCPUs in two virtual machines are created, the efficiency gets its peak value. However, when the number of domains increases, the time spend in scheduling make decision module also grow. The scalability experiment shown on the figure 9 shows that even take this element into account, the CON-Credit algorithm still exhibit good scalability in large scale system.

## 5 Related Work

In this section, we present the related work on scheduling in the XEN VMM. Several extensions to Xens Credit scheduler are proposed to improve I/O performance, by adding a highest priority status named BOOST. The BOOST related credit scheduler sort the RUN queue based on their remaining credits, and tickling the scheduler when events are sent. The [9] has largely focused on improving the efficiency of I/O operations and has not explored the impact of the scheduler on I/O unrelated task. Scheduler improvements for I/O are likely to also benefit these innovative I/O designs. Kim et al[10] proposed a task-aware virtual machine scheduling mechanism based on inference techniques using gray-box knowledge. The proposed mechanism infers the I/O-boundness of guest-level tasks and correlates incoming events with I/O-bound tasks. Chuliang Weng et al[11] analyzed the CPU scheduling problem and presented a hybrid scheduling framework for the CPU scheduling in the virtual machine monitor. Lee et al[12] have identified the area of soft real-time application domains and the performance problems they encounter in virtualized environments. L Shi et al [13]

proposed vCUDA, a general-purpose graphics processing unit computing solution for virtual machines. vCUDA allows applications executing within VMs to leverage hardware acceleration, which can be beneficial to the performance of a class of HPC applications. Philip M. Wells et al. [14] proposed a simple hardware technique to detect when a VCPU is spinning, without requiring any software modification, and preempt that VCPU in favor of one which is making forward progress. Hui Kang et al. [15] designed and implemented the MRG scheduler, a new Xen scheduler for VMs running MapReduce workloads. The scheduler facilitates MapReduce job fairness by introducing a two-level group credit based scheduling policy. Chuliang Weng et al [16] proposed an adaptive dynamic co scheduling method to mitigate the problem, while avoiding unnecessary overhead for co scheduling, and implement a prototype ASMan. Neither of above work combines the strength of MapReduce framework with the flexibility of virtual machine technology. We propose a schedule algorithm, CON-Credit , which is a fundamental approach for adapting to the diversity of VMs in the cloud platform, to decide when and how to map VCPUs to cores wisely. Moreover, the hybrid scheduling framework supports distributing core resources among VCPUs based on demand, as well as distributing equally.

## 6 Conclusion

More and more parallel computing tasks have been deployed in the cloud computing and virtualization platform. In this paper, we designed and implemented a scheduling algorithm named CON-Credit that make full use of the inherent characteristics of multi-core architecture, since the traditional virtual machine scheduling algorithm does not adapt to the parallel task scheduling. The CON-Credit achieved the dynamic classification and matching of the physical CPUs, and improved the execution efficiency of the *MapReduce* task through cooperative scheduling between the ordinary tasks and parallel tasks. The experiments prove that the algorithm worked well for the parallel task scheduling and mixed task scheduling, and have strong adaptability and scalability. For now the CON-Credit algorithm is only implemented in XEN virtual machine platform. In future it will be ported to other VMM platform like KVM and Hyper-V. In addition, the cloud computing platform may involved a large number of heterogeneous cores, that is a good candidate to exhibit the flexibility and scalability of CON-Credit. How to combine the parallel computing tasks and heterogeneous virtual computing environment, and design a more adaptable virtual machine scheduling algorithm is still a topic worth further studying.

## References

1. Chisnall, D.: The definitive guide to the Xen hypervisor, pp. 222–224 (November 2007)
2. Credit Scheduler, <http://wiki.xensource.com/xenwiki/creditscheduler>



3. Ibrahim, S., Jin, H., Lu, L., Qi, L., Wu, S., Shi, X.: Evaluating MapReduce on Virtual Machines: The Hadoop Case. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) *CloudCom 2009*. LNCS, vol. 5931, pp. 519–528. Springer, Heidelberg (2009)
4. Ghemawat, S., Gobiuff, H., Leung, S.-T.: The Google file system. In: *SOSP*, pp. 29–43 (2003)
5. Matsunaga, A., Tsugawa, M., Fortes, J.: CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications. In: *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pp. 222–229 (2008)
6. Ibrahim, S., Jin, H., Cheng, B., Cao, H., Wu, S., Qi, L.: CLOUDLET: towards MapReduce implementation on virtual machines. In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, pp. 65–66 (2009)
7. Yeung, J.H.C., Tsang, C.C., Tsoi, K.H., et al.: Map-reduce as a Programming Model for Custom Computing Machines. In: *16th International Symposium on Field-Programmable Custom Computing Machines*, pp. 149–159 (2008)
8. Tsoi, K.H., Ho, C.H., Yeung, H.C., Leong, P.H.W.: An arithmetic library and its application to the n-body problem. In: *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 68–78 (2004)
9. Kim, H., Lim, H., Jeong, J., Jo, H., et al.: Task-aware Virtual Machine Scheduling for I/O Performance. In: *Proceedings of the 4th International Conference on Virtual Execution Environments (VEE)*, pp. 101–111 (2009)
10. Weng, C., Wang, Z., Li, M., Lu, X.: The Hybrid Scheduling Framework for Virtual Machine Systems. In: *Proceedings of the 4th International Conference on Virtual Execution Environments (VEE)*, pp. 111–120 (2009)
11. Ongaro, D., Cox, A., Rixner, S.: Scheduling I/O in virtual machine monitors. In: *Proceedings of the 4th International Conference on Virtual Execution Environments (VEE)*, pp. 1–10 (2008)
12. Lee, M., Krishnakumar, A., Krishnan, P., Singh, N., Yajnik, S.: Supporting soft real-time tasks in the XEN hypervisor. In: *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 97–108. ACM (2010)
13. Shi, L., Chen, H., Sun, J.H., Li, K.L.: vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. *IEEE Transaction on Computers*, doi:10.1109/TC.2011.112
14. Wells, P.M., Chakraborty, K., Sohi, G.S.: Hardware support for spin management in overcommitted virtual machines. In: *Proc. of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT 2006)*, Seattle, Washington, USA, September 16-20 (2006)
15. Kang, H., Chen, Y., Wong, J.L., Wu, J., Sion, R.: Enhancement of Xen’s Scheduler for MapReduce Workloads. In: *HPDC 2011*, San Jose, California, USA, June 8-11 (2011)
16. Weng, C., Liu, Q., Yu, L., et al.: Dynamic Adaptive Scheduling for Virtual Machines. In: *HPDC 2011*, San Jose, California, USA, June 8-11 (2011)

# Dynamic Pricing Strategy for Cloud Computing with Data Mining Method

Xing Wu<sup>1,2,3</sup>, Ji Hou<sup>1</sup>, Shaojian Zhuo<sup>1</sup>, and Wu Zhang<sup>1</sup>

<sup>1</sup> School of Computer Engineering and Science,  
Shanghai University, Shanghai, China

<sup>2</sup> State Key Laboratory of Software Engineering,  
Wuhan University Wuhan, China

<sup>3</sup> State Key Laboratory for Novel Software Technology,  
Nanjing University Nanjing, China  
xingwu@shu.edu.cn

**Abstract.** Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a metered service over a network (typically the Internet). To maximize the revenue of cloud service providers, a dynamic pricing model is proposed, which consists of two data mining methods. The first data mining method is the k-means algorithm with which historical data are classified into groups. The second one is Bayes decision that can forecast the trend of user-preferred cloud service packages. In proposed pricing model, BP-neutral network is applied to forecast the price which can maximize the revenue. Compared with the static pricing model and the models without k-means algorithm, the proposed model can meet customers' demand better and outperform them in revenue maximization.

**Keywords:** maximum revenue, various resources, dynamic pricing, Cloud Computing.

## 1 Introduction

Nowadays, more and more users are concentrating on Clouds Computing systems. To cater to these demands, some corporations have implemented cloud computing to go along with the new technology, such as Google and Amazon. Google engaged in the development of cloud computing technology and proposed Google File System (GFS), MapReduce and Bigtable[1]. While Amazon EC2 (Elastic Compute Cloud) has successfully gain profit from its cloud services. To maximize the revenue [2] of cloud providers, a proper pricing strategy is indispensable.

Other corporations have developed their own cloud platforms too. For instance,

As the number of cloud users varies from time to time and the demands of users fluctuate, a dynamic pricing model to manage the revenue is more effective than the static pricing model. Thus a dynamic pricing model implementing the neutral

networks to make a forecast according to requests made by users is proposed to maximize the revenue of cloud providers. In the proposed model we also forecast the user-preferred pattern [3] of cloud service packages using the Bayes decision. Furthermore, some data mining methods are utilized to classify the customers based on their behaviors.

## **2 Related Works**

### **2.1 The Definition of Cloud Computing**

Cloud Computing has a strong relationship with Grid Computing. Although there are so many relations between these two kinds of styles, it is the differences that exist. Although there are lots of definitions for cloud computing, we still don't have a universally agreed definition. A popular definition for cloud was proposed by Ian Foster as follows: A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet [4]. As can be seen in this definition, the most important developed trends in cloud that the Grid Computing doesn't have are virtualized, dynamically-scalable platforms and services which are delivered on demand to external customers over the Internet. Since the resources have been virtualized as commodities, the transactions in cloud computing should follow the law of economics. Compared with Grid computing, the services on Cloud Computing can even be virtualized as commodities which makes the cloud computing much easier to be charged.

Amazon EC2 charges for not only the cloud infrastructures but also cloud services. An elastic model is proposed in the EC2 which means cloud can be seen as resources without a boundary [5]. To extend this concept, cloud computing should provide services without a boundary and thus should be charged dynamically.

### **2.2 The Revenue Management of Cloud Computing**

As the pricing becomes an indispensable part of the cloud computing, it is a problem how to make the profits maximum. Some scholars analyzed the cost-benefit of Cloud Computing [6] and believed cloud computing has advantages in following aspects: the reliability, the pricing and the response time concerned with the service quality. With the cloud computing becoming widely used, pricing could play the vital role to make the revenue maximum which is related to the revenue management of any company. The term revenue management is most commonly used for the theory and practice of maximizing expected revenues by opening and closing different fare classes or dynamically adjusting prices for products. Putting this conception into clouding pricing is meaningful. In the revenue management of cloud computing, a dynamic pricing strategy is required to maximize the revenue.

### 2.3 The Dynamic Pricing of Cloud Computing

There has been some scientific research concerned with dynamic pricing models for cloud computing. Arun Anandasivam and Marc Premm proposed both a static pricing model and a dynamic pricing model which utilizes a heuristic algorithm to forecast the pricing [7]. In this research, the static model can't be adjusted to changing situations, whereas the proposed dynamic pricing model adapts to the real cloud market. However the proposed dynamic pricing model only contained a math model without an implementation and not available for multiple kinds of cloud resources. Furthermore the model didn't make revenue maximum. So we make an improvement for multiple kinds of cloud resources.

There is a big difference between one kind of resource and multiple kinds of cloud resources for dynamic pricing. For multiple kinds of cloud resources, the relationship among these resources should be considered. To simply the situation of multiple cloud resources, "combo" and "package" are defined. Combo is the combination selected by customers, while package is the combination predetermined by company previously.

Furthermore, a data mining method is used to classify the historical data which is used for forecasting. Different people have different demands. With this method the customers can be put into different classifications, so that a more specific forecast can be made using the sorted data.

When a forecast for price adjustment is made, there should be a time scale. In [7], time interval is the second-scale period which is too precise to realize it. Thus time scale in hours is used in our pricing model. Before a forecast for the combos is made, a data mining method is proposed to classify the various resources into several classifications as different customers have different requests. Then the most similar historical data can be utilized to forecast future usage with back-propagation neural network approach [8]. As the fluctuation of price has no regular patterns, the back-propagation neural network approach is appropriate for this situation [9]. Back-propagation neural network is effective when there is no obvious rule [10]. After the forecast of back-propagation neural network, a matrix of prices and amount of requests which are accepted by customers will be gotten. Then an equation of the maximum revenue can be acquired. After we seek partial derivative for each resource price components, extreme points can be sought and then find the max point which is to say to find the maximum revenue. Meanwhile the dynamic pricing is determined.

As the conception that cloud is infinite, we don't need to consider remained resources or resources [11] that will be set free to adjust the price as other traditional industries.

When we forecast the trend of packages [12], the time scale will be much longer, such as time scale of months which is more reasonable. In this part, a way to forecast the trend of using situation of packages is proposed and it is especially applied for the packages.

### 3 Pricing Determination in Various Cloud Resources

#### 3.1 Scenario

When some cloud resources are priced, first of all, what kind of services is used should be decided, for example, a package or a combination, and then the using time, which is to say, how long the resources need to be used, which decides the time scale in our dynamic pricing model. In *Dynamic Pricing* how various resources that are chosen by customers themselves are priced is going to be talked about so as to get a biggest profit. In the next *Forecast Model* how to forecast the trend of using situation of different packages will be talked about, in which the contents of packages have been decided by the operating agencies.

In *Dynamic Pricing*, the time scale is a comparatively short time. It can be defined as hours, while in *Forecast Model* the time scale will be much longer, and it can be defined as several days even several months. Because the package is more static than the combination, once the price and contents of the package is defined, it will not change until next season or next time when the company decides to make a change, while the combination is always changing as customers have different requests.

#### 3.2 Basic Model and Dynamic Pricing

There is a certain amount of packages as  $X_{i,i=1,2,3,\dots}$ . Each  $X_i$  includes different resource types of uncertain amount. Different resource types in a package can be defined as a vector  $X_i = (x_1, x_2, x_3, \dots, x_k)$  where  $x_k$  represents a kind of resource such as CPU or MEM and so on.  $X_i$  can also represent resource combinations which are made by customers themselves in *Dynamic Pricing*. User requests (what have been accepted, the below is the same) are always made over times. T represents the spot when a user request is made.

When pricing, not only the present price, but also the change of request trend is necessary. If there are more requests in the future, it can indeed influence pricing. The customers' behaviors can be thought as a serious of changing requests. Thus a metric is defined to represent the trend of changing combination amounts from then to future. As the follows:

**Table 1.** Trend of Changing Combination Amounts

| ... | $t_i$     | $t_{i+1}$     | $t_{i+2}$     | $t_{i+3}$     | ... | $T-2t$        | $T-t$        | $T$   | $T+t$     | $T+2t$     | ... | $t_j$     | $t_{j+1}$     | ...   |
|-----|-----------|---------------|---------------|---------------|-----|---------------|--------------|-------|-----------|------------|-----|-----------|---------------|-------|
| ... | $X_1$     | $X_1$         | $X_1$         | $X_1$         | ... | $X_1$         | $X_1$        | $X_1$ | $X_1$     | $X_1$      | ... | $X_1$     | $X_1$         | $X_1$ |
| ... | $X_2$     | $X_2$         | $X_2$         | $X_2$         | ... | $X_2$         | $X_2$        | $X_2$ | $X_2$     | $X_2$      | ... | $X_2$     | $X_2$         | $X_2$ |
| ... | ...       | ...           | ...           | ...           | ... | ...           | ...          | ...   | ...       | ...        | ... | ...       | ...           | ...   |
| ... | $X_{n_i}$ | $X_{n_{i+1}}$ | $X_{n_{i+2}}$ | $X_{n_{i+3}}$ | ... | $X_{n_{-2t}}$ | $X_{n_{-t}}$ | $X_n$ | $X_{n+t}$ | $X_{n+2t}$ | ... | $X_{n_j}$ | $X_{n_{j+1}}$ | ...   |

In Table 1, T represents the present time,  $t_i$  represents a past time while  $t_j$  represents a future time. The period of t is not defined which can be chosen by the real situation. For example, t can be 1 hour or half an hour, and so on. And when pricing, both

historical amount of accepted requests and relevant price should be considered. With the historical data mentioned before, the accepted requests and the relevant price in the future can be forecasted by BP Neural Network algorithm.

Through adjusting the price of different kinds of resources in the future, a relevant request amount in the future can be forecasted by BP Neural Network. Then the revenue could be maximized by the method which will be mentioned later.

If only one resource is put into consideration in dynamic pricing model, it can be represented as a single symbol, but several resource types for a dynamic pricing model are different. It has to be concerned with a matched problem. And different resource types can form a large quantity of combinations

There are  $i$  historical spots considered to price a combination. As cloud resources are by some means combined, the used historical data should be similar with the combination which is priced. Thus the historical data could be classified into different classifications.

In order to be quickly calculated and self-match, which is more appropriate for the dynamic pricing model and more convenient, the algorithm K-means which is one of data mining methods is proposed as following pseudo code:

---

*Algorithm1. K-means*

---

Required: past data and request now dealt as

$$\mathbf{X} = \{X_1, X_2, \dots, X_n, X_{request}\}$$

(1) Define  $c$  as classification number.

(2) Define the permissible error as  $\varepsilon$  and define  $t=1$ ;

(3) Initializing cluster centers  $w_i(t), i=1, 2, \dots, c$ . where  $c$  centers are chosen from  $X$  random.

(4) Sum of squared error criterion function is as the target function:

$$J_e = d_{ij} \sum_{i=1}^c \sum_{j=1}^n \|x_j - w_i\|^2 \quad d_{ij} = \begin{cases} 1, & x_j \in w_i \\ 0, & x_j \notin w_i \end{cases}$$

(5) Modified clustering center:

$$w_i(t+1) = \frac{\sum_{j=1}^n d_{ij} x_j}{\sum_{j=1}^n d_{ij}}, i = 1, 2, \dots, c.$$

(6) Calculate the error:

$$E = \sum_{i=1}^c \|w_i(t+1) - w_i(t)\|$$

(7) **If**  $E < \varepsilon$  **then**

**For**  $i=1$  to  $c$

Use the  $J_e$  to judge to which group the  $X_{request}$  belongs

**If**  $j_e$  is minimum **then**

Answer =  $i$ ;

**Endif**

**Endfor**

**else**  $t=t+1$  goto (4)

**Endif**

---

“Answer” is the classification the present request belongs to. Thus the historical data in classification “Answer” is considered to be used in the Back Propagation Neural Network. The Back Propagation Neural Network can be defined as function network(x) which will be used in maximizing revenue.

Before using network(x), the data should be dealt with, so as to match with network(x).  $\mathbf{P}_i(p_1 \ p_2 \ \dots \ p_k)$  represents the price of each  $\mathbf{x}_i$  of chosen classification and then each price of  $\mathbf{x}_i$  should be calculated by unit as  $\mathbf{P}_i$ . After that there is a new  $\mathbf{P}_i(p_1 \ p_2 \ \dots \ p_k)$  in which  $p_k$  is all by unit,  $x_{ik}$  is amount of each resource in  $\mathbf{x}_i$ .

$$\mathbf{P}_i(p_1 \ p_2 \ \dots \ p_k) = \mathbf{P}_i \left( \frac{p_1}{x_{i1}} \ \frac{p_2}{x_{i2}} \ \dots \ \frac{p_k}{x_{ik}} \right) \tag{1}$$

And then a Price demand prediction analysis matrix can be defined as following, in which each row is the new  $\mathbf{P}_i$  and the last col is the sum amount of all the accepted requests as  $r_i$

$$\begin{pmatrix} p_{11} & p_{12} & \dots & p_{1k} & r_1 \\ p_{21} & p_{22} & \dots & p_{2k} & r_2 \\ p_{31} & p_{32} & \dots & p_{3k} & r_3 \\ \dots & \dots & \dots & \dots & \dots \\ p_{n1} & p_{n2} & \dots & p_{nk} & r_n \end{pmatrix} \tag{2}$$

As to pricing, a dealt algorithm can be defined as mentioned in the previous section. What is predicted is the trend of users’ behaviors, thus some previous time nodes’ data is used to forecast the future spot’s users’ behavior, which is to say, if the users will accept the price. The process before is to make the price unit, but the amount of visitors is also different, thus making them in a same standard is necessary. For example, at 1<sup>st</sup> hour there are 1000 visitors and 800 people accept the price which is to say there are 800 accepted requests and price p1, at 2<sup>nd</sup> hour there are 100 visitors and 90 people accept the price, which is to say there are 90 accepted requests and price p2 that is less than p1. Even though p2 is less, price p1 attract more accepted requests, because the visit quantity of 1<sup>st</sup> hour is more than 2<sup>nd</sup> hour. When a forecast of customers’ behavior is made, a unified standard is necessary, because the customers will not care how many people visit the website at a spot, what they care is only the price. And considering the resource amount is different in each request, the sum of each resource amount should be used not the request amount. In order to solve those problems, which can influence the forecast of customers’ behaviors, the following algorithm is proposed:

At a certain time T, there are several requests for different combinations. S is defined as a vector in which each component represents the sum of each resource amount in a data center as  $\mathbf{S}=(s_1, s_2, s_3, s_4, \dots, s_n)$ . And the vector  $\mathbf{x}_i$  is the request of that spot. Each component of  $\mathbf{x}_i$  represents each resource amount. Then all resources of requests should be summed by adding the  $\mathbf{x}_i$  up as

$$\mathbf{Sx} = \sum_n \mathbf{x}_i, \mathbf{X}_i = (x_1, x_2, x_3, \dots, x_k) \tag{3}$$

If  $\mathbf{s}_x$  is not matched to S, a special augmented matrix of  $\mathbf{Sx}$  can be defined as  $\tilde{\mathbf{Sx}}$  in which a zero is added to where there is no request of this kind of resource to make sure

that number of  $S_x$ 's cols equal  $S$ 's. Thus  $S_x/S$  will be a part of the equation to price in a same standard. As it is known two vectors can't make a division, thus some little adjustments can be made and do not change the equation's meaning. A diagonal matrix for  $S$  like the style:

$$\text{diag}(s_1 \ s_2 \ \dots \ s_3) = \begin{pmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_n \end{pmatrix} \tag{4}$$

In the equation an inverse matrix  $s^{-1}$  is needed for  $\text{diag}(s_1 \ s_2 \ \dots \ s_3)$  to make calculation. Thus  $S_x/S$  can be thought as another forms  $S_x \times S^{-1}$ . And percentage that amount of resources of accepted requests in sum of resources at a spot multiplies the percentage that accepted requests  $R(t)$  in amount of visitors as  $V(t)$ . So a function to calculate at each spot the relationship between price and the resource amounts can be made.

$$\psi(t) = \frac{R(t)}{V(t)} \times \frac{S_x(t)}{S} = \frac{V(t)}{R(t)} \times (S_x(t) \times S^{-1}) \tag{5}$$

At last, a forecast should be made by Back Propagation Neural Network algorithm with training data of historical data and simulation as the form (10). General number of hidden layer neuron to determine the empirical formula is:

$$i = \sqrt{m+n} + a \tag{6}$$

Where  $I$  is the number of hidden layer neurons,  $n$  is the number of input layer neuron. The learning regular of neutral network is to confirm a  $W$  to make the error minimum.

$$F(W) = (D-T)^T (D-T) \tag{7}$$

And  $D = (d_1 \ d_2 \ \dots \ d_n)$  is the idea output, if  $D$  is the output of the last layout of neutral network then  $n$  will be the number of output. The  $W$  can be made by several tries as

$$W_k(t+1) = W_k(t) + \delta W_k(t) \tag{8}$$

Where  $k$  means the order of layouts of neutral network and  $\delta$  means the adjust value. The neutral network will correct the error of the ideal output and the normal output from the back to the front until the error is the minimum. When the neutral network makes the error minimum, it will be along the decrease which is most fast as

$$\delta W_k(t) = \left( \frac{\partial F(W)}{\partial W_{ij}}(t) \right)_{n_k-1 \times n_k} \tag{9}$$

After the structure of the neural network is decided, the input vector and the output vector are going to be designed. There is input vector as {price of various resources}, where accepted request is from  $\psi(t)$ . And output vector as {a standard calculated by (5)}

$$\mathbf{P} = \begin{pmatrix} p_1 & p_1 & \dots & p_k \\ p_2 & p_2 & \dots & p_k \\ \dots & \dots & \dots & \dots \\ p_{n1} & p_{n2} & \dots & p_{nk} \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} \psi(f_{n1}) \\ \psi(f_{n2}) \\ \dots \\ \psi(f_{ni}) \end{pmatrix} \tag{10}$$

And in neutral network the Excitation function is defined as



$$f(x) = \frac{1}{1 + e^{-x}} \quad (11)$$

Because  $f(x) < 1$  some adjustments of input P and output T are made to make them less than 1. They can be divided by a certain number which depends on real situation. Then the network can be trained as network(x) whose input is price and output is the percentage calculated in (5).

---

*Algorithm 2. BP Neural Network algorithm In Matlab*

---

```
%Training net (just a template)
clear all
function net=network(P,T)
S1=11 according to (4);
S2=1;
net = newff(minmax(P),[S1 S2],{'tansig','logsig'},'traingdx','learnngdm');
net.performFcn='sse'
net.trainParam.show=1;
net.trainParam.mc=0.95;
net.trainParam.epochs=50;
net.trainParam.goal=0.001;
[net,tr]=train(net,P,T);
```

---

After training, the simulation can be made. Also the input should be made less than 1 by dividing them with a number and let the output multiply the same number. The output is the symbol of accepted percentage like (5) forecasted by neural network. If a higher price is not expected because there will be a risk that the higher price may make a fewer revenue, a network(0) is proposed or even a lower price to attract more accepted requests. As to how to set the price which is put into network(x), a price range maybe between  $ra = -10\% \sim 10\%$  which is just a suggestion can be taken into the neural network as network  $(x \times (1 + ra))$ . Following algorithm will finally decide the price:

---

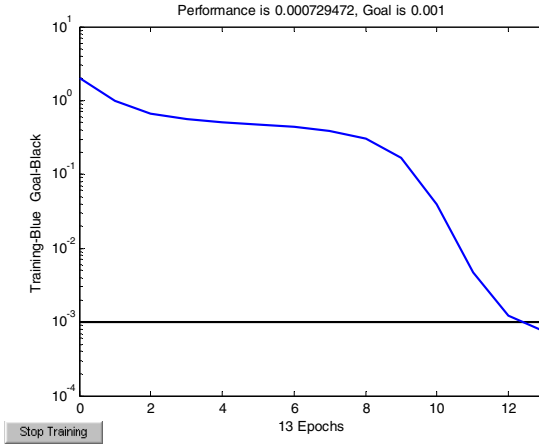
*Algorithm 3. Making revenue maximum with discrete method*

---

```
n ← number of resources
Mr ← 0
Pr ← 0
For i=1 to n
For x=-10% to 10%
     $P_i = P_i \times (1 + x)$ 
     $r_i = \text{network}(x)$ 
If  $P_i \times r_i > Mr$  then
    Mr =  $P_i \times r_i$ 
    Pr =  $P_i$ 
Endif
Endfor
Endfor
```

---

Then the result  $Mr$  represents the maximum revenue in some degree, and the price  $Pr$  is the price. When a forecast is made using Back Propagation Neural Network algorithm with proper training, as it can be seen in Fig.1, in only several times the goal can be reached. The one of the simulation in neural network is as following figure:



**Fig. 1.** Performance of BP Neural Network

But there is still a problem about the way in which revenue is maximized. Price adjustment is discrete. If the maximum revenue being more accurate is expected, a continuous method is proposed. Firstly, the Back Propagation Neural Network algorithm to forecast the relationship between price and the accepted requests as the way mentioned before is also necessary. The equation of max revenue can be defined as follows:

$$y = p_1 \times network(p_1) + p_2 \times network(p_2) + \dots + p_i \times network(p_i) \tag{12}$$

Where  $network(p_i)$  means the forecast result of neural network and  $p_i$  is the price of this kind of the resource, so  $y$  is the revenue. Secondly, a linear regression to fit a function of price and the requests corresponding to the price is proposed to find the relationship between  $p_i$  and  $network(p_i)$ . And a proper nonlinear regression model is better than the linear regression. Then (12) can be changed as the follows:

$$y = f(p_1) + f(p_2) + \dots + f(p_i) \tag{13}$$

Thirdly, partial derivative of  $y$  respect to each  $p_i$  is calculated separately.

$$\frac{\partial y}{\partial p_j} = \frac{\partial f(p_1)}{\partial p_j} + \frac{\partial f(p_2)}{\partial p_j} + \dots + \frac{\partial f(p_i)}{\partial p_j} \quad j=1, 2, 3, \dots, I \tag{14}$$

When  $\frac{\partial y}{\partial p_j} = 0$  the corresponding price of each kind resource is gotten, then the maximum revenue is also achieved.

### 3.3 Forecast Model

As it is known, cloud resources can not only be as a combination, but also be sold as a package. A package is more static than a combination since the combos are always changing. Thus when a forecast model is made for packages, a longer time scale than the combo is necessary. The time scale in this part can be a week, a month, even half a year.

And when there is a package which is not priced or not be introduced, whether it is popular in the daytime or night is still unknown. Thus a Bayes to predict its using trend is needed so that a good price can be made. Even though it may have already been introduced, the use trend of it is not known exactly, because it can't reflect people's real desire that which kind of package they want, since there is not a really desired package for the customers, they only can buy a kind of package which is most closed to their desired package. Using Bayes model to forecast can help us know what the customers want much more specific, which is to say, at which time what users want exactly can be known.

And in this forecast model there is only a forecast of the trend of packages' using situation not pricing. This part is not like *Dynamic Pricing*. Langer time scale will take a pricing problem in economy. For example in *Dynamic Pricing* time scale is 1 hour, in this one hour the prime cost of MEM will not change so much but in a longer time scale as 1 month it's hard to say. Thus a forecast is just made for trend not pricing dynamically. In this part, a Bayesian Decision Theory to forecast is proposed when the package will be used more and its possibility.

For example, there are past 1 month data which are the data of combination not the data of packages because data of packages is of no value for customers' really desired packages and when there is a wish to make a new kind of package, the data of combo will be used. The data then are divided into three time periods as Morning (6:00-14:00), Afternoon (14:00-22:00) and Night (22:00-next day 6:00).

After that a Distribution model should be determined such as Normal distribution or T distribution or  $\Gamma$  distribution. There a Normal distribution is proposed as it's easy to understand.

Now there are several packages as  $\mathbf{x}_i = (x_1, x_2, x_3, \dots, x_k)$  where  $x$  represent each different resource. 1 month data is divided by three time period as follow:

**Table 2.** Data of Combinations in a Month

|           | Day1 | Day2 | Day3 | ..... | Day30 | Day31 |
|-----------|------|------|------|-------|-------|-------|
| Morning   | M1   | M2   | M3   | ..... | M30   | M31   |
| Afternoon | A1   | A2   | A3   | ..... | A30   | A31   |
| Night     | N1   | N2   | N3   | ..... | N30   | N31   |

where  $A_1$  means all the accepted combination in Day 1 Afternoon. N and M are the same. And  $A_1 = (\mathbf{X}_1 \mathbf{X}_2 \dots \mathbf{X}_n)$  where each  $\mathbf{X}_i = (x_1, x_2, x_3, \dots, x_k)$  in A means a combination, n is the amount of combinations. First the three time period's average and the standard deviation are calculated.

Morning:

$$ans_i = \frac{\sum_{(x_i \in \mathbf{X}_i)} }{n} \quad \mathbf{ans} = (ans_1 \quad ans_2 \quad \dots \quad ans_i) \tag{15}$$

$$stdM = \frac{\sum_{i=1}^{31} ans_i}{31} \tag{16}$$

$$sM = \sqrt{\frac{[(X_1 - stdM)^2 + (X_2 - stdM)^2 + \dots + (X_n - stdM)^2]}{n}} \tag{17}$$

The Afternoon and night are the same. Then the three groups of average and standard deviation are taken into the equation of Multidimensional Normal distribution

$$c \cdot \exp\left(-\frac{1}{2R} \sum_{i,j=1}^m R_{ij} \frac{x_i}{\sigma_i} \frac{x_j}{\sigma_j}\right), c = (2\pi)^{-\frac{m}{2}} (\sigma_1 \dots \sigma_m)^{-1} R^{-\frac{1}{2}} \tag{18}$$

And R is the determinant of Correlation matrix  $(p_{ij})$ ,  $R_{ij}$  is its Algebra formula. Then the three Normal distributions will be simultaneous to figure out two Demarcation point or interface. The Bayes formula is used:

$$P(B|A) = \frac{P(B) \times P(A|B)}{P(A)} \tag{19}$$

In details there is a kind of package X, then

$$P(w_i|X) = \frac{P(w_i) \times P(X|w_i)}{P(X)}, w_i \in \{Morning \ afternoon \ night\} \tag{20}$$

Prior probabilities can be the same or based on the real situation. Thus that when the package is more popular can be known, that is the forecasted trend, and its relative probability.

When there are more people want to use this kind of package, maybe the package can be a higher price. As how to make the revenue maximum, you can have the trend which is forecasted by the Bayesian model, combined with a specific economic model.

Through Multidimensional normal distribution there is a basic figure.

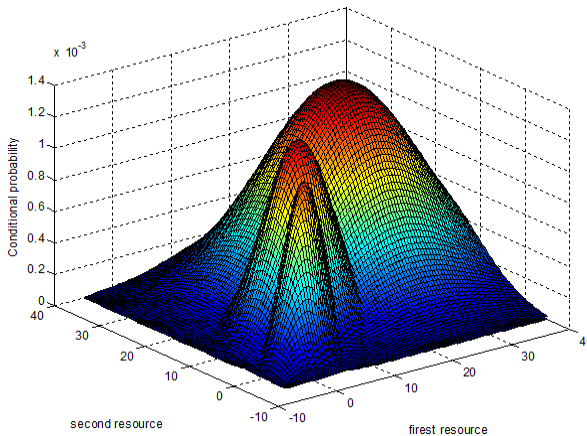
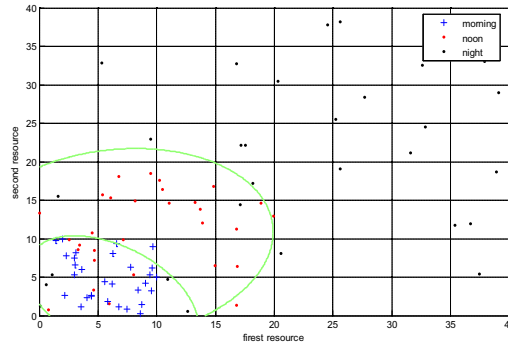


Fig. 2. The 3D view of Bayesian Decision

Then there is a vertical view to look at a Demarcation interface.



**Fig. 3.** Vertical view the 3D view of Bayesian Decision

The package which is forecasted can be made into the vertical view which is the Demarcation interface to look up its possibility in each time period. As can be seen from the vertical view of Bayes Decisions, in the morning people more like the package with first resource 5 and second resource 5 around. And in the afternoon people more like the package with first resource 15 and second resource 15 around while in the evening people's requests are more dispersed between 25 to 40 of both resources although there are several requests of night is between 5 to 20 (the black dot), it is less possibility. So the price according to the forecast trend can be made.

## 4 Numerical Studies

In this section, numerical studies are conducted to evaluate the results between the dynamic pricing model and the static pricing model. The request is set by a Normal distribution. Dynamic pricing model is realized in Matlab and make a comparison.

### 4.1 Comparison between Static Model and Dynamic Model

In this section, there is a comparison about the revenue between our dynamic pricing model and the static pricing model with two various resources as A and B. In Fig.1 it is the comparison of prices between the dynamic pricing model and the static pricing model. As it can be seen, the static pricing is always 10 which is the black line while the dynamic price is between 11 and 9. With the dynamic price changing there are different requests accepted as in Fig.2. In Fig.2 it can be seen that the requests of static pricing model which is the blue line are corresponding to the Normal distribution as mentioned before while the requests of our model is more or less. But no matter the requests of our model is more than the static pricing model or less, the revenue is more in our model as in Fig.3 and Fig.4. In Fig.3 the time scale is hours, the revenue is summed of each hour in continues 24 hours. In Fig.4, the revenue of each day is summed in continues 31 days.

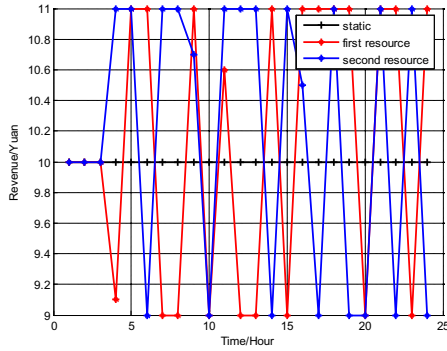


Fig. 4. Price difference between the dynamic model using k-means method and the static model

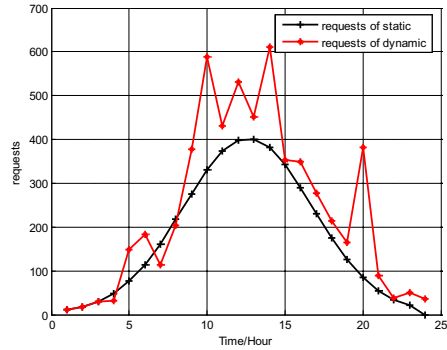


Fig. 5. Requests amount difference between the dynamic model using k-means method and the static model

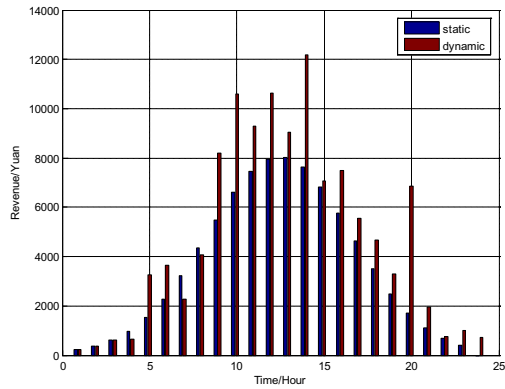
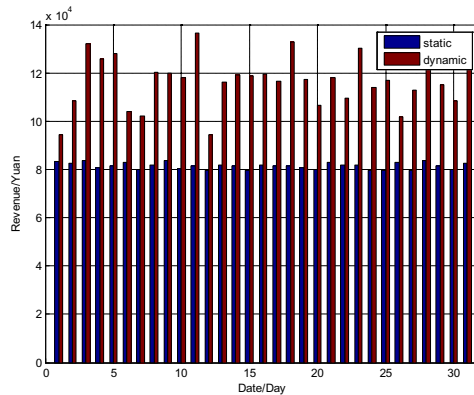


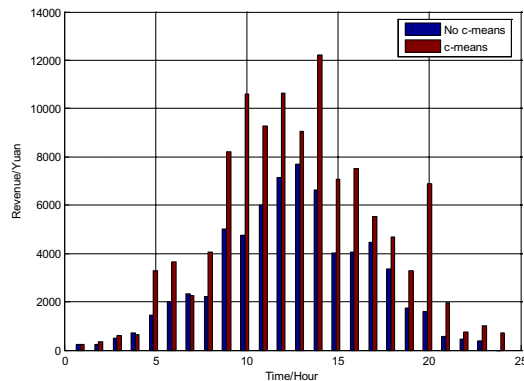
Fig. 6. Hourly revenue difference between the dynamic model using k-means method and the static model



**Fig. 7.** Daily revenue difference between the dynamic model using k-means method and the static model

## 4.2 Performance of K-Means in the Algorithm

In this section, a comparison is made that whether there is a classified data based on different groups of users. In Fig.5 it can be seen if a c-means to classify the users' behaviors into different groups is applied, more revenue can be gotten.



**Fig. 8.** Hourly revenue difference between the model using k-means method and the model without using k-means method

**Acknowledgement.** This paper is supported by Doctoral Fund of Ministry of Education of China (20123108120027), by Science and Technology Commission of Shanghai Municipality (No. 10510500600 and No. 12511502900), by State Key Laboratory of Software Engineering(SKLSE) SKLSE2012-09-36, by State Key Laboratory of Novel Software Technology KFKT2012B30 and by Shanghai Leading Academic Discipline Project (No. J50103).

## References

1. Peng, B., Cui, B., Li, X.: Implementation Issues of a Cloud Computing Platform. *Bulletin of the Technical Committee on Data Engineering* 32(1), 59–67 (2009)
2. Xu, H., Li, B.: Maximizing revenue with dynamic cloud pricing: The infinite horizon case. In: *Proc. of IEEE ICC, Next-Generation Networking Symposium*, pp. 2929–2933. IEEE Press, Budapest (2012)
3. Caron, E., Desprez, F., Muresan, A.: Pattern Matching Based Forecast of Non-periodic Repetitive Behavior for Cloud Clients. *J. Grid Computing*, 49–64 (2011)
4. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud Computing and Grid Computing 360-degree compared. In: *Grid Computing Environments Workshop, Austin*, pp. 1–10 (2008)
5. Constantinou, E., Hill, N.: Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon’s EC2. In: *Cloud Computing and Its Applications, Chicago, IL* (2008)
6. Kondo, D., Javadi, B., Malecot, P., Cappello, F., Anderson, D.: Cost-Benefit Analysis of Cloud Computing versus Desktop Grids. In: *IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–12. IEEE Press, Rome (2009)
7. Anandasivam, A., Premm, M.: Bid Price Control and Dynamic Pricing in Clouds. In: *17th European Conference on Information Systems, Verona*, pp. 328–341 (2009)
8. Wang, D., Zeng, X., Keane, J.: A clustering algorithm for radial basis function neural network initialization. *Neurocomputing* 77(1), 144–155 (2012)
9. Rathnayake, V., Premaratne, H., Sonnadara, D.: Performance of neural networks in forecasting short range occurrence of rainfall. *Journal of the National Science Foundation of Sri Lanka* 39(3), 251–260 (2011)
10. Feng, Y., David, H.: A short-range quantitative precipitation forecast algorithm using back-propagation neural network approach. *Advances in Atmospheric Sciences* 23(3), 405–414 (2006)
11. Luis, R., Eddy, C., Adrian, M., Frédéric, D.: Using clouds to scale grid resources: An economic model. *Future Generation Computer Systems* 28, 633–646 (2012)
12. Yang, X., Xie, W., Huang, J.: A c-means clustering approach based on cloud model. In: *IEEE International Conference on Fuzzy Systems*, pp. 965–968. IEEE Press, Hong Kong (2008)



# Detecting Communities and Corresponding Central Nodes in Large Social Networks

Shengyi Jiang<sup>1</sup> and Meiling Wu<sup>2</sup>

<sup>1</sup> School of Informatics, Guangdong University of Foreign Studies,  
510006 Guangzhou, China

<sup>2</sup> School of Management, Guangdong University of Foreign Studies,  
510006 Guangzhou, China

{jiangshengyi, 402745606}@163.com

**Abstract.** Community structure and central nodes are crucial for analyzing real network systems. However, previous methods investigate these two issues separately. This paper proposes an efficient method to discover both network communities and corresponding central nodes together. This method utilizes an incremental agglomerative clustering to group nodes of large networks based on node structure equivalence. Moreover, central nodes are regarded as special objects and detected by employing outlier detection from the clustering results. This algorithm has been applied to several real networks, and achieves both efficiency and high quality.

**Keywords:** Community detection, central nodes recognition, node centrality, social networks.

## 1 Introduction

Social systems can often be described in terms of complex network, sets of nodes joined together in pairs by links [1]. Community structure detection and node centrality are two important tools for analyzing real-world social networks, since communities represent functional modules of networks and node centrality reveals important individuals that influence in group processes [2].

In recent years, community detection techniques have drawn great attention. Some efficient community detection algorithms for networks have been proposed in recent researches [3], including modularity-based methods [4], [5], spectral methods [6], dynamic methods [7], and methods based on information theory [8], overlapping community detection [9], hierarchical community detection [10], and dynamic community detection [11]. Measures of node centrality have been presented over the years to quantify the importance of individual in network, such as betweenness centrality [12], closeness centrality [13], Google's PageRank [14], and node centrality in weighted networks [15].

As the growing popularity of social network service has provided access to large amount of network data, the tasks of discovering communities and central nodes in such network can be challenging, both in terms of efficiency and quality. In previous

network studies, researchers focused on either community detection or central nodes recognition methods. Particularly, existing node centrality measurement requires a lot of time to compute and sort centrality for all nodes, most of which are marginal and unconsidered in real applications. For large social networks which count in millions of nodes, these methods are computational expensive.

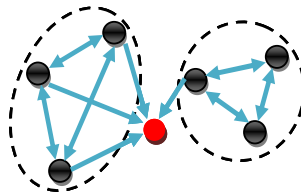
In this paper, we propose a clustering-based network analysis algorithm for simultaneously discovering communities and corresponding central nodes. We obtain clusters of nodes by employing a constrained single pass clustering algorithm [16]. Then, we employ clustering-based outlier detection approach to compute outlier factor of clusters for measuring cluster centrality instead of single node centrality, which efficiently discover central and considered nodes in central clusters [17]. Finally, we combine clusters into communities based on link density between clusters. Our method can also adapt to directed, weighted or unconnected networks.

The rests of this paper are organized as follows. In section 2, definitions of our method are introduced. Section 3 presents the method. Experimental results are presented in section 4. Finally, section 5 concludes this paper and gives further work.

## 2 Preliminaries

There is no accurate definition for community, but several accepted ones are used by different researchers [3]. In this paper, we adopt the definition that a community is a group of objects similar to each other inside than outside them. In order to group the nodes into communities, we will introduce distance between the nodes in the network.

The distance used in this paper is based on node structural equivalence [18]. This distance must be small if the two nodes are structure equivalent, and on the contrary it must be large enough. If two nodes have the same or similar neighbor nodes, they have high structure equivalence even though they are not adjacent, as shown in Fig. 1.



**Fig. 1.** In this network, two sets of nodes enclosed in dash circle have high structural equivalence respectively. And the red node is unique to others as it is in central position.

We embed network nodes in a multi-dimensional space, and denote a node with a multi-dimensional space vector according to its neighbors. Since node degree in real networks follows power-law distribution, most nodes have a few links, so most dimensions of node space vectors are relatively low and the network can be compressed to smaller storage. The distance is computed from the information given by the connection between a node and its neighbors. To compare two nodes, we have two prerequisites:

*Prerequisite 1.* A node in a network is presented with a multi-dimensional space vector  $V = (v_1, v_2, \dots, v_p)$ , where  $p$  is the number of neighbors for the node. The value of each dimension in  $V$  is the weight of edge from node to its each neighbor.

*Prerequisite 2.* If the network is directed, edge from node  $v$  to node  $u$  is not equal to edge from node  $u$  to node  $v$ .

We can now give the definition of distance between nodes below:

**Definition 1.** Let  $v$  and  $u$  be two nodes in a network,  $V = (v_1, v_2, \dots, v_p)$  and  $U = (u_1, u_2, \dots, u_q)$  are vectors for  $v$  and  $u$  respectively, and we define the distance between  $v$  and  $u$  by:

$$d_{vu} = \sum_{i=1}^{|v|} \sum_{j=1}^{|u|} \sqrt{f(v_i, u_j)} \quad (1)$$

Where  $f(v_i, u_j)$  is computed as the following formula:

$$f(v_i, u_j) = \begin{cases} (v_i - u_j)^2, & \delta(i, j) = 0 \\ v_i^2, & \delta(i, j) = 1 \\ u_j^2, & \delta(i, j) = -1 \end{cases} \quad (2)$$

Where  $\delta(i, j)$  is to estimate whether the  $i$ -th neighbor of  $v$  is the same as the  $j$ -th neighbor of  $u$ .  $\delta(i, j) = 0$  when  $v_i$  and  $u_j$  represent the same neighbor,  $\delta(i, j) = 1$  when  $u$  not has the  $i$ -th neighbor of  $v$ ,  $\delta(i, j) = -1$  when  $v$  not has the  $j$ -th neighbor of  $u$ .

When nodes aggregate to clusters, we should give the distance between existing clusters and a node. Here we also embed a cluster with a multi-dimensional vector which is a centroid of all node vectors in this cluster.

**Definition 2.** Let  $C$  be one cluster with  $m$  nodes,  $M = (M_1, M_2, \dots, M_m)$  be the  $m$  node vectors,  $k$  be the number of identical neighbors of the  $m$  nodes, and we denote cluster  $C$  with vector  $C = (c_1, c_2, \dots, c_m)$ , and each dimension  $c_i$  by:

$$c_i = \frac{\sum_{j=1}^m f(M_j, i)}{m} \quad (3)$$

Where  $c_i$  is the value of the  $i$ -th dimension of  $C$ , which is the centroid of the same dimension of the  $m$  node vectors.  $f(M_j, i) = 0$  if node vector  $M_j$  not has the  $i$ -th dimension of cluster vector  $C$ . Otherwise,  $f(M_j, i)$  is equal to the value of the same dimension of  $M_j$ .

Now we generalize distance between nodes to a cluster in a straightforward way.

**Definition 3.** Let  $v$  be a node with vector  $V = (v_1, v_2, \dots, v_p)$ ,  $C$  be a cluster with vector  $C = (c_1, c_2, \dots, c_q)$ . We define the distance  $d_{vc}$  between node  $v$  and cluster  $C$  by:

$$d_{vc} = \sum_{i=1}^p \sum_{j=1}^q \sqrt{f(v_i, c_j)} \quad (4)$$

Where the computation of  $f(v_i, c_j)$  is similar to equation (2) since both parameters are space vectors.

In section 3, in order to find central nodes clusters, we should compute the outlier factor of clusters based on distance between clusters. Here we give the definition of clusters distance and outlier factor.

**Definition 4.** Let  $A$ ,  $B$  be two clusters with vector  $A = (a_1, a_2, \dots, a_p)$  and  $B = (b_1, b_2, \dots, b_q)$ . We define the distance between the two clusters by:

$$d_{AB} = \sum_{i=1}^p \sum_{j=1}^q \sqrt{f(a_i, b_j)} \quad (5)$$

Where the computation of  $f(a_i, b_j)$  is similar to equation (2) since both parameters are space vectors.

**Definition 5.** Let  $CP = \{C_1, C_2, \dots, C_p\}$  be the clustering results. The outlier factor of cluster  $C_i$ ,  $OF(C_i)$  is defined as 2-power means of distances between cluster  $C_i$  and the rest of clusters:

$$OF(C_i) = \sqrt{\left( \frac{\sum_{j \neq i} d(C_i, C_j)^2}{|CP| - 1} \right)} \quad (6)$$

Where  $OF(C_i)$  measures how far the nodes in  $C_i$  departs away from the rest of network. The larger  $OF(C_i)$  is the more outer  $C_i$  departs away from the rest.

For combining different clusters to form communities, we should measure the link density between clusters.

**Definition 6.** Let  $C_i, C_j$  be two clusters, and  $L$  is the number of edges linking  $C_i$  and  $C_j$ . We define the link density between the two clusters by:

$$LD(C_i, C_j) = \frac{L}{(|C_i| * |C_j|)} \tag{7}$$

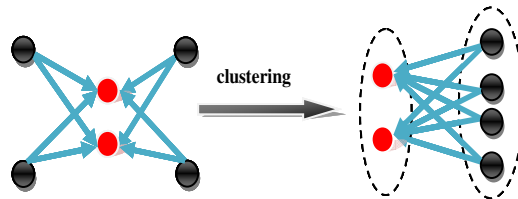
### 3 The Algorithm

In the previous section, we have introduced distance computation between nodes based on node structural equivalence. We now propose our algorithm that finds clusters of nodes with similar structure property, and then discover communities and corresponding central nodes by clustering results. Our method has the following features:

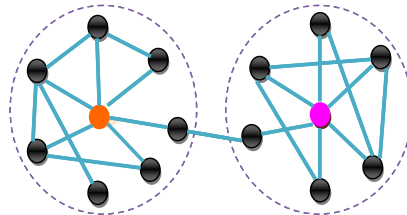
*Feature 1.* Central nodes and non-central nodes in a community are separated when clustering according to their different structural equivalence, as shown in Fig. 2.

*Feature 2.* Since network node degrees follow power-law distribution, central nodes which have higher degree, are much less than non-central nodes. We regard these central nodes as outliers and employ outlier detection approach to find them.

*Feature 3.* A central node is always in the center of community instead of the complete network, which we call community constraint. The example is shown in Fig. 3. Our method can discover community-based central nodes and achieves efficiency.



**Fig. 2.** Separation of central and non-central clusters. In this network, the nodes are separated to central cluster and non-central cluster after clustering.



**Fig. 3.** Example of community constrained central nodes. There are two communities in this network, each of which has its own central node. The community constrained central node can only have effect on members in the same community, but not other communities.

### 3.1 Detecting Communities and Corresponding Central Nodes

Grouping nodes into clusters. We use an efficient incremental clustering algorithm that only scans network data in single pass without loading the whole network in memory at one time, which can process large networks. By clustering, clusters containing central nodes and clusters including non-central nodes in different communities are generated by node structural equivalence.

Identifying central clusters. We distinguish clusters containing central nodes from clustering results. Central nodes, such as celebrities or leaders, are often distant from non-central nodes, so we regard clusters of central nodes as outliers that appear to deviate markedly from other nodes of the network. For purpose of identifying central clusters, we employ outlier detection method based on clustering results.

Forming clusters into communities. The clusters generated by clustering are components of communities, so the next step is to decide how to combine central clusters and non-central clusters into communities. We form clusters into communities according to clusters link density.

The details about our method are described as follows:

- Step 1.* Initialize the set of clusters,  $S$ , as the empty set, and read a vector of node  $v$ .
- Step 2.* Create a cluster with the node vector.
- Step 3.* If no nodes are left in network, go to step 6, otherwise read a node vector, and find the cluster  $C^*$  in  $S$  that is closest to the node with the distance by  $d_{vC^*}$ .
- Step 4.* If  $d_{vC^*} > R$ , go to *Step 2*.
- Step 5.* Merge  $v$  into cluster  $C^*$  and modify the vector of cluster  $C^*$ . Go to *Step 3*.
- Step 6.* Sort clusters  $S = \{C_1, C_2, \dots, C_p\}$  by  $OF(C_1) \geq OF(C_2) \geq \dots \geq OF(C_p)$ .
- Step 7.* Search the smallest  $b$  which satisfies  $\sum_{i=1}^b \frac{|C_i|}{n} \geq \mathcal{E}$ , and label clusters  $C_1, C_2, \dots, C_b$  with central clusters, while  $C_{b+1}, C_{b+2}, \dots, C_p$  with non-central clusters.
- Step 8.* Compute link density  $LD$  between central and non-central clusters;
- Step 9.* If  $LD \geq \lambda$ , merge central and non-central cluster to form a community.

### 3.2 Parameter Selection

In our method, several parameters are required to be set, including the distance threshold  $R$  in clustering, the proportion of central clusters  $\mathcal{E}$ , and the link density threshold  $\lambda$  in forming clusters to communities.

Selecting threshold  $R$ . The threshold  $R$  may influence the quality of clustering and the time-efficiency of the algorithm. As  $R$  increases, both the number of produced clusters and time-costing will increase. In order to gain a reasonable and relative stable threshold  $R$ , we employ sampling techniques to determine the threshold [19].

The details are described as follows:

- Step 1.* Choose randomly  $N_0$  pairs of node vectors in the network.
- Step 2.* Compute distance between each pair of nodes.
- Step 3.* Compute average  $ex$  of distance from *Step 2*.
- Step 4.* Selecting  $R$  as  $e \bullet ex$ , where  $e$  in  $[0.8, 1.8]$ .

When  $N_0$  reaches a higher value,  $ex$  remains stable. The value of threshold  $R$  is closely related to the size of network and its application areas.

Selecting parameter  $\mathcal{E}$ . The value  $\mathcal{E}$  is an approximate ratio of the outlier nodes to the whole network. Since node degree of most real networks follows power-law distribution, the proportion of central nodes in a network is usually less than 20%. We may determine  $\mathcal{E}$  exactly based on the node degree distribution of real networks.

**Selecting Parameter  $\lambda$ .** The parameter  $\lambda$  may influence the formation of communities. As  $\lambda$  decreases, central clusters may connect to several non-central clusters, thus the number of communities will decrease while the size of communities will increase. The value is related to the density of network, and we set  $\lambda$  from 0.6 to 1.0 in a rather dense network.

### 3.3 Complexity Analysis

We analyze time complexity for each step of our algorithm. The time complexity of clustering algorithm is nearly linear with the size of network, by  $O(n)$ . Supposing  $k$  clusters generated during clustering process, the time consume of finding central clusters is  $O(k^2)$ . We supposed  $\alpha * k$  clusters are central clusters while  $(1 - \alpha) * k$  are non-central clusters ( $0 < \alpha < 1$ ). The time complexity in forming clustering into communities can be expected to be  $O(k^2)$ , which is very small compared to  $n$ .

In summary, the complexity of our algorithm can be expected to be  $O(n)$ , which makes the method deserve good scalability. Since it is no need to load the whole network at one time, the space complexity varies from capacity of memory.

## 4 Experimental Results

In this section, we verify the validity of our algorithm on a number of test-case networks and we have compared it with three other community detection algorithms [4], [5], [7]. We utilize modularity [20], which is the most widely used evaluation metric of network community detection, to measure our algorithm performance. The environment settings of our empirical computer are as follows: Pentium(R) D CPU 2.80 GHz, 2.79 GHz, 2.00 GB; Operation System is Microsoft Windows XP SP2.

#### 4.1 Comparison of Community Detection Results

The networks include Zachary's karate club network [21], scientific collaboration network [22], and phone network. Zachary's karate club network is a network of friendship among 34 members of a karate club. Over a period of time the club split into two factions due to leadership issues and each member joined one of the two factions. Scientific collaboration network is the network of coauthorships between scientists posting preprints on the Condensed Matter E-Print Archive between Jan 1, 1995 and March 31, 2005. It consists of 39577 nodes and 175692 edges. Phone network is a network of communication between customers in a China mobile operator in a month, and it consists of 1330749 nodes and 11667521 edges.

The results of community detection by using our method and other algorithm are shown in Table 1. The table displays the modularity and number of communities. Empty cells correspond to unavailability of our machine performance or a computation time over 24 hours. The results demonstrate that our method is approximate to all other methods in quality, and finds central nodes at the same time.

**Table 1.** Comparison of community detection results

|             | Karate  | Collaboration | Phone              |
|-------------|---------|---------------|--------------------|
| Nodes/edges | 34/77   | 39577/175692  | 1330749 / 11667521 |
| CNM         | 5/0.381 | 1907/0.626    | -/-                |
| BGLL        | 4/0.420 | 1842/0.722    | 2231/0.715         |
| Infomap     | 3/0.490 | 1954/0.786    | -/-                |
| Our Method  | 2/0.51  | 1858/0.781    | 3125/0.730         |

#### 4.2 Details of Karate Network Results

For interpreting the details of results of community and corresponding central nodes, we examine a small social network- karate network.

Clustering results are shown in Table 2. Seven clusters are generated, including central clusters and non-central clusters. Two communities  $C_1$  and  $C_2$  are formed based on the clusters in Table 2, as shown in Fig. 5, where cluster 2 is linked to central cluster 1 with  $LD_{21}=1.0$ ; cluster 4 is linked to central cluster 1 with  $LD_{41}=0.889$ ; cluster 2 is linked to central cluster 3 with  $LD_{23}=1.0$ ; cluster 5 is linked to central cluster 6 with  $LD_{56}=0.882$ ; cluster 5 is linked to central cluster 7 with  $LD_{57}=0.882$ .

In order to verify the results of central nodes recognized by our method, we show the centrality results computed by betweenness, closeness, degree and PageRank, as shown in Table 3. The values in Table 2 are the orders of centrality of node 1, 3, 33 and 34. The results demonstrate that the central nodes we find using our method agree with other centrality methods.

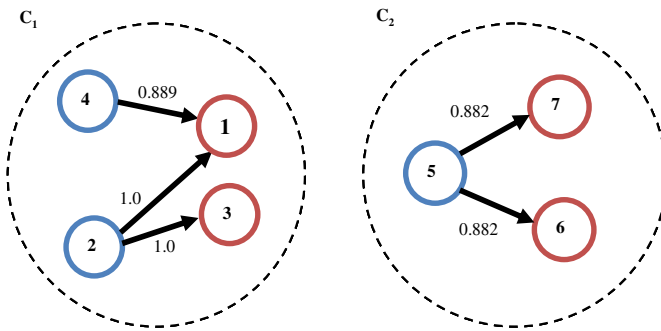


**Table 2.** Clustering results on Zarchary’s karate club network

| Cluster | Members   | Type        |
|---------|---|-------------|
| 1       | 1   | central     |
| 2       | 2,4,8,14  | non-central |
| 3       | 3   | central     |
| 4       | 5,6,7,11,12,13,17,18,22                           | non-central |
| 5       | 9,10,15,16,19,20,21,23,24,25,26,27,28,29,30,31,32 | non-central |
| 6       | 33  | central     |
| 7       | 34  | Central     |

**Table 3.** Comparison of different centrality results of central nodes

| Node | Betweenness | Closeness | Degree | PageRank |
|------|-------------|-----------|--------|----------|
| 1    | 1           | 1         | 2      | 2        |
| 3    | 4           | 2         | 4      | 4        |
| 33   | 3           | 7         | 3      | 3        |
| 34   | 2           | 3         | 1      | 1        |



**Fig. 4.** Two communities are detected by our method. Community  $C_1$  includes non-central cluster 2 and 4, while central cluster 1 and 3; Community  $C_2$  includes non-central cluster 5, while central cluster 6 and 7.

## 5 Conclusion and Discussion

We have proposed a novel method to detect communities and corresponding central nodes in large network. In order to group nodes efficiently, a distance between nodes based on node structure equivalence is introduced. This distance can be utilized with agglomerative clustering algorithm to aggregate nodes without loading the whole network at one time. During clustering process, central nodes and non-central nodes in network are separated, which enable us to find central nodes within communities.

Moreover, extensive experiments have been run to compare existing approaches. They show that our method provides excellent results in community detection and

central nodes recognition. Our approach is also relevant for the computation of overlapping communities. Finally, we pointed out that the method can be extended to detect hierarchical communities and track evolving communities, which are our interesting research directions.

**Acknowledgements.** This work was supported by the National Natural Science Foundation of China (No.61070061), Guangdong Province Colleges and Universities Discipline Construction Project (No.2012KJCX0049), National Key Technologies R&D Program Project (No.2012BAH02F03), Guangzhou Science and Technology Program Project (No.2011J5100004).

## References

1. Strogatz, S.H.: Exploring complex networks. *Nature* 410, 268–276 (2001)
2. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* 99, 7821–7826 (2002)
3. Fortunato, S.: Community detection in graphs. *Physics Reports* 486, 75–174 (2010)
4. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. *Physical Review E* 70, 066111 (2004)
5. Blondel, V.D., Guillaume, J., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, P10008 (2008)
6. Izudheen, S., Mathew, S.: A Method for Community Detection in Protein Networks Using Spectral Optimization. *International Journal of Database Management Systems* 3, 161–167 (2011)
7. Rosvall, M., Bergstrom, C.T.: Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. USA* 105, 1118–1123 (2008)
8. Rosvall, M., Bergstrom, C.T.: Maps of information flow reveal community structure in complex networks. *Proc. Natl. Acad. Sci. USA* 105, 1118–1123 (2008)
9. Psorakis, I., Roberts, S., Ebden, M.: Overlapping community detection using Bayesian non-negative matrix factorization. *Physical Review E* 83, 066114 (2011)
10. Shen, H., Cheng, X., Cai, K., Hu, M.: Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications* 388, 1706–1712 (2009)
11. Greene, D., Doyle, D.: Tracking the evolution of communities in dynamic social networks. In: *Advances in Social Networks Analysis and Mining*, pp. 176–183 (2010)
12. Freeman, L.C.: A set of measures of centrality based on betweenness. *Sociometry* 40, 35–41 (1977)
13. Freeman, L.C.: Centrality in social networks: I. Conceptual clarification. *Social Networks* 1, 215–239 (1979)
14. Austin, D.: How Google Finds Your Needle in the Web’s Haystack. *American Mathematical Society Feature Column* (2006)
15. Opsahl, T., Agnessens, Skvoretz, J.: Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks* 32, 245–251 (2010)
16. Jiang, S.: Efficient Classification Method for Large Dataset. In: *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian*, pp. 1190–1194 (2006)

17. Jiang, S., Song, X., Wang, H., Han, J.-J., Li, Q.-H.: A clustering-based method for unsupervised intrusion detection. *Pattern Recognition Letters* 27, 802–810 (2006)
18. Lorrain, F., White, H.: Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology* 1, 49–80 (1971)
19. Grubbs, F.E.: Procedures for detecting outlying observations in samples. *Technometrics* 11, 1–21 (1969)
20. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. *Physical Review E* 69, 026113 (2004)
21. Zachary, W.W.: An information flow model for conflict and fission in small groups. *Journal of Anthropological Research* 33, 452–473 (1977)
22. Newman, M.E.J.: The structure of scientific collaboration networks. *Proc. Natl. Acad. Sci. USA* 98, 404–409 (2001)

# The Design and Prototype Implementation of a Pipelined Heterogeneous Multi-core GPU

Junyong Deng<sup>1,2</sup>, Libo Chang<sup>2</sup>, Guangxin Huang<sup>2</sup>,  
Lingzhi Xiao<sup>2</sup>, Tao Li<sup>2</sup>, Lin Jiang<sup>2</sup>, Jungang Han<sup>2</sup>, and Huimin Du<sup>2</sup>

<sup>1</sup> School of Micro-electronics, Xidian University, Xi'an 710071, China

<sup>2</sup> School of Electronic Engineering, Xi'an University of Posts & Telecommunications, Xi'an 710121, China

**Abstract.** Because of the widespread use of 3D graphics processing units, this paper presents the design and implementation of a heterogeneous multi-core graphics processing unit, the HMGPU-9. HMGPU-9 supports OpenGL2.0 and DirectDraw with programmable vertex shaders and fragment shaders. It integrates 9 heterogeneous processor cores and many sophisticated application-specific accelerators into a XC6VLX550T FPGA. It employs dual-rail handshake protocol in its rendering pipeline to achieve high performance. It is capable of assigning graphics processing tasks to different processors for efficiency and flexibility. The pixel filling rate can reach 289.92Mpixel/s at its peak performance.

**Keywords:** heterogeneous, multi-core processor, graphics processing unit, pipeline, parallel execution.

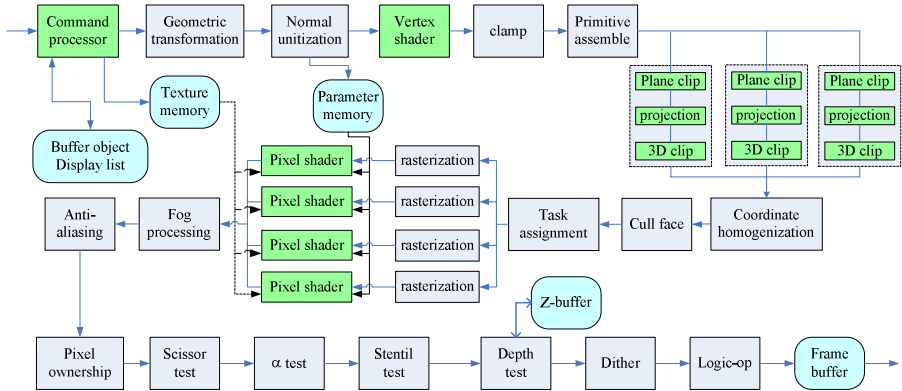
## 1 Introduction

In recent years, with the increasing demand of vertex shading, pixel shading, geometric rendering, physical computing and general purpose computing, the GPU (Graphics Processing Unit) has evolved from a graphics accelerating application-specific circuit to a large scale parallel processing SoC [1] (System on Chip). However, there remain many challenges to be resolved, such as the maximization of parallelism, the effective distribution of rendering tasks, and the balance between performance and flexibility. Taking all the issues into consideration, this paper presents the design and FPGA prototyping of a pipelined heterogeneous multi-core graphics processing unit, HMGPU-9. This graphics processing unit supports OpenGL2.0 style of shaders and DirectDraw 2D graphics, and allows the parallel processing of graphics rendering tasks. HMGPU-9 achieves a balance between performance and flexibility, and its pixel filling rate reaches up to 289.92Mpixel/s.

## 2 Architecture Design

As defined in the OpenGL specification, the traditional graphics processing pipeline includes several stages. The pipeline starts with the model-view transformation stage to

transform vertices and normals from object coordinates into eye coordinates. Vertex shading is then performed. Shaded vertices are then assembled into geometric primitives such as lines and triangles. Clipping and projective transformation are then performed on the primitives. After that, back face culling is performed to remove primitives that need not be scan-converted. The primitives are eventually transformed into window coordinates via viewport transformation[2]. This kind of stream processing is appropriate to be done in a pipeline[3]. The processing requirement of real-time graphics and that of general purpose computing tasks demand for ever-increasing high performance. On the other hand, it has also become increasingly difficult to improve performance merely relying on increased clock frequency. It is thus critical to exploit the parallelism among tasks to improve performance[4]. A stage in the graphics pipeline is typically a coarse grain task which can be parallelized. For example, pixel shading stage can employ many programmable shaders to perform fragment shading in parallel. A combination of programmability and task specific acceleration results in Application Specific Instruction Processors (ASIP). This is the approach adopted by the prototype graphics processing unit of this paper.



**Fig. 1.** The architecture of HMGPU-9

According to the above, the first choice architecture for implementing a graphics rendering engine[5,6] is a heterogeneous multi-core SoC to achieve the objectives of programmability and coarse-grained parallelism in the rendering pipeline. After research and performance simulation of the features of different graphics processing tasks, we decide to adopt the heterogeneous multi-core approach for our implementation of a graphics processing unit. This culminates the HMGPU-9 architecture, which integrates 9 processor cores with different ISAs (Instruction Set Architecture) and many application-specific accelerators in the rendering pipeline. HMGPU-9 performs various forms of parallel processing in the graphics rendering pipeline, and settles one difficulty of the integration of multi-cores[7]. Figure 1 shows the architecture of HMGPU-9. The programmable cores include the command processor, the vertex shader, the plane-clip/projection transformation/3D-clip (CPC) controllers, and the pixel shaders. Due to the excessive computation demanded by

clipping, projection and fragment shading, HMGPU-9 integrates 3 clipping/projection controllers and 4 pixel shaders to speed up the processing of clipping and fragment shading.

For data transfer between any pair of modules, a common scheme is to use a FIFO[4], as shows in figure 2. However, since on-chip SRAM is a limited resource, HMGPU-9 uses the dual rail handshake protocol to connect a pair of adjacent modules, as shown in figure 3. In this scheme, a pair of *Valid* and *ready* signals are used to control the input and output of each pipeline stage. *Valid* indicates the availability of data, and *ready* means the current stage can receive input data.

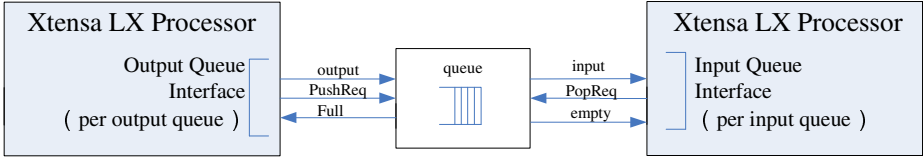


Fig. 2. Xtensa LX Queue Interface

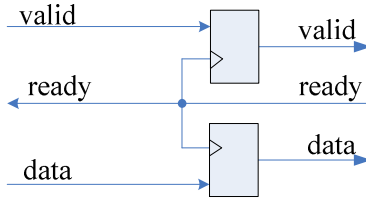


Fig. 3. dual-rail handshake protocol Interface

When the data signals *data* arrives at the current stage, the control signal *valid* should be logic '1'; and when there is no data, *valid* be logic '0'.

When *ready* is logic '1', *data* and *valid* are accepted by current pipeline stage; and when *ready* is '0', the pipeline is paused.

### 3 Circuit Design and Prototype System Realization

HMGPU-9 consists of 9 processor cores of different types and many complicated application-specific circuits. The 9 cores include a front-end command processor, a vertex shader unit supporting fixed point arithmetic and IEEE754 float-point operations, three CPC controllers supporting fixed point and IEEE754 float-point operations, four pixel shaders supporting fixed point as well as scan-conversion operations. To achieve desired performance, the scan-conversions units employ a sophisticated pipeline design. A scan-conversion unit is about three times the complexity as the vertex shader. The application-specific circuits include a 4-channel DMA controller, a primitive assembly unit, a memory management and DMA unit, a coordinate homogenizer, a back-face culling controller, a viewport transform unit, a fragment operation unit, a pixel-cache, a z-buffer, a frame buffer and a display controller.

### 3.1 Heterogeneous Microprocessors

#### 3.1.1 Front-End Command Processor

The front-end command processor is responsible for the analyzing of OpenGL and DirectDraw commands and for data transfer between the host and the GPU. Input commands can be classified into five categories: buffer object commands, display list commands, rendering program loading commands, status commands and other commands (the rest OpenGL commands and DirectDraw commands).

The command processor is basically a RISC processor with a few special purpose instructions. The command processor uses 32-bit long instructions, contains a register file of 32 general-purpose registers. For the specific requirements of command processing, this processor also interfaces to 27 interface registers (IFR), which are used to communicate address, data and control information among the command processor, the DMA controller, the memory management unit (MMU), the host and the on-chip memory. The command processor has a four stage pipeline: the instruction fetch (IF) stage, the instruction decoding stage, the execution stage and the write-back stage. The IF stage can access the IM unit, which stores the assemble program to handle the input commands. The execution stage and the write-back stage also read and write the on-chip memory. Figure 4 shows the architecture of command processor.

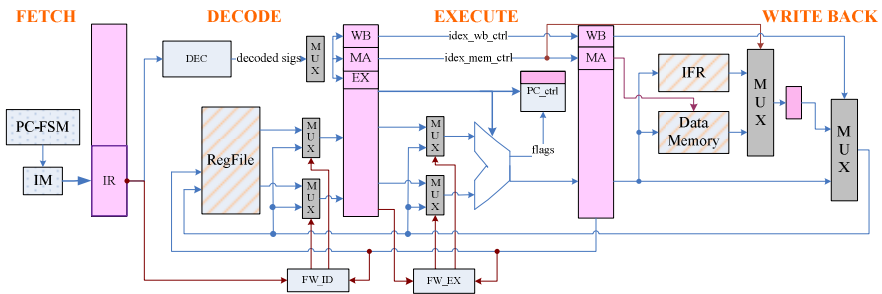


Fig. 4. The architecture of the command processor

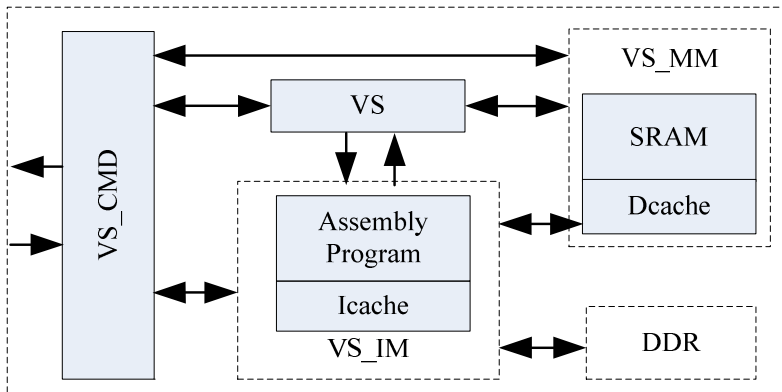


Fig. 5. the architecture of vertex shader

### 3.1.2 Vertex Shader

The vertex shader deals with the vertex, texture, raster, normal, light and geometric transformation related commands. It includes fixed-point and float-point instructions. Figure 5 shows the architecture of vertex shader. The *VS\_CMD* module handles the reading and writing of parameters in commands. The *VS\_MM* module takes charge of the management of parameter storage memory and data cache. The *VS* module accomplishes geometric transformation and light computing. *VS\_IM* loads the rendering assemble program from the off-chip memory.

### 3.1.3 CPC Controller

Plane clipping and 3D clipping are complicated operations in the graphics pipeline. Such operations are time-consuming. A CPC controller accomplishes plane-clip, projection transformation and 3D-clip. Plane-clip indicates the function of clipping the objects in world space with the user-defined clip planes. Projection transformation finishes multiplying of projection matrix and input vertices, matrix loading, computing of the inverse of transpose matrix and processing of stack commands. 3D-clip eliminates the primitives out of the view frustum in order to reduce computing of following components. CPC controller also utilizes the Harvard architecture and consists of four pipeline stages, as shown in figure 6.

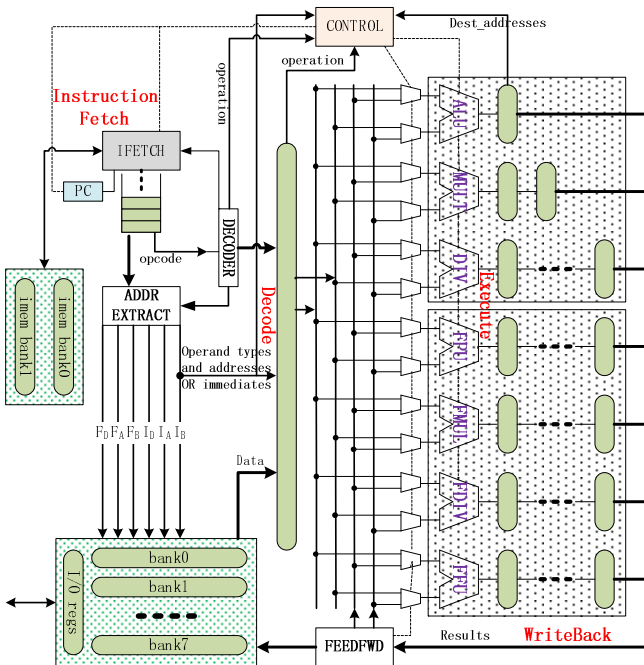


Fig. 6. The architecture of CPC



### 3.1.4 Pixel Shader

The pixel shader handles such operations as rasterization, fragment shading, texture mapping and fog effects etc. The ISA of a pixel shader includes fixed-point instructions and fragment shading specific operations. As shown in figure 7, the pixel shader integrates four scan converters *pls\_sc1*, *pls\_sc2*, *pls\_sc3*, and *pls\_sc4*, a scheduler *PLS\_SCHEDUAL*, and four fragment shaders *pls\_pps1*, *pls\_pps2*, *pls\_pps3*, and *pls\_pps4*. A scan-converter is about three time the complexity of that of a fragment shader.

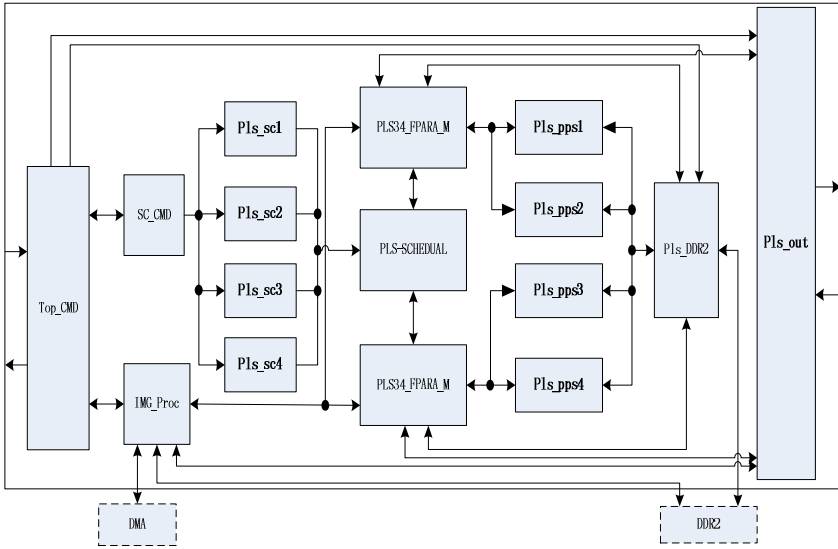


Fig. 7. The architecture of pixel shader

## 3.2 HMGPU-9 Prototype System

In accordance with architecture design described above, according to the circuit scale and performance requirements, the prototype system is designed on the Xilinx's FPGA XC6VLX550T, with the latest development board DNV6\_F2PCIE. As for the command entering, the EMU interface is secondary developed.

### 3.2.1 System Description

The HMGPU-9 graphics processing system is shown in figure 8. The left monitor displays the operating results of HMGPU-9, and the right part is the host control interface.

Figure 9 indicates the resource utilization report of HMGPU-9. According to the formula provided by Xilinx, the scale of HMGPU-9 is nearly 5.27 million logic gates.

The frequency of HMGPU-9 is 72.48MHz, and it can handle 4 pixels per second since there are 4 pixel shaders. The pixel filling rate is 289.92Mpixel/s.



**Fig. 8.** The prototype system of HMGPU-9

Loading device for application Rf\_Device from device xc6vlx550t, package ff1760, speed -1

```

Slice Logic Distribution:
Number of occupied Slices:          69,024 out of  85,920   80%
Number of LUT Flip Flop pairs used: 249,212
  Number with an unused Flip Flop:   90,562 out of 249,212   36%
  Number with an unused LUT:         34,401 out of 249,212   13%
  Number of fully used LUT-FF pairs: 124,249 out of 249,212   49%

IO Utilization:
Number of bonded IOBs:              146 out of   1,200   12%
Number of LOCed IOBs:               146 out of    146   100%

Specific Feature Utilization:
Number of RAMB36E1/FIFO36E1s:       534 out of    632   84%
Number of RAMB18E1/FIFO18E1s:       91 out of   1,264    7%
Number of BUFG/BUFGCTRLs:           14 out of    32   43%

```

**Fig. 9.** The circuit scale of HMGPU-9

### 3.2.2 Test Results

Figure 10 shows the rendering result of HMGPU-9. It indicates that HMGPU-9 supports all the 3D primitives, such as point, line and triangles, and the geometric transformation, shading mode, lighting, texture and DirectDraw related commands.

## 4 Summary

This paper discusses the design and implementation of a multi-core SoC graphics rendering engine. We present a pipelined heterogeneous multi-core GPU *HMGPU-9*. *HMGPU-9* accomplishes the graphics rendering tasks on different processor cores, and integrates the nine cores with many application specific hardware accelerators. *HMGPU-9* supports OpenGL1.3 and DirectDraw. It uses about 5.27 million logic gates and runs at a pixel filling rate of up to 282.92Mpixle/s.



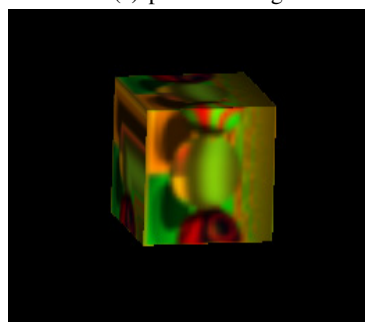
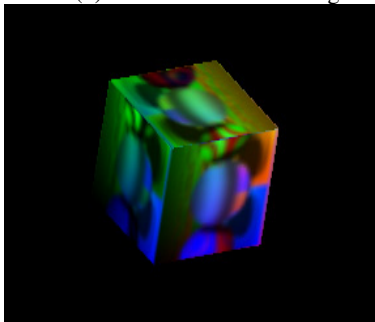
(a) Chinese map rendered by HMGPU-9(left) Chinese map rendered by Visual Studio running on PC(right)



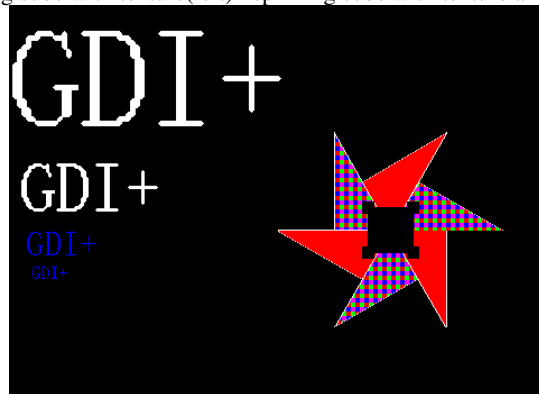
(b) smooth mode of shading



(c) sphere with light



(d) spinning cube with texture(left) spinning cube with texture and light(right)



(e) results of DirectDraw commands

**Fig. 10.** The test resultsof HMGPU-9

**Acknowledgments.** This paper is supported by the National Natural Science Foundation of China under Grant Nos.61136002, 61272120; the Shaanxi Province Science and Technology Research and Development Program under Grant No.2011K06-4; the Shaanxi Education Department Program under Grant No.2010JK817. We really appreciate the trust of the State Natural Science Funds Commission to our early research work.

## References

1. Han, J.-G., Liu, Y.-Y., Zhang, X.: GPU:The Past, Present and Future. *Journal of Xi'an University of Posts and Telecommunications* 16(3), 61–64 (2011)
2. Shreiner, D., Woo, M., Neider, J., Davis, T.: *OpenGL® Programming Guide*, 6th edn. China Machine Press, Beijing (2009)
3. Tumeo, A., Branca, M., Camerini, L., Ceriani, M., Monchiero, M., Palermo, G., Ferrandi, F., Sciuto, D.: Prototyping Pipelined Applications on a Heterogeneous FPGA Multiprocessor Virtual Platform, pp. 317–322. *IEEE* (2009)
4. Shee, S.L., Parameswaran, S.: Design Methodology for Pipelined Heterogeneous Multiprocessor System. In: *DAC 2007*, San Diego, California, USA, June 4-8, pp. 811–816 (2007)
5. Javaid, H., Parameswaran, S.: A Design Flow for Application Specific Heterogeneous Pipelined Multiprocessor Systems. In: *DAC 2009*, San Francisco, California, USA, July 26-31, pp. 250–253 (2009)
6. Javaid, H., Ignjatovic, A., Parameswaran, S.: Rapid Design Space Exploration of Application Specific Heterogeneous Pipelined Multiprocessor Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29(11), 1777–1789 (2010)
7. Chen, F.-Y., Zhang, D.-S., Wang, Z.-Y.: Research of the Heterogeneous Multi-Core Processor Architecture Design. *Computer Engineering & Science* 33(12), 27–36 (2011)

# A Parallel Approach for Real-Time OLAP Based on Node Performance Awareness

Wei He and Lizhen Cui

School of Computer Science and Technology, Shandong University,  
Middle of Shunhua Road, Jinan, P.R.China 250101  
hewei@sdu.edu.cn

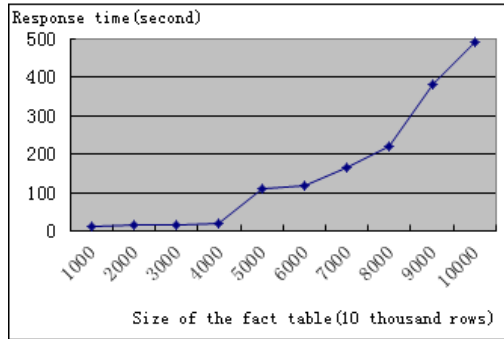
**Abstract.** Real-time OLAP applications require better performance for data ETL (Extraction, Transformation, Loading) and OLAP queries, and parallel processing via multiple nodes has become a research hotspot in recent years. Focusing on performance bottlenecks of current OLAP applications, this paper discusses a light-weighted parallel approach with a performance prediction and data calibration model according to node characteristics, based on which large-scale fact tables are partitioned to multiple nodes by parallel ETL process, and computation for OLAP queries are concurrently performed on the nodes with data fragments. The approach is implemented based on an open-source OLAP engine Pentaho Mondrian, and verified by experiments and a business project. According to the results, the method can effectively improve the performance of both OLAP queries and data ETL tasks for large-scale high-dimensional fact tables by decentralizing the DBMS workload brought by complicated SQL queries and data loading.

**Keywords:** Real-time Data Warehouse, OLAP; Parallel Processing, Performance Awareness, Data Partitioning.

## 1 Introduction

In recent years, with the increasing requirements for business intelligence, both application domains and patterns of data warehouse are being extended. Besides supporting strategic decisions for enterprises with subject-oriented, stable, integrated and historical data set [1], data warehouse is also used to provide ad-hoc OLAP processes based on real-time business data, which is considered as a new development trend in the domain of data warehouse technologies. With the advantages of relational database in technical maturation, manipulation flexibility and wide application, ROLAP(Relational OLAP) can provide efficient approaches of information modeling access and operation for data analysis and decision supports [2]. From the viewpoint of SQL, a ROLAP query usually involves multiple complicated database operations including table joining, aggregation, and grouping, which is a typical computation-intensive task. Therefore, performance becomes one of the most important issues in real-time ROLAP applications which concerns both users and developers.

In a real-time OLAP business project of supermarket sales and labor cost analysis in which our research team was ever involved, we observed the performance changes of MDX queries and ETL processes with the increasing size of their fact tables. For a data cube with 12 dimensions and 8 measures, figure 1 shows the performance curve of a MDX query with all of the dimensions. Once the size of the fact table exceeds a particular value, such as 50 million rows, there is a sharp decline of the OLAP query performance.



**Fig. 1.** Performance of OLAP query with different size of fact tables

According to the analysis of more experiment results, with the increasing size of fact tables in OLAP model, the physical abilities such as CPU, memory and I/O limit the query and process performance of the DBMS server. As a result, the large number of complicated SQL statements generated by OLAP engine becomes the bottleneck of OLAP queries.

Performance has always been one of the critical issues in OLAP applications. Recently, parallel process technologies using multiple nodes for data warehouse have become research hotspot, which mainly focus on parallel computation of data cubes. Parallel methods break through the ability bottleneck of single storage or processing node, and greatly improve the performance of data warehouse applications by distributing data or dispatching tasks among multiple nodes. However, Current approaches have limitations in several aspects, such as architecture complexity, usability and automatic optimization capabilities. For instance, the greatest disadvantage of parallel database is constructing and tuning difficulty, as well as the lack of complete low-cost and open-source solutions [4]. On the other hand, Map/Reduce-based methods usually assume equal abilities of different nodes to simplify process model, which ignore the abilities of forecast and adjustment based on nodes with different characteristics in their parallel processing models. Additionally, there has not been any effective direction for traditional centralized ROLAP applications to transfer to parallel process patterns.

In this paper, we propose a light-weighted parallel process approach for data warehouse applications with large-scale and high-dimensional data cubes which can efficiently improve performance of both data ETL tasks and MDX queries by decentralizing the load on single server. Our method covers the whole procedure of OLAP applications including data ETL and query process. The contributions of our

work include: (1) we propose a data partition strategy with dynamic optimization ability based on node characteristics, as well as a parallel data ETL mode. (2) Then, we give a parallel process algorithm for OLAP queries based on multiple data nodes. Our approach is implemented and verified based on Pentaho Mondrian, a popular open-source OLAP engine, and applied in a real-time data warehouse business project.

Following is the organization of this paper. Section 2 gives the details of our ROLAP parallel process model including both data ETL and multiple dimensional queries. In section 3, we depict the performance forecast model and calibration algorithm based on node characteristics. In section 4, implementation of the prototype is introduced and some experiment results are discussed. Section 5 summarizes recent research works related to our studies. Last section (Section 6) concludes this paper.

## **2 Parallel Process Model for Relational OLAP**

### **2.1 Overview**

Firstly, we propose the performance forecast and control model based on node characteristics, and design parallel data ETL approach which is used to load data from different sources to multiple OLAP partition nodes. Then, we add parallel process ability for multi-dimensional queries to traditional OLAP engine, so that the workload of single node for OLAP and DBMS process is reduced. At last, the parallel process methods for data load and multi-dimensional query are integrated to form the whole solution for efficient OLAP applications.

The system structure is shown in figure 2, which includes two parts of data ETL and multi-dimensional query process based on multiple decentralized nodes of data partitions. Each data node is deployed with relational DBMS to store and manage a data partition of fact tables. To simplify the algorithm of result merging, row-partitioned strategy is adopted, which store data segments of the fact table with the whole schema in each data node, as well as all of the dimension tables. Meta-data describe useful information required by the system, including description of fact tables and their data segments, such as the data nodes, number of rows distributed on each node, characteristics and workload of each node etc.

#### **(1) Process Structure for Data ETL**

The parallel process structure of data ETL is a master/slave pattern, which consists of a master node in charge of scheduling ETL tasks and multiple slave nodes responsible for executing sub ETL tasks. The task scheduling node creates detailed ETL plan for a fact table, assigns sub tasks to the task executing nodes and records meta-data according to data distribution of the fact table once the ETL task is completed.

Our strategy of decentralized storage for high-dimensional large-scale fact tables is based on neither static nor equal distribution. Instead, the performance evaluation and forecast model is constructed based on node characteristics and data partitions, which directs data ETL task scheduling node to assign sub tasks with different data size to the task scheduling nodes.

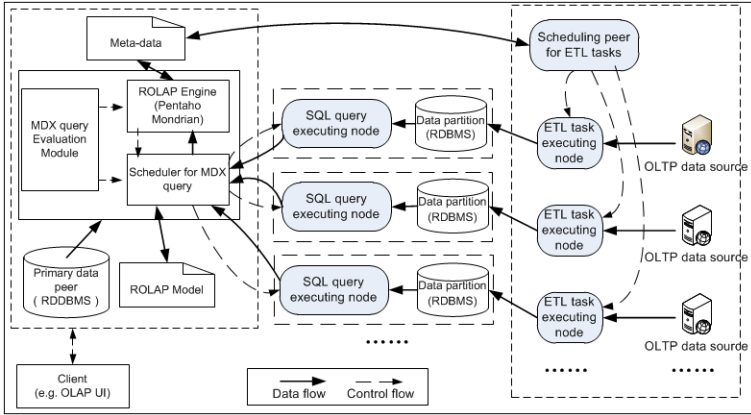


Fig. 2. Performance of OLAP query with different size of fact tables

(2) Process Structure for Multi-dimensional Queries

The other core part of our method is reconstructing traditional OLAP engine to support parallel process for multi-dimensional queries, which also consists of a master node and multiple slave nodes for executing sub query tasks. Based on traditional OLAP engine, query evaluation module and query scheduling module are added. Query evaluation module intercept and evaluate the SQL statements before they are submitted to DBMS, which are generated by OLAP engine according to multi-dimensional OLAP queries. The costly SQL statements will be submitted to query scheduling module to parallelize the SQL query task. Query scheduling module creates sub tasks, assigns them to data nodes of the fact table, and merges the results returned from the nodes. Then, the merged result will be submitted to OLAP engine to figure out the final multi-dimensional cube in memory.

Query executing nodes receive and execute SQL query tasks based on data partitions. The query executing node can be deployed on data nodes to eliminate the cost of data transferring between different nodes, as there is little workload on query executing nodes brought by computing and complicated logic.

2.2 Data Distribution Model

We use row-partitioned strategy for high-dimensional large-scale fact tables, which means each data node contains data segments with the whole schema, along with all of the required dimension tables.

(1) Strategy for Partitioning Data

Instead of equal distribution of data segments, we use the strategy based on node capabilities for partitioning fact tables.

Assuming there are  $n$  data nodes:  $p_1, p_2, \dots, p_n$ . We use eigenvector  $(f_1, f_2, \dots, f_m)$  to represent the capabilities of current node, in which each component measures a



particular characteristic of the node, such as main memory, processor number, speed of CPU and I/O etc. Function  $f(i, j)$  means the value  $f_j$  for node  $p_i$ . E.g. if the memory size of node  $p_i$  is 2 gigabytes, then  $f(i, j) = 2$ .

Then, we use  $N(i, j)$  to denote the measure value by unitizing  $f(i, j)$ :

$$N(i, j) = \frac{|f(i, j) - \min(j)|}{|\max(j) - \min(j)|} \tag{1}$$

Based on the above definition, we propose the evaluation function  $cap$  for node capabilities:

$$cap(i) = \sum_{j=1}^m N(i, j) * w(j) \tag{2}$$

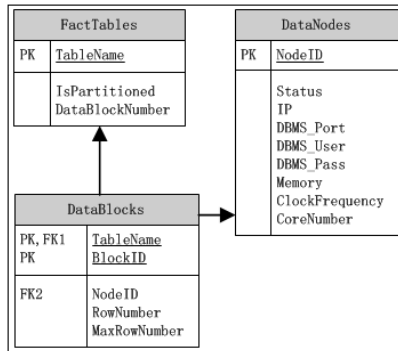
Where  $w(j)$  represents the weight value of performance characteristic  $f(j)$ .

Assuming there are  $R$  rows in fact table  $T$ , we compute the expected size  $R(T, i)$  of data partition stored on node  $I$  based on above evaluation function:

$$R(T, i) = \frac{cap(i)}{\sum_{j=1}^n cap(j)} * R \tag{3}$$

**(2) Meta-data of Data Partitions**

Data partition information is saved as meta-data relations, whose concept view is shown in figure 3. Table *factTables* describes fact tables in defined OLAP models, in which attribute *IsPartitioned* means whether current table is partitioned and *DataBlockNumber* denotes its partition number. Data partition meta-data *DataNodes* contains detailed information of each node including the status, address, DBMS connection information and characteristics related to performance. *DataBlocks* describes data partitions of fact tables including location and partition size, which is required by OLAP the query task scheduling module and updated by the data ETL task scheduling node.



**Fig. 3.** Concept model of meta-data for data partitions

The meta-data is embedded into the cube definition in traditional OLAP model. An example of the extended OLAP model is shown in figure 4.

```

<Cube name="OPERATION_TIME" cache="true">
  <table name="OP_TIME" isPartitioned="true" dataBlockNum="4">
    <block id="1" nodeId="server06" rowNumber="16000000" />
    <block id="2" nodeId="server07" rowNumber="16500000" />
    <block id="3" nodeId="server11" rowNumber="14000000" />
    <block id="4" nodeId="server09" rowNumber="11000000" />
  </table>
  <!--ORG Dimension -->
  <DimensionUsage name="Org" source="Org" foreignKey="org_id" />
  <!--Time Dimension-->
  <DimensionUsage name="Time" source="Time" foreignKey="calendar_id" />
  . . . . .

```

Fig. 4. Extended OLAP model fragment

### 2.3 Parallel Process Approach for OLAP Queries

Based on the data distribution model, performance of OLAP queries can be improved by decentralize the workload of computing and I/O from single DBMS server to multiple nodes with smaller data partitions.

#### (1) Distributed Computing Model for Aggregation Functions

The result of any OLAP query can be expressed as a cube aggregated based on particular dimension structure and hierarchies [3]. As mentioned in the beginning of this paper, the SQL statements with many aggregating and group operations generated by OLAP engine account for the majority of latency time.

Almost each aggregation function based on relational algebra has distributed computable feature, which means the computation can be performed in 2 phases: (1) aggregation operation is executed based on multiple data nodes respectively; (2) The final result is merged based on multiple aggregation values. We enumerate and analysis the OLAP aggregation functions provided by popular OLAP engine such as Server Analysis Service and Pentaho Mondrian, and construct distributed computing model for each aggregation function. Assuming the partitions of fact table  $T$  are distributed on  $n$  nodes with sub tables  $T_1, T_2, \dots, T_n$ , and function  $f(T_i, c)$  is used to denote the aggregation result for sub table  $T_i$  according to dimension attribute  $c$ .

E.g. for function *AVERAGE*, we have

$$AVERAGE(T, c) = \sum_{i=1}^n SUM(T_i, c) / \sum_{i=1}^n COUNT(T_i)$$

Similarly, other aggregation functions in OLAP queries, such as *COUNT*, *DISTINCT-COUNT*, *MAX*, *MIN*, *TOP(k)*, can be concurrently computed based on their distributed computable feature. For nested functions, the functions in the innermost layer are concurrently performed on data partitions, then the aggregation functions in outer layer is computed based on merged results.

#### (2) Parallel Process Algorithm for OLAP Queries

Based on the distributed computing model for OLAP aggregation functions, we propose a parallel process algorithm for OLAP queries which consists of several

phases including decomposing query task, concurrently performing sub tasks and merging results.

The process structure is based on master/slave pattern as shown in section 2.1, which is implemented by extending open source OLAP engine Mondrian. In the following we give the brief description of the algorithm.

// scheduling control algorithm for OLAP queries:

```

ParallelOLAPQuery(MDXStatement x) {
  OLAP engine analysis x based on OLAP model;
  Set<<SQL-Statement, Result>> resultset = null;
  for (each SQL statement s generated by OLAP engine) {
    resultset.add(<s, null>);
    if (s is a query for partitioned fact table) {
      Object[] d = dimension columns in s.select clause;
      Object[] m = aggregation functions in s.select clause;
      for (each aggregation function agg_func in m) {
        // kv_set is sub-queries set
        kv_set.add((m.agg_func, null));
      }
      //invoke Mapper function to assign sub query tasks
      for (each node p of current table) {
        assign TaskMapper(kv_set, s) to node p;
      }
      //start a thread to asynchronously receive and merge results from multiple nodes
      Reducer(kv_set, result);
      Program will be blocked until all sub tasks are completed;
    }
    else {
      //if current table is not partitioned
      result = DBMS_Execute_SQL(s);
    }
    resultset.put(<s, result>);
  }
  OLAP engine computes current cube based on resultset;
}

```

// create and assign sub query tasks to data nodes:

```

TaskMapper(Set<key k, values v> kv_set, SqlStatement s) {
  for (each aggregation function agg_func in kv_set) {
    //perform query s on local DBMS;
    result = DBMS_Execute_SQL(s);
    kv_set.put((m.agg_func, result));
  }
}

```

//reducer thread, used to merge the results from a data nodes

```

Reducer(Set<key k, values v>input, Object result) {
  Merge input into result according to aggregation function type;
}

```

The process time of above algorithm is the maximal SQL executing time among  $n$  data nodes, plus merging time for the sub results. The algorithm uses a similar

mechanism to Map/Reduce which supports concurrent process for both query executing on the nodes and results merging on the master node. Due to the constraints of where and group-by clauses, size of the returned results returned is much smaller than its fact table. As a result, cost of merging operation in the algorithm is very low.

### 3 Node Performance Forecast and Calibration Model

In section 2.2, based on the data distribution strategy, expression 3 gives the expected value  $R(T, i)$  of partition rows on data node  $p_i$  for fact table  $T$ , which is applied to initial data partitioning based on static node characteristics. In data warehouse applications, data from multiple sources will be continuously loaded to OLAP system. Considering the changes of the number and configurations of data nodes, we propose the node performance forecast and calibration model, which is used to adjust the data partitioning strategy among data nodes dynamically.

(1) First, the performance forecast model for each node is constructed.

Based on the performance data retrieved from the SQL executing logs with different data partition sizes on current node, performance sampling data can be expressed:

$$PS = \{(r_1, t_1), (r_2, t_2), \dots, (r_n, t_n)\}$$

using linear least square method. The following 2 fitting functions are implemented by MatLab via mechanism analysis:

$$t(r) = a1 \times r^3 + a2 \times r^2 + a3 \times r + a4,$$

$$t(r) = a1 \times e^{a2 \times r}$$

The fitting function with smaller deviation from actual sampling data is selected as the performance forecast function for current node.

(2) Secondly, the capability bottleneck of each node is forecasted according to the performance fitting curve.

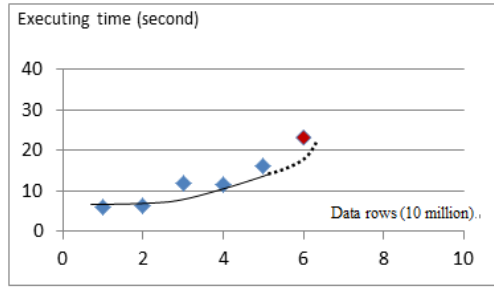
For a particular node, the performance bottleneck is identified by the point with a sudden steep gradient on the performance fitting curve, which means the latency time increases rapidly once the size of data partition reaches a particular value.

Assuming the cubic polynomial  $t(r) = a1 \times r^3 + a2 \times r^2 + a3 \times r + a4$  is selected as fitting curve, its second time derivative can be computed as:

$$t'(r) = 6 \times a1 \times r + 2 \times a2 \times r$$

Then the  $r$  value of the inflexion is computed according to the equation  $t'(r) = 0$ . If the third derivative of the inflexion is greater than 0, it can be defined the mutation point. Similar method can be applied to other fitting functions to identify mutation points.

For example, figure 5 shows a node performance fitting curve according to some sample data of SQL executing logs. If the latency time of queries on current node exceeds 20 seconds, the node is considered the bottleneck of the whole OLAP



**Fig. 5.** A node performance fitting curve based on sampling data

process. Then, the limit of data partition size on current node can be forecasted based on above method with the constraints of the overall performance.

- (3) Lastly, the expected value of data partition size for each node is updated into the parallel OLAP meta-data according to the forecasting results of its performance bottleneck, which is used to direct data ETL task assignment and scheduling.

## 4 Applications and Experiments

The parallel process approach discussed in this paper is implemented and applied in a business project of real-time OLAP for supermarket sales and labor cost analysis. The parallel OLAP process is implemented based on the kernel of a traditional OLAP engine Pentaho Mondrian. Data portioning and ETL parallel process for large-scale fact table is realized using Java platform and similar Map/Reduce pattern.

In the experiments, a fact table with about 200 million rows and other dimension tables are selected to construct an OLAP model with 12 dimensions and 6 measure attributes. An OLAP query including all of the dimensions and measures are selected, which is shown in following.

```
select {[Measures].[VOLUME], [Measures].[Total Time], [Measures].[Operation Time],
[Measures].[UOM Per Time]} on 0,
CrossJoin(CrossJoin(CrossJoin(CrossJoin(Descendants([Org],[Chain],4),[Org_Type],[AllType].Children),CrossJoin([Time],[Y2011],[Y2011-Q1],[Y2011-Q1-P1].Children,[Job].Children)),
CrossJoin(CrossJoin([OperationType].AllMembers),{[Task].Members}),{[Volume_Driver],[Driver].Members})),[Operation].[AllOperation].Children) on 1 from [OPERATION_TIME]
```

To avoid the effect of main memory cache on query performance, the cache mechanisms of DBMS and OLAP engine are disabled.

### a) Executing Time of Different Phases for OLAP Queries

The total executing time of an OLAP query composes of several parts: 1. Time of preprocessing by OLAP engine including syntax parsing and lexical analysis for multi-dimensional query statement, along with generating SQL queries. 2. Executing time of the generated SQL statements. 3. Duration of computing cube based on the results of the SQL queries. The results of this experiment are illustrated in figure 6 and 7.

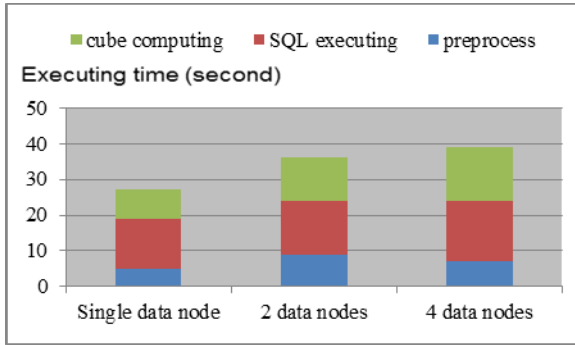


Fig. 6. Composition of OLAP executing time for 40 million rows

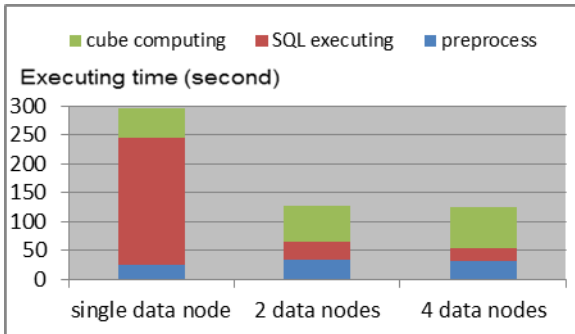


Fig. 7. Composition of OLAP executing time for 90 million rows

According to the results, executing time of the SQL queries for DBMS forms the majority of the total latency time. Once the data size of a node reaches its border value, the performance bottleneck appears. Parallel process with multiple data nodes can decentralize the workload of single DBMS server. With the increasing data size of the fact table, performance of SQL queries has an obvious improvement.

**b) Executing Time with Different Data Size.**

By constructing data for tact table *OPERATION\_TIME* with different size, this experiment compares the difference between traditional OLAP process with single DBMS node and parallel approach with multiple nodes, which is shown in figure 8.

For fact table with small data size, the parallel process approach has lower performance than traditional process with single node due to extra operations including task scheduling, assignment and merging multiple results. However, the workload of single DBMS node increases continuously. As a result, the performance becomes unacceptable once data size reaches a particular value. Our parallel approach eliminates this bottleneck by decentralizing the workload by partitioning data on multiple nodes.

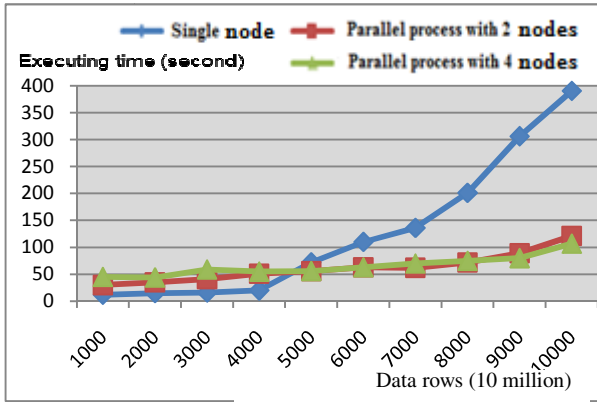


Fig. 8. Executing time with different size of fact table

**c) Executing Time for Different Data Distribution Strategies**

This experiment compares the query performance of 3 different data distribution strategies: even distribution, distribution based on static node characteristics and distribution based on performance prediction and Calibration model. To verify the effect of different strategies and environments on query latency, multiple nodes with different configurations are selected, and changes are applied to the nodes during the procedure of this experiment.

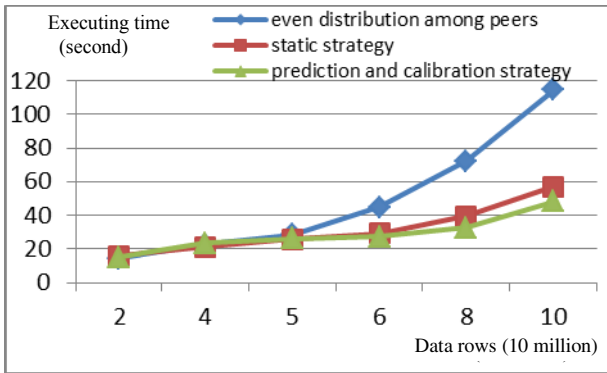
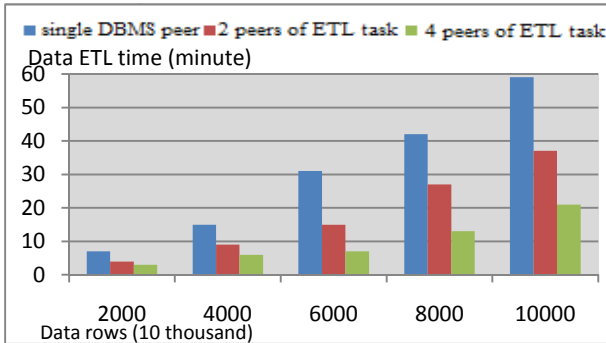


Fig. 9. Comparison among data distribution strategies

In the case of fact table with small data size, there are no significant differences among 3 strategies. With increasing data amount of the fact table, performance of the even data distribution strategy gets worse because some nodes becomes executing bottleneck due to the difference of performance-related characteristics such as memory, CPU and I/O speed. In dynamic environments where the number and configurations of nodes may be changed, the strategy of performance prediction and calibration shows advantages than static distribution model.

#### d) Data ETL for Large-Scale Fact Tables

This experiment verifies the performance of data ETL (Extraction, Transformation and Loading) for large fact tables, which is shown in figure 10.



**Fig. 10.** Performance of data ETL for large fact tables

According to the experiment, the parallel data ETL process has obvious advantages because of the simplicity of parallelizing data ETL tasks without extra costs of coordinating sub-tasks and merging sub-results.

On the other side, many indexes are usually created for high dimensional fact tables to optimize SQL queries, which have obvious effect on data ETL tasks. Some compromises should be adopted to balance database queries and data ETL tasks.

## 5 Related Works

Current research works on parallelizing OLAP focus on concurrently computing cubes using multiple nodes, which can be categorized into parallel processing methods based on shared data and partitioned data. The main idea of cube computing based on shared data is to break down the whole calculation task and assign sub-tasks to multiple nodes, which focuses on extendibility or load balance of tasks [5, 6]. Data partition based approaches place data segments on different nodes according to a particular strategy. Computing tasks are performed based on local data, and the whole cube is figured out by merging all of the local results [7].

In recent years, SN pattern (shared nothing architecture) has become a research hotspot because of good scalability and extendibility. Literature [8] studied a parallel computing method for data cube running on a shared-nothing architecture. In the following studies, the authors enhanced their cube computing methods by introducing load balance among nodes and improving the parallelization of tasks on multiple nodes [9].

Generally speaking, data partition based on shared nothing architecture can achieve better performance improvement because of sub tasks with smaller data size. However, most of current researches ignored the differences among nodes in their capabilities. E.g. the studies in paper [9] assumed the precondition that data partitions



are evenly distributed among the nodes. Recently, some researchers noticed this point and began to perform researches on performance prediction [10, 11].

Moreover, current works on parallel process for data warehouse focus on cube computing in main memory, instead of looking into detailed factors affecting the whole OLAP performance. Actually, for OLAP applications with large fact tables and high dimensions, the cost of DBMS-side SQL queries even exceeds that of cube computing.

Map/Reduce programming model is drawing more and more attention in the domain of analysis and computing for large-scale data set. Some researchers carried out systematic analysis and evaluation for the advantages and disadvantages of parallel database and Map/Reduce technologies in data warehouse applications [4]. HadoopDB [12] is a hybrid structure which integrates the high performance of DBMS and the scalability of Map/Reduce architecture.

## 6 Conclusions

In this paper, we discuss a light-weighted parallel process approach for the entire procedure of data warehouse applications with large-scale and high-dimensional data. Our method covers the whole procedure of OLAP applications including data ETL and query process. Firstly, we propose a data partition strategy with performance prediction and calibration model based on node characteristics, along with a parallel data ETL architecture which partitions large-scale high-dimensional fact table to multiple data nodes. Then, we give a parallel process algorithm for OLAP queries based on multiple data nodes. Our approach is implemented and verified based on Pentaho Mondrian, a popular open-source OLAP engine, and applied in a real-time data warehouse business project. According to the benchmark testing and applications, the approach can effectively decentralize the DBMS workload brought by complicated SQL queries and data loading, and improve the performance of both MDX queries based on high-dimensional large-scale fact tables and data ETL tasks.

The works in this paper only focuses on improvement on DBMS-side performance. In future researches, we plan to improve cube computing by integrating some latest results of computing and selecting materialized views into current works.

**Acknowledgement.** This work is supported by the National Natural Science Foundation of China under Grant No. 61003253 and Shandong Distinguished Middle-aged and Young Scientist Encouragement and Reward Foundation under Grant No. BS2010DX016.

## References

- [1] Inmon, W.H.: Building the data warehouse, 4th edn. Wiley, New York (2005)
- [2] Li, W.H., Feng, Y.C., Ma, X.M., Fu, Q., Hu, W.B.: ROLAP-Oriented Temporal Vertical Partitioning Method Based on Rough Sets. Chinese Journal of Computers 31(7), 1109–1121 (2008)

- [3] Zhang, Y.S., Zhang, Y., Huang, W., Wang, S., Chen, H.: Distributed Aggregate Functions Enabled Parallel Main-Memory OLAP Query Processing Technique. *Journal of Software* 20(suppl.), 165–175 (2009)
- [4] Stonebraker, M., Abadi, D., DeWitt, D., Madden, S., Paulson, E., Pavlo, A., Rasin, A.: MapReduce and parallel DBMSs: friends or foes? *Commun. ACM* 53(1), 64–71 (2010)
- [5] Dehne, F., Eavis, T., Hambrusch, S., Rau-Chaplin, A.: Parallelizing the data cube. *Distributed and Parallel Databases* 11(2), 181–201 (2002)
- [6] Dehne, F., Eavis, T., Rau-Chaplin, A.: A cluster architecture for parallel data warehousing. In: *Proc. IEEE International Conference on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia (2001)
- [7] Goil, S., Choudhary, A.: High performance OLAP and data mining on parallel computers. *Journal of Data Mining and Knowledge Discovery* 1(4), 391–417 (1997)
- [8] Chen, Y., Dehne, F., Eavis, T.: Parallel ROLAP Data Cube Construction on Shared-Nothing Multiprocessors. *Distributed and Parallel Databases* 15, 219–236 (2004)
- [9] Dehne, F., Eavis, T., Rau-Chaplin, A.: The cgmCUBE project: Optimizing parallel data cube generation for ROLAP. *Distributed and Parallel Databases* 19(1), 29–62 (2006)
- [10] Duggan, J., Cetintemel, U., Papaemmanouil, O., Upfal, E.: Performance prediction for concurrent database workloads. In: *SIGMOD* (2011)
- [11] Popescu, A.D., Ercegovac, V., Balmin, A., Branco, M., Ailamaki, A.: Same Queries, Different Data: Can we Predict Runtime Performance. In: *ICDEW* (2012)
- [12] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D.J., Silberschatz, A., Rasin, A.: HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. In: *VLDB* (2009)

# A Parallel Multigrid Poisson PDE Solver for Gigapixel Image Editing

Zhenlong Du, Xiaoli Li, Xiaojian Yang, and Kangkang Shen

College of Electronics & Information Engineering, Nanjing University of Technology,  
Nanjing 210009, P.R.China  
duzhlcad@gmail.com

**Abstract.** With the development of image acquisition technology, gigapixel images are easily produced and widely used in modern society. How to efficiently compile these gigapixel images within gradient domain is the research focus in the community of image processing and computer graphics. To solve Poisson equations involving large-scale unknowns is crucial for gigapixel image editing in gradient domain. Traditional multigrid approach separately performs iteration, restriction and interpolation, bears heavy communication costs between RAM and external memory. In the paper, a parallel multigrid Poisson solver for gigapixel image editing is proposed, which exploits the locality and relevance of memory accessing and updating among the iteration, restriction and interpolation for parallel performing the iteration, restriction and interpolation in the sweeping window. Image stitching experiments show that the presented method exhibits the higher efficiency than the Poisson solver of successive overrelaxation, gauss-seider iteration and traditional multigrid.

**Keywords:** Poisson PDE solver, parallel multigrid, gigapixel image editing.

## 1 Introduction

Poisson partial differential equation (PDE) is one kind of elliptic PDE which is widely used in community of science and engineering, such as machinery, physics, information, etc. Since the introduction of Poisson PDE to the image editing by Prez [1], image editing within gradient domain based on Poisson PDE becomes the research focus of image editing, such as, image cloning and composition [2,8], photo montage [3], matting [4], all achieved the photorealistic editing effect. However, with the development of digital acquisition technology, the resolution of acquired images have been increasing, so the space and time complexity in image editing based on Poisson PDE becomes more and more higher. Therefore, the investigation of fast Poisson solver is very significant.

Iterative Poisson PDE solver converges fast in the high frequency region but slow in the low frequency region which easily leads to the solution to fall into the local smooth region, therefore, it is unsuitable for solving Poisson PDE with

large-scale unknowns. Multigrid Poisson solver respectively smoothes the error residual in different frequency levels, and has better efficiency than iteration approach, hence is widely used in gigapixel images editing within gradient domain. But the traditional multigrid approach separately performs iteration, restriction and interpolation, not fully exploits the relevance among data in different stages, bears heavy communication load between RAM and external memory.

A parallel [7] multigrid Poisson solver [6] is proposed which exploits the locality and relevance of memory accessing and updating among the stage of iteration, restriction and interpolation. The presented method in [6] parallel performs the iteration, restriction and interpolation in the sweeping window which effectively reduce the communication load between RAM and external memory and increase the processing efficiency. In the paper a parallel multigrid Poisson PDE solver is presented, which has better performance in solving Poisson equations with large-scale unknowns. The proposed method is testified in gigapixel image editing within gradient domain and demonstrates the high efficiency than iterative Poisson solver.

## 2 Poisson PDE Solver

The general form of binary Laplace PDE  $U(x, y) : R^2 \rightarrow R^2$  ( $R$  is the real set) is as the follows.

$$\Delta U(x, y) = \nabla \cdot \nabla U(x, y) = f(x, y) \quad (1)$$

Where  $\Delta$  is Laplace operator,  $\nabla$  is partial derivative operator, and  $f(x, y)$  is the boundary condition. Eq.(1) is the Poisson PDE with Neuman boundary when  $f(x, y) = 0$  holds, and it is the Poisson PDE with Dirichlet boundary when  $f(x, y) = c$  ( $c$  is a constant) is satisfied.

In the domain  $\Omega = R \times R$ , the purpose of image editing within gradient domain is to make  $U(x, y)$  best close to the gradient field  $\vec{G}(x, y)$ , it is equivalent to the functional  $\min \| U - \vec{G} \|$ , and the Poisson solver is able to be derived and expressed as Eq (2).

$$\Delta = \nabla \cdot \nabla \vec{G} = \text{div} \vec{G} \quad (2)$$

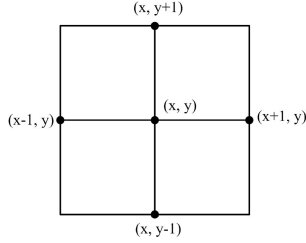
where  $\text{div}$  is the divergence operator.

The numerical solver of Eq. (2) is firstly converted to the discrete representation. There exists many discretization schema, the five-point discretization is represented as the follows.

$$U(x+1, y) + U(x, y+1) + U(x-1, y) + U(x, y-1) - 4U(x, y) = \text{div} \vec{G}(x, y)$$

For each pixel  $(x, y)$  within the image domain  $\Omega$ , there is a linear equation. The equations of all pixels form an linear equations, which is denoted by  $\mathbf{L}\mathbf{U} = \mathbf{B}$ , in which  $\mathbf{L} = (a_{ij})_{n \times n}$ ,  $\mathbf{U} = (u_i)_{n \times 1}$  and  $\mathbf{B} = (b_i)_{n \times 1}$  are separately the Laplace matrix, unknowns matrix and boundary matrix.

There are many numerical solving methods for Poisson equation such as iteration [5] (including gauss-seider iteration, Jacobi iteration and conjugate gradient



**Fig. 1.** Five-point discretization

iteration), multigrid [5] and discrete cosine transform, etc. When the number of unknowns in domain reaches  $10^4$ , the size of  $\mathbf{L}$  is  $10^4 \times 10^4$ . For the large-scale image (the number of unknowns are greater than  $10^6$ ),  $\mathbf{L}$  is  $10^6 \times 10^6$ . Though  $\mathbf{L}$  is the banded matrix, and generally it is sparse, the large-scale unknowns still deteriorates the iteration speed and convergence performance.

### 3 Multigrid Poisson PDE Solver

Multigrid Poisson PDE solver acquires the solution in hierarchical manner, which iterates and smoothes the high-frequency error residual in current level and continually smoothes the low-frequency residual in the next low-resolution level. The error residual is restricted from the higher resolution level to the next lower resolution one. The essence of the multigrid Poisson PDE solver is that smoothing and iterating the high-frequency error residual in the low-resolution level which are again restricted to the lower resolution level, repeats the process until the error residual is sufficiently smooth. Each iteration at a low-resolution level provides a more accurate calibration result for the next high-resolution level. Multigrid Poisson Solver could quickly smooth the high-frequency residual existing in different frequency spectrums, hence it efficiently accelerates the convergence procedure.

Let  $h$  be the discretization length. Linear equations discretized from Poisson PDE at  $h$  level is expressed as the follow.

$$\mathbf{L}_h \mathbf{U}_h = \mathbf{B}_h \quad (3)$$

Let  $\mathbf{U}_h$  and  $\overline{\mathbf{U}}_h$  separately be the accurate solution and approximate solution of Eq. (3),  $\mathbf{V}_h$  and  $\mathbf{d}_h$  separately be the error quantity between  $\mathbf{U}_h$  and  $\overline{\mathbf{U}}_h$  and error residual, which are defined as  $\mathbf{V}_h = \mathbf{U}_h - \overline{\mathbf{U}}_h$  and  $\mathbf{d}_h = \mathbf{L}_h \overline{\mathbf{U}}_h$ , respectively.

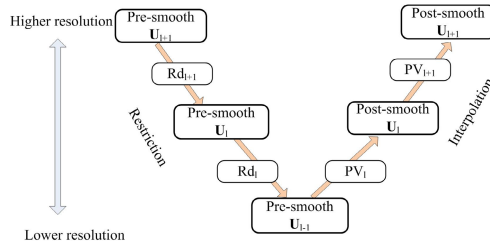
Multigrid Poisson solver performs in V-cycle manner, includes approximation, restriction and interpolation three procedures. It covers the following steps, solving the approximate solution  $\overline{\mathbf{U}}_h$ , restricting the error residual  $\mathbf{d}_h$  to the lower resolution level  $H$  by Eq. (4), acquiring  $\mathbf{V}_H$  by  $\mathbf{L}_H \mathbf{V}_H = -\mathbf{d}_H$ , returning the calibration to the higher resolution level  $h$  by interpolation, finally, solving the approximate solution at the highest resolution level, that is  $\overline{\mathbf{U}}_h^{new}$  ( $\overline{\mathbf{V}}_h$  is the

approximation of  $\mathbf{V}_h$ ). The restriction operator  $R$  and interpolation operator  $P$  used in multigrid solver are defined as Eq. (4) and Eq. (5) respectively.

$$\mathbf{d}_H = R\mathbf{d}_h \quad (4)$$

$$\bar{\mathbf{V}}_h = P\bar{\mathbf{V}}_H \quad (5)$$

The V-cycle in multigrid Poisson solver includes two procedures, one is a coarsening process from a high resolution level to the low resolution one, and the other is a refining process from the low resolution to the high resolution. Coarsening begins with the highest resolution level, restricts the error residual  $\mathbf{d}$  to the next low-resolution level. Coarsening polishes the error level by level. When the lowest resolution level is reached, the linear equations with the minimal number of unknowns is solved. Refining begins with the lowest resolution level and returns the calibration result from lower resolution level to higher resolution level via interpolation until the solution within the highest resolution level  $\mathbf{U}^{new}$  is achieved. Coarsening and refining separately correspond to the process of restriction and interpolation in the figure 2.



**Fig. 2.** The procedure of multigrid V-cycle

Both restriction and interpolation in V-cycle process need iteration, which gradually polishes the error. The iteration which performs before the restriction is named pre-smooth, and which does after the interpolation are called post-smooth.

### 3.1 Multigrid Poisson PDE Solver

Traditional multigrid Poisson solver acquires the solution of linear equations discretized from Poisson PDE in V-cycle manner. In the left half V-cycle of Figure 2, it solves the approximate solution after the fixed number of iteration, meanwhile restricts the error residual  $\mathbf{d}$  of each level to the lower resolution level through operator  $R$ . In the right half V-cycle of Figure 2, the algorithm returns the calibration result to the high resolution level through operator  $P$  by interpolation. The algorithm of traditional multigrid Poisson solver is as algorithm 1 depiction.

---

**Algorithm 1.** Multigrid Poisson PDE Solver

---

**Input:**  $k, N$

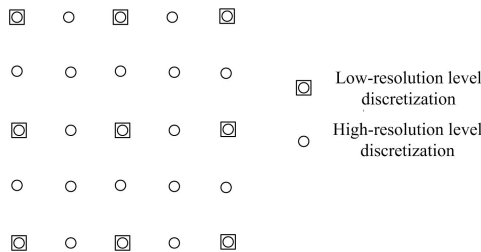
**Output:**  $\mathbf{U}$

1. Approximately solving  $\mathbf{L}\mathbf{U} = \mathbf{B}$ , and acquiring  $\overline{\mathbf{U}}_{old}$ .
  2. Restriction
    - Do  $h = N - 1, \dots, 2, 1$ 
      - 2a. Approximately solving  $\mathbf{L}_h\mathbf{U}_h = \mathbf{B}_h$  after  $k$  iterations, producing  $\overline{\mathbf{U}}_h$ .
      - 2b.  $\mathbf{V}_h \leftarrow \mathbf{U}_h - \overline{\mathbf{U}}_h, \mathbf{d}_h \leftarrow \mathbf{L}_h\overline{\mathbf{U}}_h - \mathbf{B}_h$ .
      - 2c.  $\mathbf{d}_h = \mathbf{R}\mathbf{d}_{h+1}$
    - End Do
  3. Solving  $\mathbf{L}_0\mathbf{V}_0 = \mathbf{d}_0$
  4. Interpolation
    - Do  $h = 2, \dots, N - 1, N$ 
      - 4a. Approximately solving  $\mathbf{L}_h\mathbf{V}_h = \mathbf{d}_h$  after  $k$  iterations.
      - $\mathbf{V}_h = \mathbf{P}\mathbf{V}_{h-1}$
      - If  $h = 1, \overline{\mathbf{U}}_h \leftarrow \overline{\mathbf{U}}_{old} + \mathbf{V}_h$
    - End Do
- 

Where  $k$  is the number of iterations and  $N$  is the number of multigrid levels.

## 4 Parallel Multigrid Poisson PDE Solver

The conventional multigrid poisson solver separately performs the process of approximation, restriction and interpolation. The data  $\mathbf{V}$  and  $\mathbf{d}$  at each level in the RAM need to be loaded twice, one is used for restriction, the other is for interpolation. Due to the limited capability of memory, only part of data could be loaded into RAM, the rest of data are gradually loaded into RAM according to the computation requirement. Therefore, for solving Poisson PDE with large-scale unknowns, traditional multigrid Poisson solver bears heavy communication between RAM and external memory, increases the operation time and reduces the computation efficiency.



**Fig. 3.** The discretization point relationship between low-resolution and high-resolution

The figure 3 demonstrates the discretization points relationship between the neighbouring low-resolution and high-resolution level. When solver runs, matrix  $\mathbf{L}$ ,  $\mathbf{B}$ ,  $\mathbf{U}$  and  $\bar{\mathbf{U}}$  load only one time in RAM, the elements in them could be indexed only according to the grid position when they are used in different resolution level. At each level,  $\mathbf{V}$  and  $\mathbf{d}$  are produced and used for transferring the linkage among different resolution level.  $\mathbf{d}$  evaluated in restriction is used for interpolation stage, therefore,  $\mathbf{V}$  and  $\mathbf{d}$  need to be resided in RAM. The total number of elements in  $\mathbf{V}$  and  $\mathbf{d}$  with  $N$  level is expressed as Eq. (6), where  $\|\cdot\|$  is for counting the number of elements in matrix. For the gigapixel image editing within gradient domain, the number elements in  $\mathbf{V}$  and  $\mathbf{d}$  are  $2 \times 10^7$ , the sum of elements in  $\mathbf{L}$ ,  $\mathbf{V}$  and  $\mathbf{d}$  would be  $5 \times 10^7$  when the multigrid level  $N$  is 5.

$$\sum_{h=1}^N \|\mathbf{V}_h\| = \sum_{h=1}^N \|\mathbf{d}_h\| \approx 10^7 \times (2 - 2^{1-N}) \quad (6)$$

Most of operations in multigrid V-cycle belong to the operation of matrix-matrix multiplication or matrix-vector multiplication which is suitable for parallelized. In this paper, by full exploitation the local accessing coherence of memory data in  $\bar{\mathbf{U}}_h$ ,  $\mathbf{V}_h$  and  $\mathbf{d}_h$ , the current accessed data is constructed a working set  $W$ , and then  $W$  is shifted along the image column direction for updating need of  $\mathbf{V}_h$  and  $\mathbf{d}_h$ . The shifting of working set is for making use of the data having been loaded in RAM.

The parallelization of parallel multigrid Poisson solver proposed in this paper embodies in two aspect, on one hand, the restriction and interpolation across different resolution level transfer with each other, on the other hand, the elements is maximally shared between the restriction and interpolation. The parallelization among different resolution level shows that, in the iteration, the updating is able to be executed when the required elements be ready. The parallelization between restriction and interpolation presents that, after finishing the restriction, when the required data get ready, interpolation could be performed. The solution could be achieved when the interpolation is accomplished at the highest resolution level.

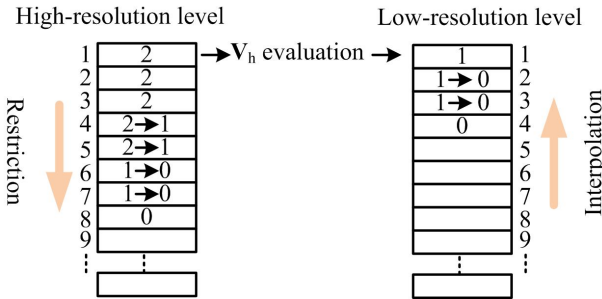


Fig. 4. The parallelization of parallel multigrid Poisson solver



In figure 4, suppose  $k = 2$  and  $N = 2$ , when the 6th and 7th execution cycle perform the restriction of  $\mathbf{d}_1 \rightarrow \mathbf{d}_0$ , the 4th and 5th execution cycle implement restriction of  $\mathbf{d}_2 \rightarrow \mathbf{d}_1$ . When the data in the first execution cycle of interpolation is updated,  $\mathbf{V}_0$  starts to be calculated and the 2nd and 3rd execution cycle could interpolate  $\mathbf{V}_1 \rightarrow \mathbf{V}_0$ .

Row of pixels is used as the operation unit for constructing working set  $W$ , which consists in the current processing row  $i$ , adjacent processed row and to be processed row. The processed row is utilized for updating the current row data, and the introduction of adjacent processed row is to calculate the error residual and calibration after updating the current row data.

**Table 1.** Data Window

|  |                           |
|--|---------------------------|
| restriction                                      |                           |
| $i_{h-1} + 2k + 1 < \lfloor (i_h - 1)/2 \rfloor$ |                           |
| data window                                      | $[i_h - 3, i_h + 2k + 1]$ |
| restriction from the resolution level of $h - 1$ | $[i_h + 2k + 1]$          |
| pre-smooth                                       | $[i_h - 1, i_h + 2k + 1]$ |
| error residual                                   | $[i_h - 3, i_h + 1]$      |
| interpolation                                    |                           |
| $i_{h+1} + 2k + 1 < 2i_h - 1$                    |                           |
| data window                                      | $[i_h - 1, i_h + 2k + 1]$ |
| interpolation to the resolution level of $h + 1$ | $[i_h + 2k + 1]$          |
| post-smooth                                      | $[i_h - 1, i_h + 2k + 1]$ |

Table 1 gives the data windows size in restriction and interpolation. Taking the restriction as an example, set  $i_h$  as the present processing line, when the  $2k + 1$  line of  $i_h$  in the  $h - 1$  level is finished updating, the  $h$  level could perform pre-smooth, therefore, the pre-smooth window is  $[i_h - 1, i_h + 2k + 1]$ . Since the smoothed line could be used for evaluating the error residual, the error residual window is  $[i_h - 3, i_h + 1]$ . The data window setting in interpolation is similar to the restriction.

The algorithm of parallel multigrid Poisson Solver is as the follow.

---

**Algorithm 2.** Parallel multigrid Poisson solver

---

**Input:**  $K, N, L$ (the number image rows)

**Output:**  $\mathbf{U}$

1. Evaluating the  $k - 1$  lines of  $\overline{\mathbf{U}}_h$  as the pre-processing data..
2. Do  $l = K, K + 1, \dots, L - K$ 
  - 2a. Evaluating  $\overline{\mathbf{U}}_h$  within  $[i_l - 3, i_l + 2k + 1]$ .
  - 2b. Calculating  $\mathbf{d}_h$  within  $[i_l - 3, i_l + 1]$ .
  - 2c. Acquiring  $\mathbf{V}_h$  within  $[i_l - 3, i_l + 1]$

End Do

3. Calculating  $\mathbf{V}_N$  on each level of  $l$  ( $l = K + 1, \dots, L$ ) and then updating  $\mathbf{U}$ .
-

The algorithm 2, parallel performs the iteration, restriction and interpolation which make good use of locality and relevance of memory accessing.

## 5 Experiment

The paper implements the parallel multigrid Poisson solver on dual-core PC computer of TongFang E2180 with 2G RAM. The presented method is used for image composition with a resolution of  $1280 \times 720$  or  $1024 \times 768$ .



**Fig. 5.** Image Composition (Data 1)



**Fig. 6.** Image Composition (Data 2)

The configurations of  $P$  and  $R$  exploited in the experiment are as the following, which are set by bilinear interpolation.

$$P = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \quad R = \frac{1}{4}P^T = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

In parallel multigrid Poisson solver, the resolution at low resolution level is 2 times to the neighbouring the high resolution level, that is to say,  $H = 2h$ , and the highest resolution is the size of source image.

The figure 5 and figure 6 are the composed image within gradient domain by the presented method in the paper. The registration [8] of all images is performed before composition. Figure 5 is a panorama with  $8100 \times 3680$  which is composed by 5 pieces of image with size  $1280 \times 720$ . Figure 6 is another panorama with  $7963 \times 3580$  which is composed by 5 pieces of image with size of  $1024 \times 768$ . The RAM usage, I/O Communication, iterations as well as operation time of the figure 5 and figure 6 are listed in the table 2.

**Table 2.** Comparison of different Poisson solver

| Algorithm           |       | RAM usage(M) | I/O Communication(M) | Iterations | Time (s) |
|---------------------|-------|--------------|----------------------|------------|----------|
| Over-relaxation SOR | Data1 | 56           | 7.53                 | 676        | 9.656    |
|                     | Data2 | 51           | 7.3                  | 615        | 8.433    |
| Jaccobi             | Data1 | 56           | 7.52                 | 1365       | 19.499   |
|                     | Data2 | 51           | 7.34                 | 1127       | 15.451   |
| Multigrid           | Data1 | 224          | 5.95                 | 232        | 3.315    |
|                     | Data2 | 204          | 5.53                 | 209        | 2.866    |
| Parallel Multigrid  | Data1 | 184          | 3.53                 | 165        | 2.357    |
|                     | Data2 | 180          | 3.36                 | 160        | 2.194    |

In table 2, the parallel multigrid Poisson solver is superior to the traditional algorithm on RAM usage, and is obviously superior to the overrelaxation iteration, Jacobi iteration and traditional multigrid algorithm.

## 6 Conclusion

Poisson PDE is a commonly used partial differential equation in the simulation and image processing. How to efficiently compile the gigapixel images within gradient domain is the research focus in recent years. Gigapixel image editing in gradient domain needs solving Poisson equation with large-scale unknowns. Traditional multigrid solver is not highly efficient on PC. Therefore, a parallel multigrid Poisson solver for gigapixel image editing within gradient domain is proposed, which exploits the locality and relevance of memory accessing and updating among the stages of iteration, restriction and interpolation for parallel performing. The approach could efficiently accomplish the gigapixel image editing within gradient domain on the PC machine.

Traditional multigrid Poisson solver separately performs iteration, restriction and interpolation which could not make good use of the locality and relevance of memory data. The presented method of parallel multigrid Poisson solver for gigapixel image editing has the higher efficiency, it better utilizes the locality and relevance of memory accessing and updating for parallel performing the iteration, restriction and interpolation.

The proposed approach of parallel multigrid Poisson solver is suitable for Poisson equation with Neumann boundary condition and structured data. Expanding the proposed approach for gigapixel image editing with complex gradient domain, namely for solving Poisson equation with Dirichlet boundary condition is our future work.

## References

1. Pérez, P., Gangnet, M., Blake, A.: Poisson Image Editing. *ACM Transactions on Graphics* 22(3), 313–318 (2003)
2. Levin, A., Zomet, A., Peleg, S., Weiss, Y.: Seamless Image Stitching in the Gradient Domain. In: Pajdla, T., Matas, J. (eds.) *ECCV 2004, Part IV. LNCS*, vol. 3024, pp. 377–389. Springer, Heidelberg (2004)
3. Agarwala, A., Dontacheva, M., Agarwala, M., Drucker, S., et al.: Interactive Digital Photomontage. *ACM Transaction on Graphics* 23(3), 294–302 (2004)
4. Sun, J., Jia, J., Tang, C.K., Shum, H.Y.: Poisson Matting. *ACM Transaction on Graphics* 23(3), 315–321 (2004)
5. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes in C*. Cambridge University Press (2002)
6. Kazhdan, M., Hoppe, H.: Streaming Multigrid for Gradient-Domain Operations on Large Images. *ACM Transaction on Graphics* 27(3), Article No. 21 (2008)
7. Chow, E., Falgout, R.D., Hu, J.J., Tuminaro, R.S., Yang, U.M.: A Survey of Parallelization Techniques for Multigrid Solvers. In: *Frontiers of Parallel Processing for Scientific Computing*. The Society for Industrial and Applied Mathematics (2006)
8. Szeliski, R.: Image Alignment and Stitching: A Tutorial. *Foundations and Trends in Computer Graphics and Computer Vision* 2(1), 1–104 (2006)

# Parallel Implementation and Optimization of Haze Removal Using Dark Channel Prior Based on CUDA

Yungang Xue, Ju Ren, Huayou Su, Mei Wen, and Chunyuan Zhang

School of Computer, National University of Defense Technology,  
Changsha, China  
xueyungangyun@163.com,  
{renju,shyou,meiwen,cyzhang}@nudt.edu.cn

**Abstract.** Haze Removal Using Dark Channel Prior is one of dehazing methods with good effects, but its disadvantage of high time complexity limits the extent of its applications. In this paper, we present its parallel implementation and optimization based on the GPU, which greatly reduces the execution time. We firstly implement the basic parallel program, and then optimize it to obtain performance acceleration with more than 20 times. We introduce the method of “guide image filter” to Haze Removal Using Dark Channel Prior, instead of “soft matting” method, which largely reduces memory requirements and the computation complexity of the original algorithm. While paralleling and optimizing the algorithm, we improve some parts of the original serial program or basic parallel program according to the characteristics of several steps, and propose a novel method of selecting atmospheric light and a new parallel method of calculating accumulative sum with keeping intermediate results, which reduce the computation complexity of counterpart and increase the parallel degree.

**Keywords:** Haze Removal, GPU, Parallel.

## 1 Introduction

Haze removal is an important issue concerned by both computer vision and image processing. Haze removal can improve the definition of haze image and correct the color distorted by haze [1]. Haze removal is widely used in lots of fields, such as automatic control system, autopilot and outdoor target recognition and so on. Haze in images will downgrade the effects of most algorithms in the field of computer vision, which assume that the input images are clear, therefore, we should remove haze in misty images before applying algorithms of computer vision to haze images.

Some algorithms of haze removal (or dehazing) are effective but take long time, so it is difficult for them to be widely used. Haze Removal Using Dark Channel Prior (HRUDCP, we will use this abbreviation for simplicity in the rest of our paper) is one of such algorithms, and it takes 10-20 seconds to process a 600x400 image. Recent researches on HRUDCP mainly focus on improving the effect and reducing computational complexity, related work can be found in [2,3,4,5], and parallel

research is simply mentioned only in paper [2]. The parallel research of HRUDCP is significant and necessary, only in this way can we quickly process haze high resolution images and even process haze videos in real time.

Strongly promoted by the applications of the graphics, image processing and video rendering, Graphics Processing Unit (GPU) has quickly developed in the past few years. With the advent of CUDA and OpenCL programming framework, GPU is not only as devices for display acceleration, but also as a coprocessor to accelerate general applications, and GPUS have succeeded in accelerating multitudinous algorithms in many fields. Therefore, In this paper, we choose GPU as the platform of the parallel implementation and optimization research.

This paper is organized as follows, section 2 introduces the each step of the original Haze removal dark channel prior and several changes in our paper. Section 3 presents the parallel implementation and optimization of HRUDCP based on CUDA. Section 4 analyzes and evaluates the parallel effect. The last section is conclusion.

## 2 HRUDCP and Parallelism Analysis

HRUDCP based on single image can obtain good effect and relatively accurate depth image information. The dark channel prior is based on statistics of the haze-free outdoor images. It is based on a key observation - most local patches in haze-free outdoor images contain some pixels which have very low intensities in at least one color channel [6]. In the haze image, the intensity of these dark pixels will be higher since all of its color channels are filled with fog. HRUDCP have several advantages: (1) good effect; (2) high quality depth map; (3) don't need additional information; (4) can automatically process. Meanwhile, it also has shortcomings to further overcome: (1) bad effect for the scene objects which are inherently similar to the atmospheric light and aren't cast on by shadows; (2) high computational complexity; (3) memory-consuming.

HRUDCP consists of five steps, whose function and detailed content are showed as follows.

(1) Computing dark channel. This step produces a corresponding dark value for each pixel of input image.

$$J^{\text{dark}}(\mathbf{x}) = \min_{c \in \{r, g, b\}} (\min_{y \in \Omega(\mathbf{x})} (J^c(y))) . \quad (1)$$

Where  $J^c$  is a color channel of  $J$  and  $\Omega(\mathbf{x})$  is a local patch centered at  $\mathbf{x}$ . In this paper, the patch size is the same with original algorithm, so the patch is  $15 \times 15$ .

(2) Estimating atmospheric light. We select a proper pixel in the input image as the atmospheric light. We firstly select the 0.1% of pixels of input image, which own brighter dark channel. and then we selected the pixel with highest intensity as the atmospheric light.

(3) Calculating the initial transmission. This step calculates the rough transmission rate based on dark channel image. Scene depth changes greatly at the edge of an object, so the transmission rate is not accurate enough.

$$t_0(x) = 1 - \omega \min_{c \in \{r, g, b\}} (\min_{y \in \Omega(x)} (\frac{I^c(y)}{A^c})) . \quad (2)$$

The last part of the formula is similar to the formula (1), the difference between them is the value of every pixel channel is normalized by dividing the corresponding channel value of atmospheric light  $A$ .  $\omega$  is a constant of 0.95.

(4) Refining the transmission. We abandon the soft matting method[7], which is used to refine the transmission in original algorithm, as it takes long time and need too much memory. In this paper, we adopt the method of “guided image filter” proposed by Doctor He Kaiming to refine the transmission rate.

$$a_k = (\Sigma_k + \epsilon U)^{-1} (\frac{1}{|\omega|} \sum_{i \in \Omega(x)} I_i p_i - \mu_k \bar{p}_k) . \quad (3)$$

$$b_k = \bar{p}_k - a_k^T \mu_k . \quad (4)$$

$$q_i = \bar{a}_i^T I_i + \bar{b}_i . \quad (5)$$

Here  $I_i$  is a  $3 \times 1$  color vector,  $a_k$  is a  $3 \times 1$  coefficient vector,  $q_i$  and  $b_k$  are scalars,  $\Sigma_k$  is the  $3 \times 3$  covariance matrix of  $I$  in patch  $\Omega(x)$ , and  $U$  is a  $3 \times 3$  identity matrix. In this paper, we also use the quick implementation of guided image filter [8] by Doctor He, which takes  $O(N)$  time.

(5) Haze removal. The haze removal image can be computed through the parameters obtained in above steps and input haze image.

$$I(x) = J(x)t(x) + A(1 - t(x)) . \quad (6)$$

Where  $I$  is the observed intensity in haze image,  $J$  is the intensity in haze removal image,  $A$  is the global atmospheric light, and  $t$  is the transmission rate.



**Fig. 1.** (a) Input image with haze (b) Result of original HRUDCP (c) Result of modified HRUDCP by introducing guide filter instead of soft matting in step of refining initial transmission. These show that modified HRUDCP is as good as original HRUDCP, and it is the main reason why we introduce the method of guide filter. We get the benefits that we can avoid the need of large memory and decrease the computational complexity of refining initial transmission.

We can know clearly that the main computing operation and the data dependence from the formula above, so we can analyze the parallelism of the algorithm. We analyze the parallelism of the algorithm using an image with width  $n$  and height  $m$ , the result is shown in the table 1, the max parallelism stands for the number of pixels that can be processed simultaneously.

**Table 1.** Parallelism analysis of HRUDCP

| step                               | Main operation  | Max parallelism          |
|------------------------------------|---|--------------------------|
| 1 Compute dark channel             | Local comparison                                      | mn                       |
| 2 Estimating atmospheric light     | Global comparison, sort                               | $\text{Log}_2(0.1\%*mn)$ |
| 3 Calculating initial transmission | Local comparison, four fundamental arithmetic         | mn                       |
| 4 Refine transmission              | Four fundamental arithmetic, 3 order matrix inversion | m or n                   |
| 5 Haze removal                     | Four fundamental arithmetic                           | mn                       |

We can conclude that HRUDCP algorithm can be paralleled from table 1. The parallelism of step 1,3 and 5 is highest, achieving the image resolution  $m \times n$ , the parallelism of step 2 is lowest, for a  $600 \times 400$  image, its parallelism is about 16. In this paper, we propose a novel method to select atmospheric light, which gets the almost same result but its parallelism can expand to  $mn/\log_2(mn)$ . The parallelism of step 4 is the height or width of the image, In this paper, we also propose a new parallel method of calculating accumulative sum with keeping intermediate results, and it expands the parallelism of step 4 from m or n to  $m\text{Log}_2(mn)$  or  $n\text{Log}_2(mn)$ .

### 3 Parallel Implementation and Optimization Based on GPU

The parallel programs based on the CUDA include serial codes and parallel kernels that work together as a CPU-GPU mode. The serial codes run on the host side, and the CUDA kernels run on the device side in multi-threaded way [9]. The GPU threads can be regarded as two layers, one layer consists of blocks, which contain many threads and use unassociated data as input, the other layer consists of threads in blocks, which can share the data in the same block through high speed shared memory. While implementing and optimizing GPU programs, there are some important principles: (1) maximize the parallelism; (2) minimize the global data access and data copy between CPU memory and GPU memory; (3) use high-speed registers and shared memory as much as possible instead of low-speed local memory and global memory which locate in GPU memory. In a ward, make full use of advantages of GPU computing, while reducing the cost of data correspondence [10].

For each subsection blew, we first introduce the most natural and basic parallel implementation on GPU (basic GPU implementation for short in the rest of this paper), and then bring forward the optimization of it (GPU optimization for short in the rest of this paper). For basic GPU implementation, the data partition is natural but its performance is not high enough. GPU optimization takes full advantage of the GPU memory hierarchy, characteristics of communication among threads, and takes into account the overhead of the kernel boot, so it greatly enhances the acceleration performance.



### 3.1 Computing Dark Channel

The patch for computing dark channel value of one pixel is shown in fig 2. To compute dark channel value of the central pixel, whose color is dark in the figure, needs such a patch containing neighbor pixels. There are three layers in the fig 1(a) and each of layer stands for a color channel of R, G, B. We select the smallest value in the patch as the dark channel of the central pixel, and we can achieve a dark channel value for every pixel in input image in this way.

In basic GPU implementation, the number of parallel threads is the same with the number of pixels in input image, that is to say, each thread computes dark channel value for one pixel. Each thread needs 675 ( $3 \times 15 \times 15$ ) load operations from global memory, which is very slow, so each thread has to spend much time on memory access.

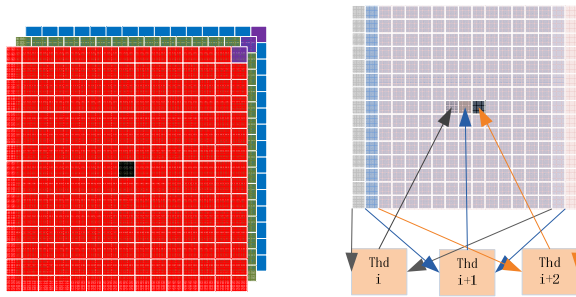


Fig. 2. (a) Patch for computing dark channel (b) Tasks mapping

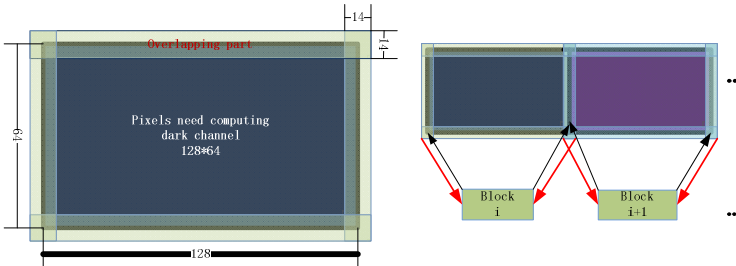


Fig. 3. (a) data block (b) data partition and task mapping

In GPU optimization, the whole data in input image was divided into blocks with size  $128 \times 64$ , and each data block is loaded into shared memory of one thread block, then each thread access to the data in shared memory of its corresponding block, rather than global memory. Shared memory access is very high, so the overheads of memory access are greatly reduced. Since data cannot transfer between shared memories of two different blocks, all data used by threads in the same thread block should be loaded into corresponding shared memory, and the data in the edge region of data block will be loaded by several thread blocks. The main part of data blocks are loaded once, and the fringe data are loaded two or four times. The details are shown in fig 3. The fig 3(a) is a data block, and fig 3(b) shows how to load overlapping data.

Each thread in thread block computes a dark channel value, but for each thread, its needed data patch are loaded from shared memory rather than from global memory.

### 3.2 Selecting Atmospheric Light

Assuming the size of input image is 600x400, the number of 0.1% brightest pixels in the dark channel image is about 240. There are two ways to implement the serial program: (1) the basic way is to search and then remove the maximum dark channel value at each time, we can obtain the 240 brightest pixels by repeating 240 times; (2) the better way is to keep an array of 240 elements, which stores the 240 examined pixels with brightest dark channel in the descent order. One pixel not examined is firstly compared with the last value in the array. If the dark channel of this pixel is smaller than the last pixel, it is abandoned, otherwise the pixel is inserted into the array and the last pixel in the original array is removed. The second method is used in our serial CPU program because it takes shorter running time, but it cannot be used in GPU program because it needs global data transfer. The first way can be used for GPU program, but its running time on GPU is the same as the second way on CPU, so the original algorithm of selecting atmospheric light cannot be speeded up on GPU.

In this paper, we propose a novel algorithm to select atmospheric light, which is of less computation and suitable for parallel. In the original algorithm, we can find out that essence of selecting atmospheric light is to take into account the dark channel values and intensities of pixels in the input image, and to select one pixel whose dark channel and intensity are both big enough. Compared to intensity of one pixel, the dark channel value is more important. Dark channel is the smallest one of the three color channels, so intuitive that the intensity is greater than the value of dark channel for the same pixel. Here we propose a new parameter of comprehensive assessment, called F, to present the related factors and the weights relationship while selecting atmospheric light. The atmospheric light is the maximum in all of the Fs. The way to compute F is shown as equation (7) (8):

$$F = a * Y + b * D . \quad (7)$$

$$a = \frac{1}{k+1} , \quad b = \frac{k}{k+1} , \quad k = \frac{Y}{D} . \quad (8)$$

Where F is the parameter of comprehensive assessment, Y is the intensity of a pixel, D is dark channel. a and b is weights for Y and D. It is obvious that a is smaller than b, so the weight of pixel intensity is smaller than that of dark channel.

The parameter of comprehensive assessment can show the essence of selecting atmospheric in original algorithm. Reasons are: (1) when D is too small, the weight of D is big, and F is very small, that is to say, too small D leads to very small F, which makes the pixel abandoned. (2) when D is big enough, for example, more than 200, then the k is close to 1, therefore, both a and b are almost equal to 1/2, that is to say, when the dark channels of pixels are all big enough, the pixels with lower intensity is abandoned. The running time of our new algorithm on CPU is about 1/10 of original algorithm.

In basic GPU implementation, the number of parallel threads is equal to the number of pixels in input image, and a parallelized method of reduction tree can be

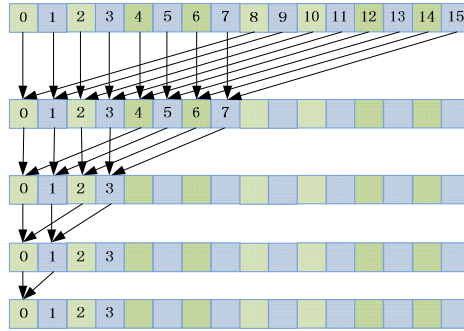


Fig. 4. The method of reduction tree

used to compute the atmospheric light. The task mapping and the method of reduction tree are shown in the fig 4.

In GPU optimization, we use thread blocks to reduce the running time because many intermediate results can be store in shared memory to reduce the overheads of global memory access. The number of thread blocks is equal to the height of input image. Each thread in the block computes the corresponding pixel’s F, and each block computes the max F in corresponding row by using parallel method of reduction tree. After that, the pixel with maximum F can be picked up as atmospheric light, and the method to find it is reduction tree, too.

### 3.3 Calculating Initial Transmission

Calculating initial transmission is similar to computing dark channel, and it concludes three steps: (1) normalize the value of each pixel channel in input image by dividing the corresponding channel of atmospheric light; (2) compute normalized dark channel value by using the same method of computing dark channel; (3) obtain the initial transmission of each pixel.

In basic GPU implementation, the number of parallel threads is equal to the number of pixels in input image. It needs three kernels to finish calculating initial transmission and each kernel accomplishes one corresponding step.

In GPU optimization, we merge the three kernels into one, which is shown in fig 5.

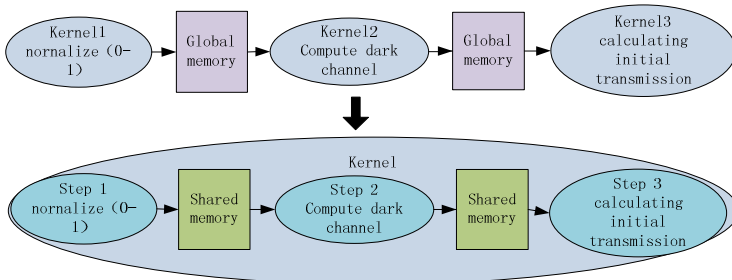


Fig. 5. Process of merging kernel

### 3.4 Refining Initial Transmission

In this step, an important function, named `box_filter`, is called for many times and it have to be divided into two kernels in parallel program. The first kernel accomplishes accumulation in row, and the second kernel accomplishes the similar computation in column. The parallel method of the two kernels is the same. The other functions of refining initial transmission are the basic four arithmetic operations among matrix. Each kind of these operations is realized by one kernel, in which the number of parallel threads is equal to the number of pixels in the input image.

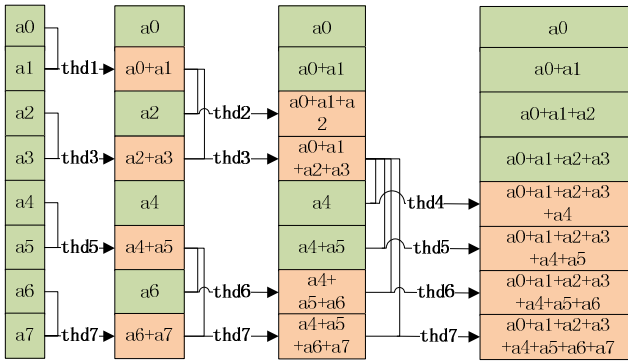


Fig. 6. Process of accumulation recording intermediate results

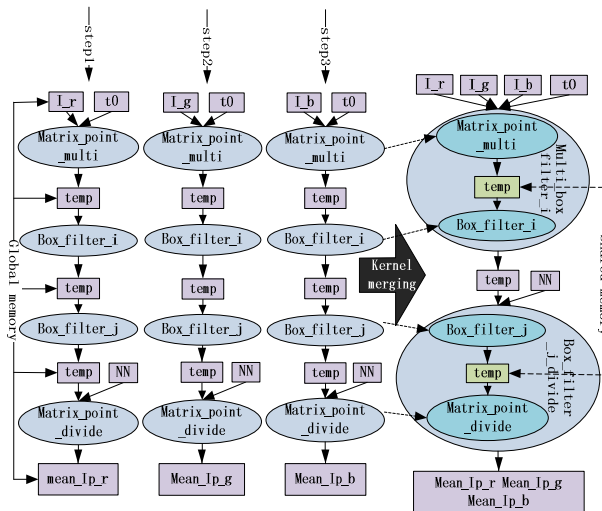


Fig. 7. Process of merging kernels in both horizontal and vertical directions

In GPU optimization, two kinds of optimization methods are used: (1) propose a novel method to compute accumulative result and to expand the parallelism inside row/column; (2) merge kernels to reduce the overheads of kernel booting.

Since not only final cumulative result but also intermediate cumulative results should be kept, the general parallel method, which only gets the final cumulative result, is not fitting. In this paper, we propose a novel method to calculate accumulative results recoding intermediate results, and the detailed process is shown in fig 6. There are many kernels in basic parallel implementation, which lead to heavy overhead of kernel booting and memory access, so we merge kernels in horizontal and vertical directions. In horizontal direction, we merge similar kernels with different input data. In vertical direction, when the output data of one kernel is the input data of another, we merge them. We provide a simple example in figure 6 to show the merging process.

Since the step 5 of haze removal is just a simple equation with little computation, we merge it into the last kernel of refining initial transmission.

## 4 Experimental Results

Environments related to experiment results are show as follow. CPU: AMD Athlon 64 X2 5200+, 2.70GHZ, 2.00GB; GPU: Geforce GTX 460; Operating system: windows XP; C++ development environment: Microsoft Visual Studio 2008; GPU Development environment: cuda IDE 4.2; Program runtime vision: release.

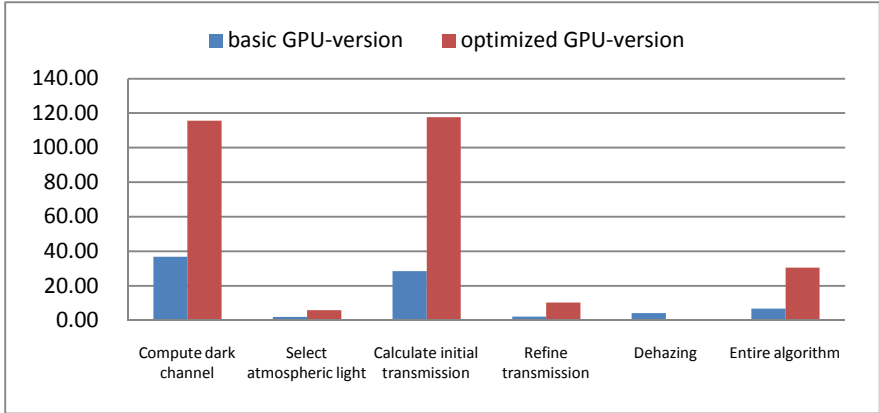
We process a 1024x768 image to evaluate the running time and speed-up of each step and entire algorithm. The running time is shown in table 2. The speed-ups of basic GPU program and optimized GPU program to CPU are shown in fig 8.

**Table 2.** Execution time of each step and entire algorithm

| step \ version                 | CPU(ms) | basic GPU-version (ms) | Optimized GPU-version (ms) |
|--------------------------------|---------|------------------------|----------------------------|
| Compute dark channel           | 1734    | 47                     | 15                         |
| Select atmospheric light       | 94      | 47                     | 16                         |
| Calculate initial transmission | 1766    | 62                     | 15                         |
| Refine initial transmission    | 1140    | 530                    | 110                        |
| Haze removal                   | 63      | 15                     | NULL                       |
| Entire algorithm               | 4797    | 701                    | 157                        |

We can find from the running time and speed-up shown above that the speed-ups are very different, the best is more than 100x but the worst is just 2-5x. The speed-ups of computing dark channel and calculating initial transmission are high because their parallelism is good and the amount of computing for each thread is relatively full. The speed-up of refining initial transmission and selecting atmospheric light is low because of low data parallelism, much data correlation, and too many kernels.

Comparing basic GPU implementation with GPU optimization, we can find that the performance of optimized GPU program substantially increase, so optimizing parallel program, according to the characteristics of the hardware and software programming, is very important to improve the performance of parallel program.



**Fig. 8.** Speed-ups of basic GPU implementation and GPU optimization

We process the images with different size and compare their running time of entire algorithm, shown in table 3.

**Table 3.** Running time of several image size

| version size | CPU(ms) | basic GPU-version (ms) | Optimized GPU-version (ms) |
|--------------|---------|------------------------|----------------------------|
| 400×300      | 734.6   | 145.3                  | 19.6                       |
| 600×400      | 1462.1  | 268.3                  | <b>36.5</b>                |
| 800×600      | 2924.5  | 442.7                  | 79.3                       |
| 1024×768     | 4812.4  | 801.6                  | 156.9                      |

From the table 3, we can find the execution time of image with size of 600x400 is less than 0.04s, that is to say, the GPU optimization program can process the image with resolution of 600x400 in real time, 25 frames per second.

In paper[2], the author also modifies original HRUDCP to decrease the computational complexity, and accelerates the modified HRUDCP on GPU, which obtains 2-3x speed-up compared to serial program of his modified HRUDCP, finally he can process foggy image with resolution 600x400 at speed of 10-12 frames per second. In paper [2], the author introduces cross-lateral filter to refine initial transmission, and the computational complexity is reduced largely, but the haze removal effect is downgraded as cost, it needs people's help to obtain as good effect as original HRUDCP. In our paper, we introduce guide filter to refine initial transmission, its computational complexity is higher than that in paper [2], but our

improved method of HRUDCP can obtain as good effect as original HRUDCP without people's assistance, so our method is more appropriate than that in paper [2], and we can process foggy image with resolution  $600 \times 400$  at the speed of 25 frames per second based on GPU, which is 2 times faster than paper [2].

## 5 Conclusion

In this article, we introduce guided image filter to haze removal using dark channel prior, and present its parallel implementation and optimization. The execution time of haze removal reduces largely, and it only takes 0.2s to process a 3M image and can process image with size  $600 \times 400$  in real time (25f/s). The further work is to improve the video dehazing process by using the relativity among frames so as to accelerate the speed of processing video data and to process video with higher resolution in real time.

## References

- [1] Narasimhan, S.G., Nayar, S.K.: Contrast restoration of weather degraded images. *IEEE Trans. Pattern Anal. Mach. Intell.* 25(6), 713–724 (2003)
- [2] Lv, X.Y., Chen, W.B., Shen, I.F.: Real-Time Dehazing for Image and Video. In: 18th Pacific Conference on Computer Graphics and Applications, pp. 62–69. IEEE Computer Society, Piscataway (2010)
- [3] Li, Y.Z.: Uneven Cloud and Fog Removing for Satellite Remote Sensing Image. In: 2011 2nd International Conference on Mechanic Automation and Control Engineering, pp. 5485–5488. IEEE Computer Society, Piscataway (2011)
- [4] Liu, Q.L., Zhang, H.Y., Lin, M.S., Wu, Y.D.: Research on Image Dehazing Algorithms based on Physical Model. In: 2011 International Conference on Multimedia Technology, pp. 467–470. IEEE Computer Society, Piscataway (2011)
- [5] Xie, B., Guo, F., Cai, Z.X.: Improved Single Image Dehazing Using Dark Channel Prior and Multi-Scale Retinex. In: 2010 International Conference on Intelligent System Design and Engineering Application (ISDEA), pp. 848–851. IEEE Computer Society, Piscataway (2010)
- [6] He, K.M., Sun, J., Tang, X.O.: Single image haze removal using dark channel prior. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1956–1963. IEEE Press, IEEE Computer Society, Piscataway (2009)
- [7] Levin, A., Lischinski, D., Weiss, Y.: A closed form solution to natural image matting. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 61–68. Institute of Electrical and Electronics Engineers Computer Society (2006)
- [8] He, K., Sun, J., Tang, X.: Guided Image Filtering. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part I. LNCS, vol. 6311, pp. 1–14. Springer, Heidelberg (2010)
- [9] NVIDIA. NVIDIA CUDA Compute Unified Device Architecture-Programming Guide Version 2.0. (2008)
- [10] NVIDIA. NVIDIA CUDATM Developer Guide for NVIDIA Optimus Platforms Version 1.0 (2010)

# Research on the Solution of Heat Exchanger Network MINLP Problems Based on GPU

Mingxing Xia<sup>1</sup>, Yuxing Ren<sup>1</sup>, Yazhe Tang<sup>1</sup>, Lixia Kang<sup>2</sup>, and Yongzhong Liu<sup>2,3</sup>

<sup>1</sup> Dept of Computer Science and Technology, Xi'an Jiaotong University,  
Xi'an 710049, Shaanxi, China

<sup>2</sup> Dept of Chemical Engineering, Xi'an Jiaotong University, Xi'an 710049, Shaanxi,  
China

<sup>3</sup> Key Laboratory of Thermo-Fluid Science and Engineering, Ministry of Education,  
Xi'an 710049, Shaanxi, China

**Abstract.** The optimization of heat exchanger network can be expressed in a Mixed Integer Non-Linear mathematical Programming (MINLP) model. However, it demands huge computing power to solve a realistic heat exchanger network optimize problem. Nowadays graphic processing unit (GPU) can be very powerful for general purpose computation. Based on the CUDA framework, this paper presents a parallel computing framework for solving the MINLP problem. We concentrate on both parallel computing model and specific GPU programming level optimization. Tests on a simple MINLP problem is conducted and the results show the new solution has 40 times faster than the one running serially on CPU.

**Keywords:** GPU, CUDA, MINLP, heat exchanger network.

## 1 Introduction

With the growing requirements of computing power for solving problems in commerce and industry area, the graphics processing unit (GPU), which has high throughput capacity and computing power, was introduced to the general-purpose parallel computing field. Compared to CPU, single GPU has more computing cores, greater throughput capacity and hardware thread switching mechanism, which makes it be more suitable to computing-intensive and throughput-intensive parallel computing process [1]. In recent years, a new parallel computing hardware architecture called CUDA (compute unified device architecture) was launched. It not only makes more people easily to do parallel programming for high-density and high-throughput applications running on GPU, but also greatly reduces the corresponding cost [4].

The heat exchanger network is one of the key subsystems in chemical, oil refining and other industries. With the goal of efficiently using energy, engineers carefully design the topology of the heat exchanger network and calculate the size of the exchanging area. Consequently, the heat exchange between the cold streams and the hot streams will realize efficient energy use and even largely determine the total cost of the system's annual expenditures. This problem can



be modeled as a mixed-integer nonlinear programming problem (MINLP) [9]. Using a deterministic algorithm can get a feasible optimal structure of the heat exchanger as well as optimal heat exchanger area. However, slightly increase in hot and cold streams number and heat exchanger series will lead to sharp increase of the solution space and hence an exponential growth of execution time. This paper presents a parallel method for solving heat exchanger network MINLP model based on GPU. In brief, we determine how to distribute different computing tasks to CPU and GPU, and we propose a strategy for efficiently making use of the massive cores of GPU. We also parallelize specific algorithms to make them running faster in GPU. The goal is to improve the computational efficiency and speed up the calculating process.

The paper is organized as follows: Section 2 introduces the MINLP model for the heat exchanger network. Section 3 gives the parallel framework for solving heat exchanger network MINLP problem based on CPU and GPU and specifically discusses the strategy to make full use of the massive GPU cores for parallel computation. Section 4 analyzes the performance of our solution by a simple comparison. Finally, the conclusion and outlook is given.

## 2 Heat Exchanger Network Problems and Its Mathematical Model

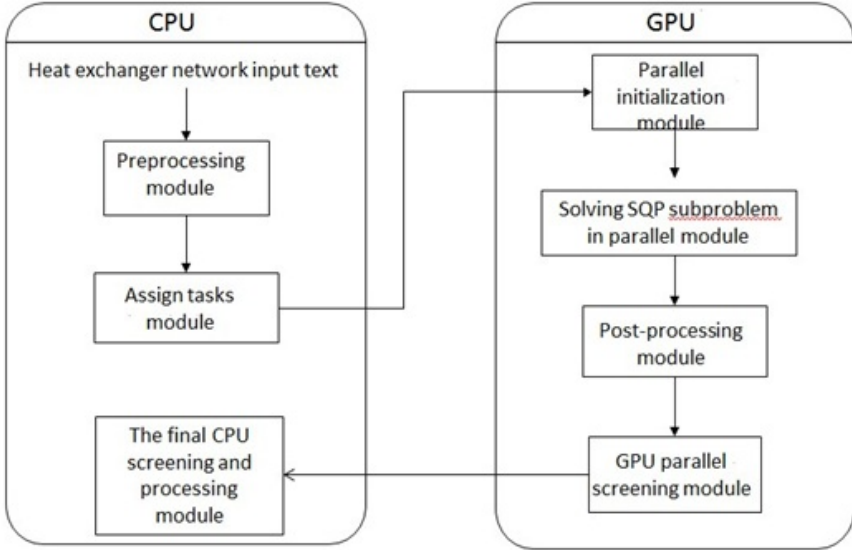
A specific heat exchanger network has  $N_h$  hot streams and  $N_c$  cold streams. The goal is to make these hot and cold streams to change to the target temperature via heat exchanger network. The optimal solution of heat exchanger network problem is actually a heat exchange between the hot and cold streams, making each close to the target temperature and having minimal cost.

The optimal solution of heat exchanger network problem can be divided into two aspects. The first is the structure of the connection between the hot and cold streams. In fact, this connection structure is the most important factor in the heat exchanger network optimization, since the connection structure will directly determine the whole process of the heat exchange. Usually the connection structure of heat exchanger network is modeled as combination of integers 1 and 0, with 1 representing there is a heat exchanger between a pair of hot and cold streams and with 0 representing no. Second, with a specific heat exchanger connecting structure, we need to determine the best heat exchangers' area to achieve minimum cost. This turns out to be a problem of nonlinear programming minimum.

## 3 Design and Implementation of Parallel Computing System

### 3.1 System Framework

Figure 1 shows the system framework for solving MINLP problems based on CPU and GPU. It can be seen that CPU carries out the pre-processing of the



**Fig. 1.** The framework of the system to solve MINLP problem based on CPU and GPU

input model and the main control of the computation, while GPU enumerates all possible heat exchanger network structures and calculates the corresponding optimal solution for each one in parallel.

### 3.2 Pre-processing of the Heat Exchanger Network Input

When solving the optimal solution of heat exchanger area and the optimal structure of heat exchanger network in chemical industry, researchers usually do not select the SQP method, a kind of deterministic algorithm. The reason is that the initial input data for the algorithm is too much and it is very likely to make mistakes when people input those data. The so-called data include objective function, every equality and inequality constraints function of the algorithm, differential evaluated expression of the objective function for continuous variables and Jacobi matrix [12] of equality and inequality constraints for continuous variables. When there are a large number of continuous variables, equality constraints and inequality constraint, the workload of manual derivation and manual calculation and entering of the Jacobi matrix will be unacceptable. To solve this problem, we design and implement the pre-processing part of the heat exchanger network input. The function of the pre-processing part is to automatically generate almost all the input data (such as the differentiation results and Jacobi matrix) according to the objective function, equality and inequality constraints. We will not talk about the details of this pre-processing due to space

limitation. The result is so exciting that we are able to get nearly all the data just from several simple expressions, as shown in Fig. 2 (a).

|   |  |
|---|--|
| $\min f(x) = -\pi x_1^2 x_2$                      | $1 \ 2, 1, 2, 0$   |
| $\text{s.t. } \pi x_1 x_2 + \pi x_1^2 - 150 = 0,$ | $2 \ f = -3.1415 * x_0^2 * x_1$                              |
| $x_1 \geq 0, x_2 \geq 0.$                         | $3 \ \text{eq}0 = 3.1415 * x_0 * x_1 + 3.1415 * x_0^2 - 150$ |
| (a)   | (b)  |

**Fig. 2.** The representation and conversion of heat exchanger network model (a) A simple non-linear programming model.(b) The string representation of model in (a)

### 3.3 The Parallel System Framework and Tasks Division for CPU and GPU

Our design uses multiple GPUs for parallel computing because there might be so many hot and cold streams in the heat exchanger network that the solving space of the MINLP problem will increase dramatically in the manner of exponential growth. Although it is said that the integer variable number of heat exchanger in every pair of hot and cold streams is close to linear growth, the number of binary combinations representing all possible connection structures is a power of 2, the number of binary variable. Computation of slightly larger model will outride the parallel computing capability of one single GPU. Things are quite different when using multi-GPUs. According to the number of GPUs and the computing capability of each GPU, CPU reasonably assigns the computation task of a certain number of binary combinations to different GPUs. The GPUs will calculate the value of the cost function for each binary combination and finally the minimum cost and the corresponding binary combination are the optimal solution.

We use block as the coarse-grained minimum parallel unit in the CUDA framework. Each block is employed to compute a unique combination of integer variables, which physically represents a specific heat exchanger network connection structure. After the heat exchanger connected structure is submitted as input to each block, the block calculates the optimal size of the heat exchanger area and ultimately get the minimum value of the total cost function in the specified connection structure. If we have enough blocks that match the number of binary combinations, we are able to compute all the binary combinations in parallel. When all blocks have completed their work, their output, namely target cost, are compared and the minimum value and the corresponding connection structure are the final answer. Obviously such a coarse-grained parallel computing framework works because the problem can be naturally divided into sub-problems that can be computed concurrently. More importantly, those sub-problems are completely independent with each other, which makes it possible to do a large

number of parallel calculations in GPU. Therefore, in this paper, the core idea of solving MINLP problems in parallel is doing coarse-grained parallel computing based on heat exchanger network connection firstly and running optimization algorithm within each block secondly, through the elaboration of multi-threading, use of registers and shared memory.

### 3.4 Solving Nonlinear Programming within a Block

After each block gets a unique combination of integer variables, which determine one unique connection structure of a heat exchanger, the next step is to solve the NLP (Nonlinear Programming) problems. To that end, this paper uses the SQP (Sequential Quadratic Programming method) [11] algorithm. The SQP algorithm uses orthogonal matrix instead of the Lagrange array Hessian matrix. These algorithms and their code are quite complex and we are not going to dig them in deep in this paper. However, we do have quite a lot considerations for speeding up the computation based on the characteristics of GPU such as setting a reasonable number of threads within a block and making efficient use of shared memory.

Specifically, we mainly consider the following factors:

- 1) Optimization on matrix-matrix and matrix-vector multiplication. This is common optimization for GPU computing.
- 2) The Jacobi matrix of equality constraints, the Jacobi matrix of inequality constraints and the orthogonal matrix are all used multiple times in the improved SQP algorithm. We take the strategy of calculating and saving them in array in advance. Once needed, the array pointers are passed to kernel function to reduce the computation. In addition, we design a faster matrix transpose algorithm. Simple test shows, using the new algorithm, the speed of transposing a matrix with 108 columns and 128 rows is 10 times faster than the CPU algorithm and 1.5-3 times faster than ordinary parallel matrix transpose algorithm on GPU.
- 3) According to the optimization strategy of the GPU memory access, we make local improvement for SQP algorithm. For instance, we do a matrix transposition before multiplication between matrix and vector, matrix and matrix, so that we can ensure that multi-threads access data in same block of memory and hence increase the degree of parallelism.
- 4) Optimization for storage on the GPU. GPUs have different type of memory (such as global memory, shared memory) and those memory has various access time from 24, 36 clocks to 400 clocks. Therefore, it would be better if we make full use of fast memory like shared memory. In this paper, we put vectors that are used very frequently into shared memory as long as those vectors do not exceed the memory size. We also set key variables accessed with high frequency into registers. At same time, we put different intermediate variables like temporary array into same space of shared memory in a time-division manner.
- 5) A large number of single-precision floating-point numbers are used in the algorithm, not only because we do not need double precision numbers for higher

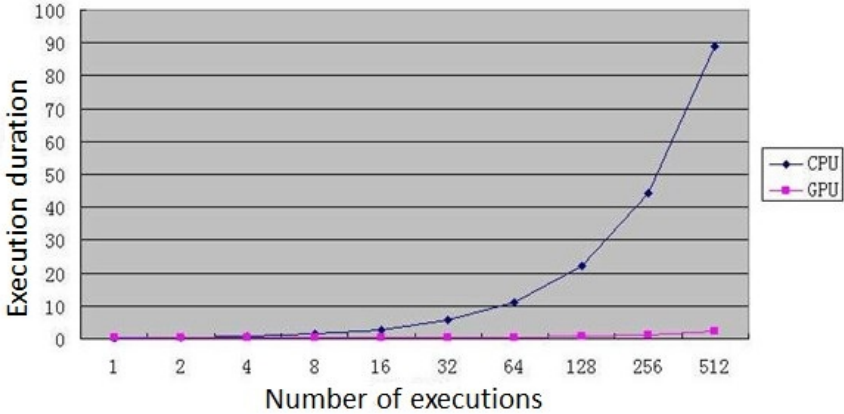
precision but also because the calculation of single precision is twice faster than that of double precision on our GPU. The paper also proposes a strategy to reduce the rounds of iterations in the SQP algorithm, aiming to sacrifice accuracy in exchange for time. The idea comes from the fact that if the changing rate of independent variables and dependent variable are both less than a reasonable threshold, we can suppose there is no need to do more iteration.

### 3.5 Experiment and Results Analysis

This section will analyze and compare the performance of CPU-based serial algorithm and GPU-based parallel algorithm on solving MINLP problems. The GPU used in the experiment is NVIDIA TESLA C2050 GPU, having 448 CUDA Cores and a top-level memory of 2687MB. There are two CPU chips of Intel (R) Xeon (R) E5640 in the server, with clock being at 2.67GHz and a total of four processing cores. The CUDA programming environment version number is 4.0.

The test case is the non-linear planning example in Figure 2 (a). For simplicity, we use only one integer combination of variables as the input of the algorithm and run the algorithm for many times. We believe this will have quite the same performance as if we run the algorithm with different integer combination of variables. We run the non-linear programming algorithm for the same times in CPU and GPU, respectively.

Figure 3 shows the experiment results which can be divided into three stages. The first stage happens when the computation only runs once or twice. In this stage, the execution time of the CPU is less than that of GPU. There may be the following reasons: 1) the code of SQP algorithm is written in C and contains a large number of branch judgment, as we all know, these branches can only execute in serial mode in GPU; 2) the SQP algorithm is not fine-grained parallelized; 3) SQP algorithm requires a large number of memory accesses, however the GPU top-level memory access cycle is very long. Shared memory can be accessed much faster, but only a small part of data can be put in it due to size limitation; 4) the CPU used in the test is a multi-core CPU so that the computing power is also very strong. The second stage starts when the computation runs more and more times, the CUDA coarse-grained parallelism in GPU begins to take effect. Each stream multiprocessor can run up to 8 blocks, that is, simultaneously run the calculation of the optimal solution of eight independent nonlinear programming models. When the number of independently running of nonlinear programming model further increases to more than the maximum number of blocks running in the GPU stream multi-processors, the GPU running time begins to grow linearly just like the running time on CPU. We can see from the experiment that the parallel algorithm designed in this paper running on GPU can be 40 times faster than the one running on CPU when we execute the algorithm to solve a large number of mutually exclusive problems of nonlinear programming.



**Fig. 3.** The simulation performance for the algorithm based on CPU and algorithm based on GPU

## 4 Conclusion

This paper analyzes and designs the GPU-based parallel algorithm for solving the heat exchanger network MINLP. Experiment shows that we can achieve acceleration of about 40 times compared with CPU algorithm implemented on the same machine. Thus, for the mixed-integer nonlinear programming solution problem, the idea of accelerating the SQP algorithm with both coarse-grained block parallel and fine-grained parallel inside blocks is feasible.

For future work, we are planning to resolve the actual complex heat exchanger network optimization problems. The scale of actual heat exchanger network is very large. It should be tough (if not impossible) to calculate using the current GPU because GPU memory is too small to hold the data. We've got three candidate research directions: first, how to better optimize the algorithm and reduce memory usage; second, how to use more kernels and more memory (the latest GPU card can support) to complete the calculation; third, how to use GPU group, which will linearly increase speedup theoretically.

**Acknowledgments.** The project has been supported by the Central Universities Fundamental Research. At the same time, we express thanks to those who give support and advice to the work of this paper.

## References

1. Cohen, J., Garland, M.: Novel Architectures: Solving Computational Problems with GPU Computing. *Computing in Science & Engineering* 11(5), 58–63 (2009)
2. Owens, J.D., Houston, M., Luebke, D., et al.: GPU Computing. *Proceedings of the IEEE* 96(5), 879–899 (2008)

3. van der Laan, J.W., Jalba, A.C., Roerdink, J.B.T.M.: Accelerating Wavelet Lifting on Graphics Hardware Using CUDA. *IEEE Transactions on Parallel and Distributed Systems* 22(1), 132–146 (2011)
4. NVIDIA. *CUDA C BEST PRACTICES Guide 4.1*: CA, Nvidia Corporation (2012)
5. NVIDIA. *NVIDIA CUDA Programming Guide 4.2*: CA, Nvidia Corporation (2012)
6. Wah, B.W., Chen, Y.: Solving Large-Scale Nonlinear Programming Problems by Constraint Partitioning. In: van Beek, P. (ed.) *CP 2005*. LNCS, vol. 3709, pp. 697–711. Springer, Heidelberg (2005)
7. Hartwich, A., Stockmann, K., Terboven, C., Feuerriegel, S., Marquardt, W.: Parallel sensitivity analysis for efficient large-scale dynamic optimization. *Optimization and Engineering* 12, 489–508 (2011)
8. Bjorkqvist, J., Westerlund, T.: Parallel solution of disjunctive MINLP problems. *Chemical Engineering Communications* 185(1), 115–124 (2011)
9. Munawar, A., Wahib, M., Munetomo, M., Akama, K.: Advanced Genetic Algorithm to solve MINLP problems over GPU. In: *IEEE Congress on Evolutionary Computation (CEC)* (2010)
10. Bjo, K., Westerlund, T.: Global optimization of heat exchanger network synthesis problems with and without the isothermal mixing assumption. *Computers & Chemical Engineering* 26(11), 1581–1593 (2002)
11. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering* 3(3), 227–252 (2002)
12. Ma, C.: *Optimization methods and matlab program design*. Science Press (2009)

# MapReduce-Based Parallel Algorithm for Detecting and Resolving of Firewall Policy Conflict

Qi Xiao, Yunchuan Qin, and Kenli Li

School of Information Science and Engineering, Hunan University,  
Changsha, 410082, Hunan, China

**Abstract.** With the complexity of networks grows so accurate in the decade, the number of firewalls rules becoming so large that the conflicts between the rules are difficult to avoid. The detection and resolution of the policy conflict become an important aspect of network security. Traditional single threaded methods are time-consumption for current large complex networks. This paper proposes a parallel algorithm of firewall policy conflict detection and resolution based on segmentation strategies. The experiments show that the algorithm is much faster than single threaded methods and is suitable for large complex networks.

**Keywords:** firewall rules, confliction, segment, MapReduce, ordering.

## 1 Introduction

In the field of computer network, the security issues are gaining more and more attention in research and industry. Firewall is the equipment that helps to ensure the security of the network, and it allows or rejects the data packets into or out of the network according to the special rules. In a network, especially for large networks, the number of rules is great, and there may be some conflicts among them, that is, when a network packet arrived, it matches at least two rules, and the matching rules may have homologous and deference actions (“Allow” and “Deny”) on the conflict. In order to guarantee the security of the network, the management of these firewall rules becomes more and more important. In a large network, there will be thousands of policy rules or more, they may be written at different times by different administrators [1]. In such situation, the management of the rules is a huge challenge for the administrators. The large number of the rules led to increasing possibility of conflicts, and made the network security facing a great threat.

At present, the research about the conflicts among firewall rules has gained great achievements. Researchers have proposed many tools for firewall analysis, some are more prominent among them, such as Firewall Policy Advisor[1] proposed by E.Al-Shaer and FIREMAN[2] proposed by L.Yuan, all of them can be used for the policy anomaly detection and analysis. The Firewall Policy Advisor only was able to detect whether there is an abnormality between conflicting rules; the FIREMAN tool could detect abnormality among multiple rules by analyzing the relationship between a rule and the package space composed by all of the rules before it. But the FIREMAN has



some limitations[3] in detecting the abnormalities, that is, the analysis results by the FIREMAN tool were only able to represent the abnormal between a certain rule and the rules before it, but cannot accurately determine which the abnormal rule is.

After the conflicts among Firewall policy rules being detected, the ultimate goal is to resolve them to ensure the security of the network. So far the momentum for the research about firewall policy conflict detecting and resolving is very rapid. A.Hari et al proposed that resolving the conflicts between the rules by increasing the resolved filter[4]. Fu et al. defined the high-level security requirements, and developed a mechanism to detect and resolve the conflicts between the IPSec rules[5]. And Golnabi et al. used the data mining techniques to resolve the conflicts of rules[6].

In addition, Hongxin et al. implemented a visualization-based firewall anomaly management environment (FAME)[7], the framework of this tool basis on the rule-based segmentation strategies, and the conflict resolution basis on the risk assessment of protected networks and the intention of policy definition, so that an average 92 percent of conflicts could be resolved. However, this tool uses a reordering algorithm that the time complexity is  $O(n^2)$ , when the number of the rules in the firewall policy becomes large, the whole process of the resolution could be time-consuming. This paper proposes a new solution that the segments obtained by the rule-based segmentation strategy are sorted by a simply dedicated algorithm, and the segments after sorting would be converted into the expression of a regular rule by an inverse function of the rule-based segmentation strategies, replacing the original rules. As the segments obtained by the rule-based segmentation strategies are pairwise disjoint, and each segment associates with only one action, which makes the conflicts no longer exist. The core of this solution is a sorting algorithm which can be parallel implemented. The basic idea of this paper is using the SecondarySort algorithm of the MapReduce model to sort all the segments. The algorithm is time efficient, and ensuring the firewall policy conflicts' resolution. When the packets arrived, the firewall rules dealt with our algorithm would decide to "allow" or "deny" the packets quickly and effectively, ensuring the security of the network.

## 2 Overview of Firewall Policy Anomalies

A firewall rule is defined by a set of conditions, when a packet going across the firewall matched all the conditions defined by a rule, the predetermined action of the rule would be performed. The set of conditions include the following five elements[8]: protocol type, source IP, source port, destination IP, destination port, these conditions determine whether to allow or deny the packets through the network[9][10]. Table 1 is an example of a firewall policy, lists 5 rules of the policy, i.e.  $r_1$ ,  $r_2$ ,  $r_3$ ,  $r_4$  and  $r_5$ . Each condition of a rule can be defined as a value or a range of values. Such as, the protocol type of  $r_5$  is "\*", that indicates the value of the protocol type is TCP or UDP, and source IP address "10.1.1.\*" means that the IP address is in the range of 10.1.1.1 to 10.1.1.255.

**Table 1.** Example of a firewall policy

| Rule | Protocol | Source IP | Source Port | Destination IP | Destination on Port | Action |
|------|----------|-----------|-------------|----------------|---------------------|--------|
| r1   | TCP      | 10.1.2.*  | *           | 192.168.1.*    | 25                  | deny   |
| r2   | TCP      | 10.1.*.*  | *           | 192.168.*.*    | 25                  | deny   |
| r3   | UDP      | 10.1.*.*  | *           | 172.32.1.*     | 53                  | allow  |
| r4   | UDP      | 10.1.1.*  | *           | 172.32.1.*     | 53                  | Deny   |
| r5   | *        | 10.1.1.*  | *           | *              | *                   | allow  |

The anomalies between firewall policy rules normally include four categories[11]: Shadowing, Generalization, Correlation, and Redundancy. A rule is shadowed when a previous rule matches all the packets that match this rule, such that the shadowed rule will never be activated. A rule is a generalization of a preceding rule if they have different actions, and if the second rule can match all the packets that match the first rule. Two rules are correlated if they have different filtering actions, and the first rule matches some of the packets that match the second rule and the second rule matches some of the packets that match the first rule. A redundant rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected.

The anomalies above defined for the relationship between two rules, we believe that the firewall policy conflict should always consider a firewall policy as a whole piece[7]. In this paper, we consider that a precise indication of the conffliction caused by a set of overlapping rules is critical for resolving the conflicts.

**Definition 1 (Policy Conflict)**<sup>[7]</sup>. A policy conflict  $pc$  in a firewall  $F$  is associated with a unique set of conflicting firewall rules  $cr = \{r_1, \dots, r_n\}$ , which can derive a

**Algorithm 1** : Segment Generation for Network Packet Space of a Set of Rules

```

Input: a set of rules, R.
Output: a set of packet
spaces segments, S
1  foreach r ∈ R do
2    sr ← PacketSpace(r);
3    foreach s ∈ S do
4      /* sr is a subset of
s*/
5      if sr ⊂ s then
6        S.Append(s-sr);
7        s ← sr;
8        break;
9      /*sr is a superset of
s*/
10     else if sr ⊃ s then
11       sr ← sr - s;
12     /* sr partially
matches s*/
13     else if sr ∩ s ≠ ∅
then
14       S.Append(s- sr);
15       s ← sr ∩ s;
16       sr ← sr - s;
17     S.Append(sr);
18   return S;

```

*common network packet space. All packets within this space can match exactly the same set of firewall rules, where at least two rules have different actions: Allow and Deny.*

### 3 Firewall Policy Conflict Detecting and Resolving Algorithm

The algorithm for firewall policy conflict detecting and resolving mentioned in this paper is divided into four steps: Firstly, all the rules are converted to the structures of packet space segments, and the segments are pairwise disjoint. If a segment contains more than one rules, and these rules have inconsistent actions, then it indicates that a conflict occurred; Secondly, set an action constraints for each conflict segment, to make each segment an uniform action to resolve the conflict; Thirdly, all these segments should be sorted by a MapReduce-based algorithm to get an orderly sequence of segments; Finally, the sequence of segments would be converted back to the representation of rules to facilitate matching of packets.

#### 3.1 Rule-Based Segmentation Strategy

In order to resolve the conflicts caused by a group of overlapping rules, Hongxin Hu et al[7] proposed a rule-based segmentation strategy, using the binary decision diagram (BDD) to represent the rules, to convert a group of rules into a set of disjoint network packet spaces. The strategy has been used in several research areas, such as network traffic measurement[12], firewall testing[13] and optimization[14]. The pseudo code of generating segments for a set of firewall rules was shown in Algorithm 1. For a set of inputted rules, each one should be converted to a packet space. There are four types of relationships between one packet space and all the other packet spaces existed: subset, superset, partial match, and disjoint. After calculating the relationships of the packet spaces by Algorithm 1, the algorithm output a set of unrelated packet space segments.

A set of segments outputted by Algorithm 1 are pairwise disjoint, and any two different network packets within the same segment are matched by the exact same set of rules.

Fig. 1 performed the process that 5 rules in Table 1 were divided into a set of pairwise disjoint packet spaces. Each packet space was denoted by a colored rectangle, and the different colors represented different kinds of spaces – allowed space (white color) and denied space (gray color). Each rule converted into a packet space, and the overlapping part of the two spaces meant that some network packets matched with two spaces simultaneously. Fig. 1 shows that rule  $r_5$  intersect with rule  $r_2$ , rule  $r_3$  intersect with rule  $r_5$ , and  $r_3$  intersect with rule  $r_4$ , there are three overlapping packet spaces. The Algorithm 1 divided all the packet spaces into packet space segments which were pairwise disjoint, the segments were classified into the following categories: nonoverlapping segment and overlapping segment, which was further divided into conflicting overlapping segment and nonconflicting overlapping segment. In order to facilitate analysis of each segment, the segments were divided

into the form like Fig. 1(c). We could find in Fig. 1 that the 5 rules in Table 1 were divided into 7 packet space segments, and all of them are pairwise disjoint, the segment  $s_2$ ,  $s_4$  and  $s_7$  are the nonoverlapping segments, others are overlapping segments, wherein segment  $s_3$  and  $s_5$  are the conflicting overlapping segments,  $s_1$  and  $s_6$  are the nonconflicting overlapping segments.

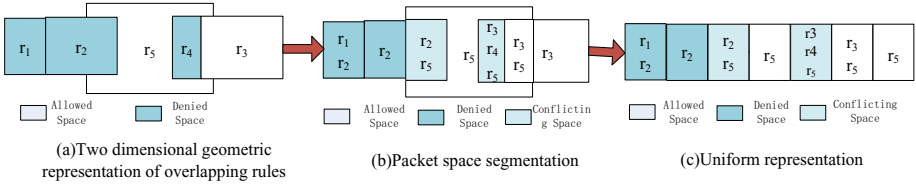


Fig. 1. Packet space representation derived from the rules in Table 1.

### 3.2 The Action Constrains of Segments

For the packet space segments converted by the Algorithm 1, if there is only one action to determine the actions of the packets in a segment, the conflicts don't exist. For the nonoverlapping segments, the action of the packets in segment is determined by the action of the rules inside. For the nonconflicting overlapping segment, the action of the packets in the segment was decided by the common action of all rules in the segment. For the conflicting overlapping segment, it involves two or more rules and the actions of the rules conflicts, so we should set the unified action constraints for this kind of segment. Hong Hu et al[7] proposed each conflicting overlapping segment should determine a unified action constraint, to make the packets involved in the conflicting overlapping segment have a unified action to resolve the conflict. The action constraint of the conflicting overlapping segment means that when the packets matching with this segment arriving, what the firewall policy should do (allow or deny)?

The formation of the action constraint should interact with the system administrators minimally, and divided into manual and automatic strategy selections. The risk detection method of the conflicting overlapping segment needs to calculate the value of a risk assessment. This value is used to determine which action the system expected to performed on the packets matching this segment. If the risk value of the segment is higher than the given value of the upper threshold (UT), the action of this segment desired is "Deny"; if the risk value of the segment is lower than the given value of the lower threshold (LT), then the action of this segment desired is "Allow"; and if the risk value of the segment is between the upper threshold and the lower threshold, the administrators should manually determine the action of this segment desired based on the characteristics of the conflict. The upper threshold and the lower threshold are determined by the administrator.

The risk value is figured out by the protected network-based vulnerability assessment[15], using the Common Vulnerability Scoring System (CVSS)[16] as the safety measure of the risk assessment. We assess the risk of the network by two main factors – exploitability of vulnerability and severity of vulnerability[17]. Another

major factor is asset importance value, for example, system administrator would set the priority of the important protective server higher than an ordinary PC, and this value represent the inherent value of a server for a network attacker or an administrator. We incorporate the CVSS base score and asset importance value to compute the risk value for each vulnerability factor as follows:

$$RiskValue = (CVSSBaseScore) \times (ImportanceValue) \quad (1)$$

In order to calculate the risk value of every conflict segment, we need to accumulate all risk values of vulnerabilities covered by a conflicting segment. For the administrators, they are more concerned about the safety value of every hole in the network. Therefore, the formula to calculate the average risk value of segment is performed as follow:

$$RL(cs) = \frac{\sum_{v \in V(cs)} (CVSS(v) \times IV(s))}{\alpha \times |V(cs)|} \quad (2)$$

Where  $V(cs)$  represents all of the vulnerabilities contained by the conflicting segments  $cs$ ;  $CVSS(v)$  indicates the reference value of the CVSS vulnerability  $v$ ;  $IV(s)$  indicates the importance value of the server  $s$ ; Variable  $\alpha$  ( $1 / |V(cs)| \leq \alpha \leq 1$ ) allows the administrators to choose the average or overall risk value to evaluate the risk of every conflict segment. When  $\alpha$  reduce, the administrators pay more attention to the overall risk value. When  $\alpha$  increases, the administrators pay more attention to the average value at risk.

### 3.3 Segments Sorting Algorithm Based on MapReduce

For each segment in packet spaces, when the packet matching it arrived, there would have a determined action to execute. Since these segments are pairwise disjoint, for each incoming packet, it matches nothing, or only matches with one segment, and the conflicts of firewall policy are resolved. However, due to the number of the segments converted by Algorithm 1 would be greater than the number of the original rules, making the matching of the packets become more complex. Especially when the number of rules is large, the number of segments may be larger than the number of rules, which would seriously affect the filtering capabilities of the firewall policy. In order to make the algorithm to be applied in actual situations, we propose the algorithm using MapReduce model<sup>[18][19][20]</sup>, adopting the cluster computers to sort all the segments, and then getting a segment sequence increasing ordered.

#### 3.3.1 MapReduce Model

MapReduce is Google's invention, mainly for dealing with a large amount of data in a distributed computing model. The model hides parallelization, fault tolerance, data distribution, load balancing and other details into a library, the user only need to concern the operations they need to perform. Its main principle is as following: using the map operation on the input data, calculating an intermediate "key / value" pairs

set, and using the reduce operation on all “value” with the same “key”, users only need to specify the Map and Reduce functions.

In a large network, when the number of rule is large, according to the difference of the relevance between the rules, the number of the segments would be several times more than the number of rules and even more than a few times, which seriously affects the matching of the arrival packets, so that the algorithm we proposed can't be used in the actual network using traditional computing model. And if we used the ordinary sorting algorithm, the large numbers of segments would make the algorithm take too much time. Therefore, the MapReduce model adopted in this paper was to prevent long computation time caused by the large amount of data which making the calculation can't be even handle.

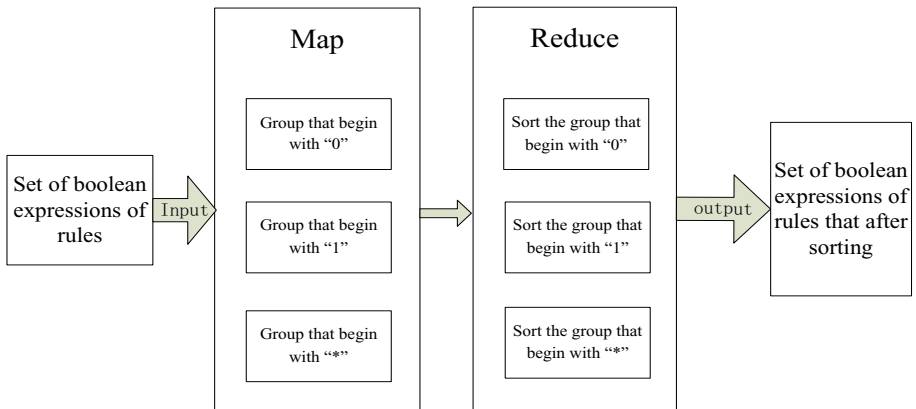
### 3.3.2 The Segments Sorting Algorithm

The section 3.1 mentioned that, all the rules are expressed as the BDD packet space format. Each rule has five conditions, expressed as a quintuple <protocol type, source IP, source port, destination IP, destination port >. BDD is used to represent the structure of the Boolean function. The packet space form of a rule expressed as BDD structure mentioned in section 3.1, is connected together by the Boolean expression representing all the conditions of a rule, to form a sequence represented by Boolean variable 0/1/\*. The packet space segments formatted by Algorithm 1 are obtained by the calculation of these sequences of rules. So for every Boolean expression of segments, it contains either the sequence of Boolean variable (0/1/\*), or its complementary set. To facilitate the sorting later, we represents these sequences as strings, the Boolean variable which can be expressed 0 and 1 expressed with “\*”. For example, the rule4 in Table 1, “UDP 10.1.2.\* \* 172.32.1.\* 53”, can be expressed as the Boolean expression like, “10000101000000000100000010\*\*\*\*\*  
\*\*\*\*\*101011000010000000000001\*\*\*\*\*00110101”. The first bit “1” indicates “UDP”. In this paper, if the first bit is “0”, it indicates “TCP”; if it is “1”, it indicates “UDP”; and if it is “\*”, it indicates the arbitrary value in “TCP” and “UDP”. The segments are formed by the Boolean expression of rules itself or its complementary set, so the Boolean expressions of segments are either similar to the Boolean expression of rules, or the collection set of this Boolean expression of rules.

For correctly sorting the segments, our algorithm splits the collection set of the Boolean expressions. For example, a simple BDD structure  $f(x_1, x_2, x_3) = x_1x_2x_3 + x_1x_2x_3$ , it can express as 110+011. The two values 110 and 011 indicate the corresponding packet segment. In order to sort all of the packet segments, our algorithm divides the BDD structure into 110 and 011, and both of them join in the queue of sorting.

In our sorting algorithm, when the number of rules is too large, the number of segments would be larger than it, so we would use a parallel algorithm based on the MapReduce model to sort the segments. Fig. 2 is a flowchart of the algorithm, for the set of inputted Boolean expressions, in the phase of the map, in accordance with the first character of the Boolean expression, all the segments would be divided into three categories, named “TCP” segment (the first character is “0”), “UDP” segment

(the first character is “1”) and the segment that the first character is “\*”, outputting a key-value pairs <segment type, segment Boolean expression>. Then, in the phase of reduce, we sort all the Boolean expressions that has the same segment type by the method of the MapReduce secondary sorting, and the larger one between two expressions would be determined by follow: starting from the second character (because all the first characters in a categories are the same), sorting the Boolean expressions according by the order of  $0 < 1 < *$ . If the second character of two expressions is the same, then gonging on the third one, and so on. Finally, in the phase of reduce output, we would sort the three categories of Boolean expressions by the order of  $0 < 1 < *$  (the first character), and all the Boolean expressions would Comprehensive been a sequential set.



**Fig. 2.** A flowchart of the our sorting algorithm

### 3.4 Segments Converted into the Representation of Rules

The segment groups after sorting are represented as strings, so it is hard to apply on matching of the packets. In order to match the packets correctly, our algorithm convert the segments back to the representation of the original rule, that convert the Boolean expressions which are represented by 97 bits back to the form of the rule which has 5 conditions and one action. The conversion algorithm is the inverse function that rules convert into the packet spaces, or the inverse function of Algorithm 1. Meanwhile, the segment’s action is the rule’s action. According to the original conversion algorithm, the first character indicates the protocol type, that we would convert “0” to “TCP”, “1” to “UDP”, and “\*” to “\*” (anyone of the two protocol types). Then according to the number of bits, we would separate other rule conditions: Source IP, Source port, destination IP, and the destination port. Particularly note that it may have some troubles when a block of bits convert to IP address, such as the block of bits are “00001010000000010000001\*\*\*\*\*”, they indicate a range of IP address and not only a IP address, the lower bound of this range is “10.1.2.0” when

“\*” are all replaced by “0”, and the upper bound of this range is “10.1.3.255” when “\*” are all replaced by “1”, that is this block of bits can convert to a range of IP address “10.1.2.\*~10.1.3.\*”.

After the conversion, the segments are all in the order of  $0 < i < *$ . When a packet comes, using the first-value matching strategy can ensure the packet matching fast and accurate. Because all the rules are in order, the special value (0 or 1) is always in the front of the arbitrary value (anyone in 0 and 1). That is, we give an example of protocol type, the rules has been arranged in the order of “TCP-UDP-any”, for all “TCP” packets coming, the firewall policy would match them with the “TCP” rules rather than the “any” rules. The matching of other conditions are the same. After the conversion, the segments are pairwise disjoint, and the rules after conversion are pairwise disjoint the same. Every rule has only one action, so the conflicts of the policy are resolved.

### 3.5 Equivalence of the Rules before and after Conversion

In order to ensure that the firewall policy which is converted by our algorithm still work normally and the function of packets filter is the same with the original firewall policy, we must guarantee the equivalence between the rules after conversion and before. The so-call equivalence between the two groups of rules means that, for any packets coming, the actions (either deny or allow) which the new firewall rules performed are the same with the original firewall rules expected. To prove the equivalence of two groups of rules, which is equivalent to prove the set of packet space is unchanged and the corresponding actions are accord with the expectations of the original rules.

Theoretically speaking, converting the rule to the format of the packet space is only a change of the expressions of the rule, and it can't change the set of packets that a firewall rule could handle. The format of packet space expresses the set of the packets which can match with all the conditions of this rule. Algorithm 1 compares and computes with these sets only. Obviously, the result of the operations within a set is the same with the collection of original set. So the result of Algorithm 1 does not change the set of the total packets, and it doesn't affect the packets which can match the firewall policy. The rules converted by Algorithm 1 represent as segments, if the actions of all rules involved in a segment are the same, then this action of all rules is the action of this segment; if they are inconsistent, we use the Common Vulnerability Scoring System (CVSS) to determine its action. The Common Vulnerability Scoring System is generally in line with the provision which the administrators expect to resolve the conflicts of the firewall policy. Then the segments are all converted to the representation of the rules. A segment indicate a packet space, the set of segments are equivalent with the original rules, so the rules after conversation are equivalent to the rules before conversation as the same.

For the sake of simplicity, we take the first two rules of Table 1 as an example to verify the equivalence between the rules converted by our algorithm and the original rules. Firstly, we would convert the rules  $r_3$  and  $r_4$  to the packet spaces, the result of conversion based on Algorithm 1 as shown in Table 2.



**Table 2.** The packet spaces of rule  $r_3$  and rule  $r_4$  in Table 1

| Segment No | The string representation of segments   |
|------------|---|
| $s_1$      | 1000010100000000100000000*****101011000<br>010000000000001*****000000000110101+100001010000000010<br>000001*****1010110000100000000001***<br>*****000000000110101+100001010000000100001*****<br>*****1010110000100000000001*****000000000110<br>101+1000010100000000100001*****10101<br>1000010000000000001*****000000000110101+10000101000000<br>0010001*****1010110000100000000000<br>1*****0000000000110101+10000101000000001001*****<br>*****1010110000100000000001*****000000000<br>0110101+1000010100000000101*****1<br>0101100001000000000001*****000000000110101+1000010100<br>0000011*****1010110000100000000001*****101011000010000000<br>00001*****000000000110101 |
| $s_2$      | 1000010100000000100000001*****101011000<br>010000000000001*****000000000110101  |

Seen from the Table 2, rule  $r_3$  and rule  $r_4$  can convert to two segments  $s_1$  and  $s_2$ , and the  $s_2$  is the collection of 8 strings of BDD format. In order not to miss any packets when filtering, we need to split the collection. The segment  $s_2$  can split into eight strings, which indicate eight ranges of packets matching. We sort these nine strings with our MapReduce-based algorithm, and get an ascending sequence, as shown in Table 3.

**Table 3.** The ascending sequence of segment  $s_1$  and segment  $s_2$  in Table 2

|   |
|---|
| 1000010100000000100000000*****101011000010000000<br>000001*****000000000110101<br>1000010100000000100000001*****101011000010000000<br>000001*****000000000110101<br>100001010000000010000001*****101011000010000000<br>000001*****000000000110101<br>10000101000000001000001*****101011000010000000<br>000001*****000000000110101<br>100001010000000010001*****101011000010000000<br>000001*****000000000110101<br>100001010000000010001*****101011000010000000<br>000001*****000000000110101<br>10000101000000001001*****101011000010000000<br>000001*****000000000110101<br>1000010100000000101*****101011000010000000<br>000001*****000000000110101<br>100001010000000011*****101011000010000000<br>000001*****000000000110101 |
|---|

After the sorting, we converted the segments back to the representation of the rules. The operation of subtraction may result in the situation that each segment expressed in a range. We take the minimum and maximum value as the upper bound

and the lower bound of this range. After the conversion, we would give the same action to the rules which split from the same segment. The result of the conversion is shown in Table 4.

**Table 4.** The nine segments in the Table 3 converted to the rules

|        |     |                       |   |            |    |       |
|--------|-----|-----------------------|---|------------|----|-------|
| $r'_1$ | UDP | 10.1.0.*              | * | 172.32.1.* | 53 | allow |
| $r'_2$ | UDP | 10.1.1.*              | * | 172.32.1.* | 53 | deny  |
| $r'_3$ | UDP | 10.1.2.*~10.1.3.*     | * | 172.32.1.* | 53 | allow |
| $r'_4$ | UDP | 10.1.4.*~10.1.7.*     | * | 172.32.1.* | 53 | allow |
| $r'_5$ | UDP | 10.1.8.*~10.1.15.*    | * | 172.32.1.* | 53 | allow |
| $r'_6$ | UDP | 10.1.16.*~10.1.31.*   | * | 172.32.1.* | 53 | allow |
| $r'_7$ | UDP | 10.1.32.*~10.1.63.*   | * | 172.32.1.* | 53 | allow |
| $r'_8$ | UDP | 10.1.64.*~10.1.127.*  | * | 172.32.1.* | 53 | allow |
| $r'_9$ | UDP | 10.1.128.*~10.1.255.* | * | 172.32.1.* | 53 | allow |

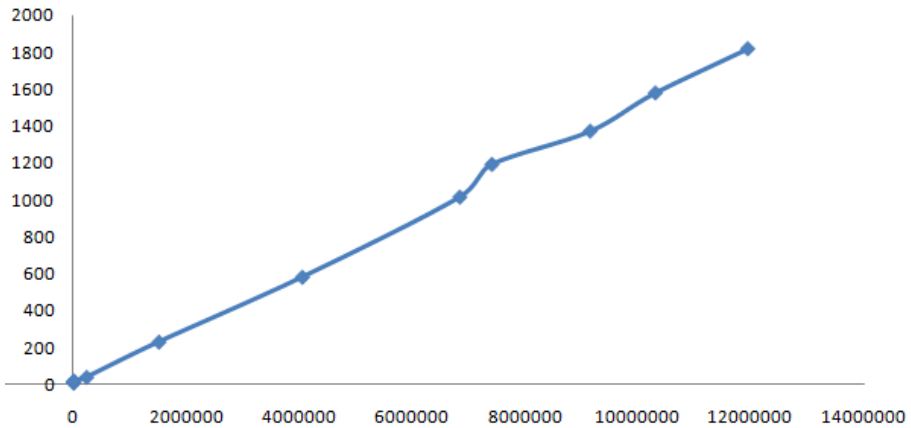
We observe the rule  $r_3$  and rule  $r_4$  of the firewall rules before conversion and know that the original rule's range to match the packets is "UDP, 10.1.\*.\*, \*, 172.32.1.\*, 53", and it is the matching packets which must be the UDP protocol and send from IP address 10.1.\*.\*, any ports, to the IP address 172.32.1.\*, port 53. For the nine rules after conversion, we connect their lower bound and the upper bound, and know that the matching packets must be the UDP protocol and is sent from IP address 10.1.1.\*~10.1.255.\*, any port, to the IP address 172.32.1.\*, port 53, and they are equivalent to the original rules. The original rule  $r_3$  and rule  $r_4$  conflict, so the packets which match the overlap of two rules cannot do the correct actions. After using our algorithm to resolve the conflict of firewall, the action of rule  $r'_2$  (equivalent to  $r_3$ ) is the same with the action of the original rule  $r_3$ , other rules' actions are the same with the difference set of  $r_4$  and  $r_3$  ( $r_4 - r_3$ ). The actions of the rules after conversion meet the expectations of the administrators.

## 4 Implementation and Evaluation

In this paper, we used the Java programming language to implement the algorithm, and the algorithm is based on the MapReduce model. Our experiments were performed on a cluster of three Intel computers, each with 8 core and 4G memory. The algorithm implemented as the steps follow: At first, the rules were converted to segments by the Algorithm 1. Then the segments were added into the MapReduce model and sorted. In the phase of map, we divided the segments according with the first bit into three categories; in the phase of reduce, we sorted the three categories of segments by our MapReduce-based algorithm, and converted the segments back to the representation of the rules, then all the data got together into an output file finally.

The rule-based segmentation strategy in our paper is similar to the strategy in the FAME<sup>[7]</sup> framework which proposed by Hongxin Hu et al. After converting the segments to the representation of strings, the strings were inputted into MapReduce model to be sorted. The MapReduce model is used to the parallel computation for

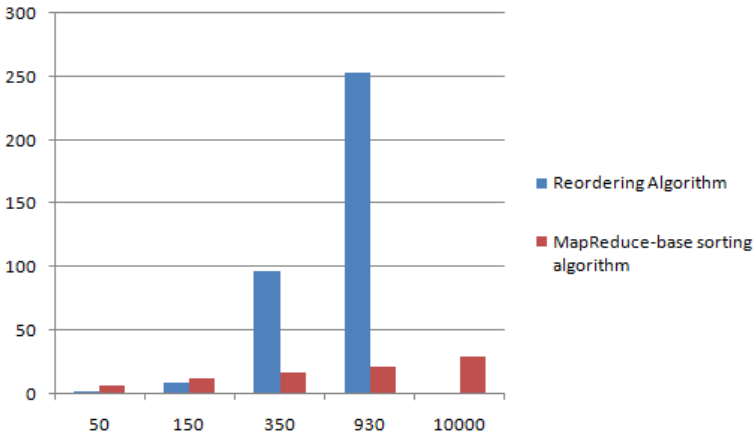
large-scale data sets, so our algorithm is mainly used for the conflicts detection and resolution of large-scale rules. When implementing we found that, the processing time of our algorithm increases along with the number of the segments. When the number of rules is small, our algorithm had great superiority in time complexity compared with other algorithms; for the number of over millions of rules, our algorithm also can handle such a large amount of data. Fig. 3 is the time consumption by our sorting algorithm for the data less than 12 million, we found that the number our algorithm could handle can achieve a super value. Seen from the Fig. 3, we found that with the increase of the number of segments, the time consumed by our sorting algorithm increases almost linearly. So our algorithm can handle tens of millions of data that others cannot handle.



**Fig. 3.** The time consumption of the algorithm in less than 12 million segments

Fig. 4 shows the comparison of two algorithms in the same scale, wherein the abscissa is the number of rules, and the ordinate is the time consumed. Seen from the figure that, when the rule number is less than 150, the reordering algorithm in the FAME framework required less time than our algorithm; when the rule number is equal to 150, the time required for sorting in two algorithm is almost the same; however, when the rule number is more than 150, the time our MapReduce-based sorting algorithm required did not change much, but the time resorting algorithm required had greatly increased; particularly, the algorithm we proposed is able to handle tens of thousands number, and the time consumption is relatively short.

In addition, the FAME framework can resolve about 92 percent of the conflicts. This is because that the reordering algorithm in the framework cannot make sure that the actions of rules after sorting fully comply with the actions of the segments they involved in. But the sorting algorithm we proposed sorted the segments directly, and made the segments' actions be the rules' actions, which guaranteed the new rules after conversion can resolve nearly 100 percent of conflicts.



**Fig. 4.** The time comparison between the reordering algorithm and the MapReduce-base sorting algorithm

## 5 Summary and Related Work

In this paper, we improved the FAME framework, changed the reordering algorithm that has high time complexity. We sorted based on the packet space segments, used a parallel sorting algorithm based on the MapReduce model to sort the segments directly, and converted the segments back to the representation of rules, so the actions of the segments are the actions of these rules. The algorithm we proposed simplifies the sorting in the original framework, and ensures the accuracy of the packets filtering by directly replacing the original rules to the rules after conversion. Therefore, it is very suitable for the use of large complex network.

Theoretically, the number of rules after conversion will be greater than the number of original rules, and it will increase the time for matching when the packets coming, the future direction of our paper is to study the merger of the rules. Some rules after conversion are sequential, so we could develop the corresponding packet matching algorithm according the characteristic of the rules, to simplify the matches of this kind of packets.

**Acknowledgments.** Thanks for the guidance and help of all the teachers and students in our laboratory for this article.

## References:

1. Al-Shaer, E., Hamed, H.: Discovery of Policy Anomalies in Distributed Firewalls. In: IEEE INFOCOM 2004, vol. 4, pp. 2605–2616 (2004)
2. Yuan, L., Chen, H., Mai, J., Chuah, C., Su, Z., Mohapatra, P., Davis, C.: Fireman: A Toolkit for Firewall Modeling and Analysis. In: Proc. IEEE Symp. Security and Privacy, p. 15 (2006)

3. Alfaro, J., Boulahia-Cuppens, N., Cuppens, F.: Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies. *Int'l J. Information Security* 7(2), 103–122 (2008)
4. Hari, A., Suri, S., Parulkar, G.M.: Detecting and resolving packet filter conflicts. In: *INFOCOM* (3), pp. 1203–1212 (March 2000)
5. Fu, Z., Wu, S.F., Huang, H., Loh, K., Gong, F., Baldine, I., Xu, C.: IPSec/VPN security policy: Correctness, conflict detection, and resolution. In: *Proceedings of Policy 2001 Workshop* (January 2001)
6. Golnabi, K., Min, R.K., Khan, L., Al-Shaer, E.: Analysis of firewall policy rules using data mining techniques. In: *IEEE/IFIP Network Operations and Management Symposium, NOMS 2006* (April 2006)
7. Hu, H., Ahn, G.J., Kulkarni, K.: Detecting and resolving firewall policy anomalies. *IEEE Transactions on Dependable and Secure Computing* 9(3), 318–331 (2012)
8. Abedin, M., Nessa, S., Khan, L., Thuraisingham, B.: Detection and resolution of anomalies in firewall policy rules. In: Damiani, E., Liu, P. (eds.) *Data and Applications Security 2006*. LNCS, vol. 4127, pp. 15–29. Springer, Heidelberg (2006)
9. Tian, D.X., Liu, X.: A fast matching algorithm and conflict detection for packet filter rules. *Journal of Computer Research and Development* 42(7), 1128–1134 (2005)
10. Hunt, R., Verwoerd, T.: Reactive firewalls-A new technique. *Computer Communications* 26(12), 1302–1317 (2003)
11. Al-Shaer, E., Hamed, H., Boutaba, R., Hasan, M.: Conflict Classification and Analysis of Distributed Firewall Policies. *IEEE Journal on Selected Areas in Communications*, vol 23, 2069–2084 (2005)
12. Yuan, L., Chuah, C., Mohapatra, P.: ProgME: Towards Programmable Network Measurement. *ACM SIGCOMM Computer Comm. Rev.* 37(4), 108 (2007)
13. El-Atawy, A., Ibrahim, K., Hamed, H., Al-Shaer, E.: Policy Segmentation for Intelligent Firewall Testing. In: *Proc. First Workshop Secure Network Protocols, NPSec 2005* (2005)
14. Misherghi, G., Yuan, L., Su, Z., Chuah, C.-N., Chen, H.: A General Framework for Benchmarking Firewall Optimization Techniques. *IEEE Trans. Network and Service Management* 5(4), 227–238 (2008)
15. Sawilla, R.E., Ou, X.: Identifying Critical Attack Assets in Dependency Attack Graphs. In: Jajodia, S., Lopez, J. (eds.) *ESORICS 2008*. LNCS, vol. 5283, pp. 18–34. Springer, Heidelberg (2008)
16. Mell, P., Scarfone, K., Romanosky, S.: *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Published by FIRST—Forum of Incident Response and Security Teams (June 2007)
17. Zhang, Y.Z., Fang, B.X., Chi, Y., Yun, X.C.: Risk propagation model for assessing network information system. *Journal of Software* 18(1), 137–145 (2007)
18. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. *Communication of the ACM-50th Anniversary Issue* 51(1), 107–113 (1958-2008)
19. Li, J.J., Cui, J., Wang, D., Yan, L., Huang, Y.S.: Survey of MapReduce parallel programming model. *Acta Electronica Sinica* 39(11), 2635–2642 (2011)
20. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. *Communication of the ACM-Amir Pnueli: Ahead of His Time* 53(1), 72–77 (2010)

# DPA-Resistant Algorithms for Trusted Computing System

Lang Li<sup>1,2</sup>, Kenli Li<sup>2</sup>, Yi Wang<sup>2</sup>, YuMing Xu<sup>1,2</sup>, Hui Liu<sup>1</sup>, and Ge Jiao<sup>1</sup>

<sup>1</sup> Department of Computer Science, Hengyang Normal University,  
Hengyang, 421002, China

<sup>2</sup> College of Information Science and Engineering, Hunan University,  
Changsha, 410082, China

lilang9111@126.com, lk1520@263.net,  
{3999103,249284257,23506686,41706014}@qq.com

**Abstract.** Side channel attacks could efficiently break cryptographic algorithm based on hardware implementation including applications on trusted computing systems. Chinese researchers had proposed a standard encryption algorithm, called SMS4, for their own wireless LAN communications in 2006. In this paper, we propose a modified fixed-value masking algorithm for SMS4 in order to resist again power analysis attack to hardware based SMS4. Furthermore, we simulate the attacking environments and port the proposed countermeasure to FPGA platform. The experimental results show that the proposed countermeasure can efficiently resist against power analysis attack.

**Keywords:** Differential Power Analysis, SMS4, Fixed-Value Masking Algorithm, Trusted Computing System

## 1 Introduction

The growing demands for information security are characterizing many communication and computer systems, especially for trusted computing system [1]. In order to meet the secure requirements of trusted computing system, we usually use extra encrypted module to protect the system. there have been a number of techniques proposed to “crack” these systems. the real-world attacks are usually against the implementation of the techniques and not the theoretical properties of the techniques. Such attacks are called side channel attacks [2-3]. These attacks that are based on "side channel information", that is, information that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process. These attacks are aimed at the physical implementation of the cryptographic algorithms. Among these, power analysis attack is most powerful attack.

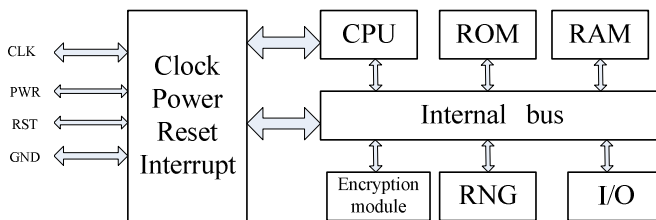
Trusted Computing Group (TCG) proposed Trusted Platform Module (TPM) based trusted system targeting at low end application such as personnel computer (PC), therefore it lacks of the consideration of resisting against power analysis attacks.

But power analysis attack becomes a big threat to TPM when it is applied to high end application such as server. There have two common power analysis attacks called Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [4]. SPA is a technique that involves directly interpreting power consumption measurements during cryptographic operations. DPA is based on statistical analysis in which the attacker can guess the correctness of the keys by comparing the differences between a sample power trace and the correct key power trace. Side Channel Attacks Resistant Design (SCARD) group had done extensive research on side channel attack especially for trusted computing system. Another research group from Motorola's security lab had proposed the countermeasures against side channel attack for trusted computing on mobile devices [5]. A. Schuster generalized a method for DPA attack to AES in the work of [6]. In this paper, we proposed a novel reliable platform dedicated for the trusted computing system with the capability to resist against side channel attack especially for power analysis attack.

## 2 Improved SMS4 against Power Analysis Attacks

### 2.1 SMS4

Active and passive attacks are two ways attacking trusted computing systems. Passive attack is more powerful attack than active one due to no trace and no damage to the original system after attacking. Figure 1 shows the architecture of a normal trusted computing system.



**Fig. 1.** The architecture of a normal trusted computing system

Passive attack is also big threat to TPM. The sensitive data as private key needs to be protected in TPM as it can be easily achieved by using power analysis attack [7]. This is also can be seen by a case study on SMS4 algorithm introduced in the followings [8].

SMS4 is a block cipher with the typical length equaling to 128 bit and key length equaling to 128 bit. The cipher and key expansion states need 32 nonlinear iterative rounds. The procedure of encryption and decryption are similar except the sequences of using round keys which is opposite to each other. The round operation consists of XOR, S-box and left circle shifting. Figure 2 shows the encryption procedure of SMS4.

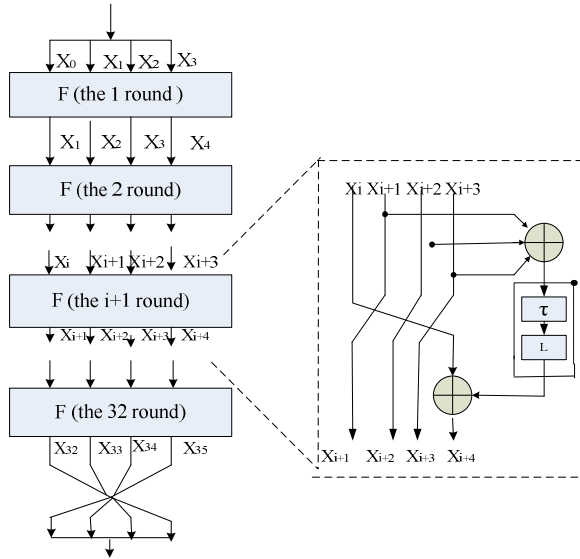


Fig. 2. The encryption procedure of SMS4

The typical computational operations of SMS4 are carried on using word length. We denote one iteration operation, called one round, as function  $F$ . Figure 3 show the procedure of  $F$  of SMS4. There have four S-boxes of SMS4 and each S-box is characterized with 8 bit inputs and 8 bit outputs.

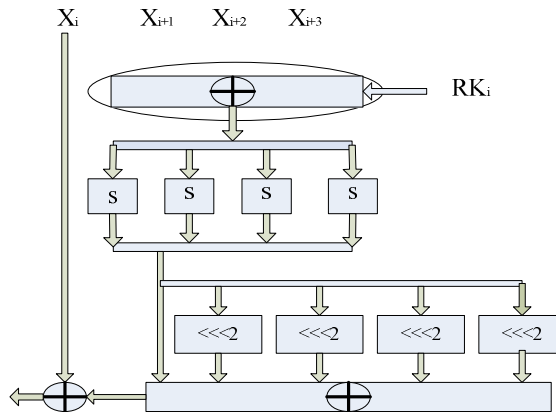


Fig. 3. The procedure of  $F$  in SMS4

According to the introduction of SMS4 algorithm given above, we choose two places as the suitable points for DPA attack. The first attacking point is the XOR operation of the first round of SMS4 algorithm and the second attacking point is the first bit output of S-box. Figure 4 shows the above two attacking points labeled as (1) and (2).



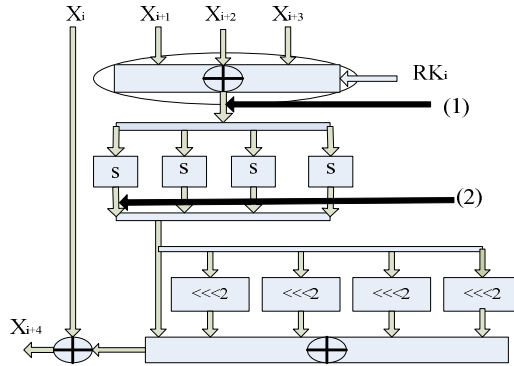


Fig. 4. Two attacking places as (1) and (2)

### 2.2 The Weakness of SMS4 Algorithm

Considering a cryptographic algorithm based on hardware implementation, the systems will consume power when the computations taking place. The quantity of each circuit’s power consumption depends on the exact processing data. In gate level, this is represented as the number of times of charging and discharging of the load capacitances. In register level, this is represented as the number of times of flip-flop flipping from 0 to 1 or from 1 to 0. In operand level, this is also represented as hamming distance (HD) between the two consecutive operand datas. In this paper, we use hamming distance model to mimic the power consumption in the real circuit. The power consumption can be represented in the following equation as:

$$w = aH(D \rightarrow R) + b \tag{1}$$

Where W is power consumption, H is HD, D and R is input and output to the register.  $H(D \rightarrow R)$  is the HD between operands D and R, a and b are two consistent parameters.

When  $a=1, b=0$ , equation (2) can be rewrite as:

$$w = H(D \rightarrow R) \tag{2}$$

### 2.3 Proposed Countermeasure of SMS4 Algorithm

In this paper, we propose a modified fixed value masking algorithm for SMS4. We precompute and store the corresponding S-box for each consistent masks. During the computation, one round not only needs one mask, but also randomly uses several masks. We can efficiently resist against high order power analysis if the attacker do not know the load points of masks. The proposed countermeasure based on [9] is described as follows:

- 1) Precompute n consistent masks, n depends on the area and power consumption of a chip;

- 2) Select one mask is to perform XOR operation with plaintext;  
During the S-box state, 32 rounds use 32 different masks;
- 3) Private key is to perform XOR operation with mask randomly;
- 4) Inverse operation during decryption

The key point of the proposed countermeasure depends on how to generate the fixed value mask. We precompute the needed  $n$  fixed value masks ( $n \geq 2$ ) and store the results in ROM. There also some rules for selecting fixed value masks to avoid simple information leaking. It does not allow to choose all '0' or '1' values for the fixed value masks and the possibility of number of '0' bit must take up around 50% of all the bit for a fixed value mask.

The pseudo-code for generation of S-box, called SboxUpdate, is described as follows:

```

SboxUpdate(S, r)
For (x=0; x<=256; x++)
S'[x]=S[x ⊕ mr] ⊕ mr;
Output S'
    
```

Where,  $x$  is the input plaintext,  $mr]$  is a fixed value mask.  $S'$  is precomputed S-box and stored in ROM. It needs 256 times computation for one fixed value mask.

### 3 Implementation of Improved SMS4

We port our design to Xilinx Virtex II FPGA platform using hardware description language Verilog HDL. The procedure of our implementation is described in Figure 5.

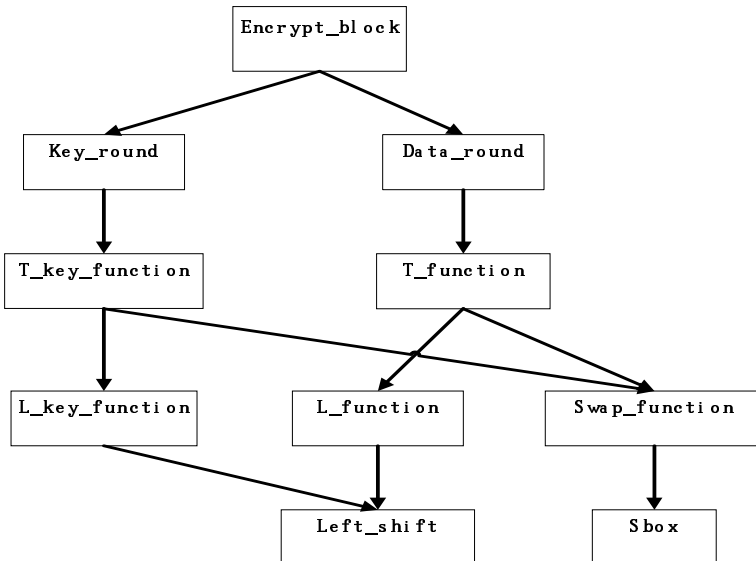


Fig. 5. The procedure of the proposed countermeasure for SMS4

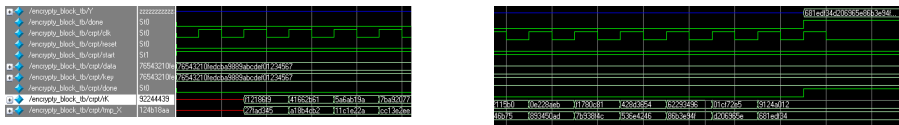
```

#          41 rK = 92244439 tmp_X = 124b18aa
#          43 rK = e89e641f tmp_X = 6ae7725f
#          45 rK = 98ca015a tmp_X = f4cba1f9
#          47 rK = c7159060 tmp_X = 1dcdfa10
#          49 rK = 99e1fd2e tmp_X = 2ff60603
#          51 rK = b79bd80c tmp_X = eff24fdc
#          53 rK = 1d2115b0 tmp_X = 6fe46b75
#          55 rK = 0e228aeb tmp_X = 893450ad
#          57 rK = f1780c81 tmp_X = 7b938f4c
#          59 rK = 428d3654 tmp_X = 536e4246
#          61 rK = 62293496 tmp_X = 86b3e94f
#          63 rK = 01cf72e5 tmp_X = d206965e
#          65 rK = 9124a012 tmp_X = 681edf34
# Y = 681edf34d206965e86b3e94f536e4246
# ** Note: $finish : D:/modism_exercise/SMS4_2/encrypt_block_tb.v(36)
# Time: 1068 ns Iteration: 0 Instance: /encrypty_block_tb
    
```

Fig. 6. Simulation results from ModelSim

Figure 5 shows the design flow chart in detail when coded with Verilog DHL. The simulation result is shown in Figure 6 and it proves the correctness of our proposed architecture.

Figure 7 shows the simulation wave form of the modified SMS4 algorithm. We select only two wave forms representing the whole simulation, (a) representing running time at ‘0’ and (b) representing running time at the end.



(a) at time ‘0’

(b) at the end

Fig. 7. The simulation waveform

#### 4 Power Analysis Attacks Experiment of Improved SMS4

We also mimic the high-order DPA attack to our proposed countermeasure. Figure 8 shows the setup of the experimental environment for the proposed countermeasure.

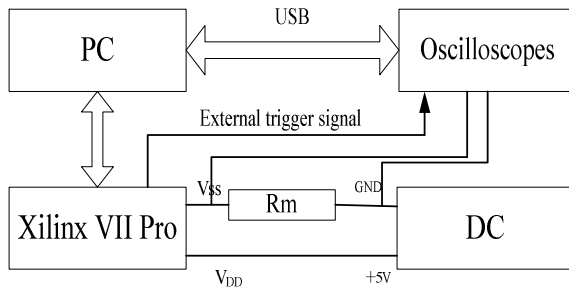
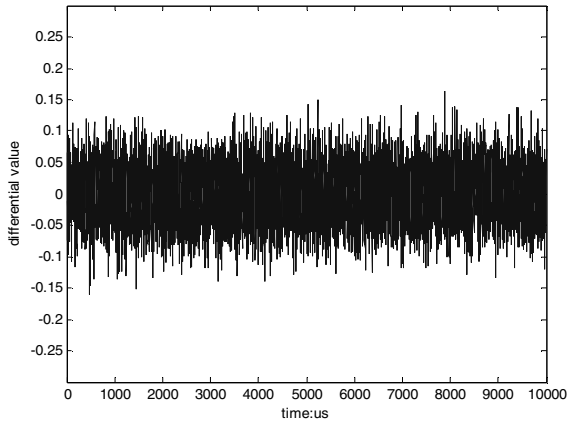


Fig. 8. Experimental environment of the proposed countermeasure

In Figure 8, Plaintext is produced in PC by software control and PC sends it to FPGA through the serial port. After receiving a full plaintext, our proposed algorithm will produce ciphertext. At the same time, an external trigger signal will trigger the oscilloscope to collect and store the waveform results. Oscilloscope will send those data to PC through USB. The channel 1 of the oscilloscope is set to collect the data and the channel 2 is set to connect to the external trigger signal. In this experiment, we use Tektronix DPO4032 digital oscilloscope and collect 50000 samples with 10000 samples depth at 25MS/s.



**Fig. 9.** Experimental results of the proposed countermeasure for DPA attack

From Figure 9, we can see that there does not exist possible guessing point for attacker to know the private key. Therefore, our proposed countermeasure efficiently resist against DPA attack.

## 5 Conclusions

In this paper, we proposed an efficient countermeasure for SMS4 and simulate it in real FPGA platform. We also analyzed SMS4 in details and shows that it can not resist against power analysis attack when ported SMS4 to hardware platform. Therefore, it is necessary to protect the SMS4 using our proposed countermeasure.

**Acknowledgements.** This research is supported by the Key Program of National Natural Science Foundation of China under Grant No.61133005, the Scientific Research Fund of Hunan Provincial Education Department with Grant No 11B018, the Scientific Research fund of Hengyang Normal University with Grant No 12CXYZ01,11B43, Hunan Postdoctoral Science Foundation with Grant No 897203005, The Construct Program for "the 12th Five-Year" Key Disciplines (Optics) in Hunan Province, The Construct Program for "the 12th Five-Year" Key Disciplines (Computer Science) in Hengyang Normal University.

## References

- [1] Jian, M., Yujie, Z.: Trusted platform module countermeasures against hardware attacks. *Information Technology* (6), 27–30 (2006)
- [2] Kocher, P., Lee, R., et al.: Security as a New Dimension in Embedded System Design. In: DAC 2004, San Diego, California, USA, pp. 753–760 (2004)
- [3] Al-Somani, T.F., Amin, A.A.: High Performance Elliptic Curve Scalar Multiplication with Resistance Against Power Analysis Attacks. *Journal of Applied Sciences* 8(24), 4587–4594 (2008)
- [4] Kocher, P., Jaffe, J., Jun, B.: Introduction to differential power analysis and related attacks (1998), <http://www.cryptography.com/public/pdf/DPATechInfo.pdf>
- [5] Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Examining Smart- card Security under the Threat of Power Attack Analysis. *IEEE Trans. on Computers* 51(5), 541–552 (2002)
- [6] Schuster, A.: Differential Power Analysis of an AES Implementation. Technical Report, IAIK-TR2004/06/25
- [7] Li, L., Li, R., Sha, E.H.M.: Survey on Security SOC Against Power Analysis Attack. *Computer Science* 36(6), 16–18 (2009)
- [8] Office of State Commercial Cipher Administration. Block Cipher for WLAN Products-SMS4 (2006), <http://www.oscca.gov.cn/UpFile/200621016423197990.pdf>
- [9] Chang, H., Kim, K.: Securing AES against Second-Order DPA by Simple Fixed-Value Masking. In: CSS 2003, pp. 145–150 (2003)

# Detection of KVM's Virtual Environment and Vulnerability\*

Li Ruan<sup>\*\*</sup>, Yikai Sun, Limin Xiao, and Mingfa Zhu

State Key Laboratory of Software Development Environment,  
Beihang University, Beijing 100191, China  
School of Computer Science and Engineering, Beihang University, Beijing 100191, China  
{ruanli,xiaolm,zhumf}@buaa.edu.cn, sunyikai00@yahoo.com.cn

**Abstract.** Recently, virtualization technology has revived and become the key support of the clouds. KVM (Kernel virtual machine) based on Linux kernel-level achieves its popularity however with security risks. Therefore, the virtual environment and vulnerability detecting of KVM is of theoretical significance and great application value. However, there are few reports on virtual environment and vulnerability detection for KVM. This paper introduces a virtual environment and vulnerability method which includes the CPU cycle based detection algorithms, the internet time detection algorithms and multi-thread counter detection algorithms, etc.. In the vulnerability detection of KVM, denial of service attack is simulated and its detection method is proposed. Function and performance tests are also introduced.

**Keywords:** Virtual machine, virtual environment detection, vulnerability detection.

## 1 Introduction

Recently, the security of virtualization gains increasing attention of academic and industrial parties under cloud computing trends and some researches also point out that virtual machine performance is no longer the most important metric of virtualization under the risk of security. The introduction of virtual machines brings more serious and broader virtual-machine featured security risks besides those for traditional systems. First, in the virtual computing environment, if one virtual machine is hijacked, the other virtual machines on the same physical machine are more possible to be hijacked than non-virtual environment because they share the same physical resources. Secondly, even if all virtual machines are assigned the same

---

\* This work was supported by the Hi-tech Research and Development Program of China under Grant No. 2011AA01A205, National Natural Science Foundation of China under Grant No. 61003015; Beijing Natural Science Foundation(4122042); The fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2012ZX-23 and the Open Research Fund of The Academy of Satellite Application under grant NO. SSTC-YJS-01-03.

\*\* Corresponding author.

severe safety level, because they can communicate without network communication, the degree of risk will be exponentially increased. In addition, the virtual machine provides some additional features. For example, VMware provides functions like a shared folder (Shared Folders), automatically drag and clipboard functions across environments inevitably require additional information between the client and the host opened transmission channel which bring a serious security risk. [1]uses the QEMU's CVE-2011-175 in KVM to conduct a virtual machine escape attack. DOS attack will even make the whole virtual system collapsed [1-6].

Although virtual machine's security gains increasing attention, existing researches focus more on the security detection and evaluation on physical environments; moreover, the security gains few focuses from the initial design of virtual machine. There are few research on security and vulnerability detection and much fewer in KVM(kernel virtual machine[1,5]. The virtual environment introduces new requirement by the coexisting of private network and public network and changing network environment [1,12]. This paper introduces a detection method of KVM's Virtual environment and vulnerability by taking into account of the CPU architecture defects and KVM's characteristics.

## 2 Virtual Environment Detection for KVM

The design idea is taking into the characteristics that there are some special CPU instructions which will execute shorter clock cycle under virtual machine than physical machine into consideration. We can construct a loop program which will compare the run time of that special instruction and the common instruction and then decide whether the current environment is under a virtual machine or not. The algorithm KVMVEDetect is designed as following where the instruction of CPUID is used as an example as in Algorithm 1.

-----  
**Algorithm 1:** KVMVEDetect

**Input:** CPUID

**Output:**  $c$  and Whether the current environment is a virtual environment or not.

1 :  $a1$  = the start time;//Execute a loop instruction under current environment and record the start time  $a1$ .

2:  $LoopTimes$  = 1000;// set an initial counter for the times of this loop instruction.

3:  $a2$  = the end time;// Record the end time when the loop instruction is ended.

4:  $a = a2 - a1$ ;// the total run time of loop  $a$  which is under the environment without special instruction environment

5: Add instruction CPUID into each loop;

6: Repeat 1-3 and record the start time  $b1$  with loop and end time  $b2$ ,

7.  $b = b1 - b2$ ;// the total run time of loop which is under the environment with CPUID

8.  $c = b/a$ ;

9. Judge whether the current environment is a virtual environment using  $c$ .

---

Because the addition of special instruction CPUID, it will take more time loop, therefore  $b > a$ . Moreover, because the defects of CPU architecture, CPUID will take different time under virtual environment and physical environment and the traditional loop will take the same time. Therefore, the value of  $c$  will be different. There are three different methods for our detection algorithms: (1) CPU cycle based detection method which uses the CPU's fixed clock cycle counter to implement detection. (2) Network time based detection method which uses outer resources, i.e. network time to implement detection. (3) Multi-thread counter detection method which adds a self-defined counter into the subthreads to implement detection.

## 2.1 CPU Cycle Based Detection

### 2.1.1 Design Idea

The fixed CPU clock cycle(TSC) is used as a baseline to record the time for executing the same special instruction. When the program starts, the counter runs. When the program's loop ends, a signal is returned to CPU and the timer of CPU ends together. The running time of the program is achieved by comparing the difference of the time and the ration is used to judge whether it is now in the virtual environment.

### 2.1.2 Implementation Method

Initialize compile instruction by call `clock_gettime(CLOCK_THREAD_CPUTIME_ID,&time)` to get current CPU time; Set the counter to 0; Loop till the counter to be the set value (e.g. 1000); Call the function `clock_gettime` to gain the current CPU time; difference the two time value to get the total time of the program running. If the special instruction of CPUID is removed from the program to get the total time of another program running. Finally, by judging the ratio of with and without special instruction of CPUID, whether it is under virtual environment is judged. The algorithm is defined as following Algorithm 2.

---

**Algorithm 2:** CPUCycleBasedDetection()

**Input:** `clock_gettime`

**Output:**  $c$ .

1. Call `clock_gettime` to achieve the current CPU time  $a1$ ;
  - Set the integer variable  $i = 0$ . Loop  $(i = i + 1)$  till  $i = 1000$ ;
  2. Call `clock_gettime` to achieve the current CPU time  $a2$ ;
  3.  $a = a2 - a1$ ;
  4. Call `clock_gettime` to achieve the current CPU time  $b1$ .
  5. Set the integer variable  $i = 0$ . Loop  $\{ (i = i + 1) ; \text{CPUID}; \}$  till  $i = 1000$ ;
  6. Call `clock_gettime` to achieve the current CPU time  $b2$ ;
  7.  $b = b2 - b1$ ;
  8.  $c = b/a$ ;
  9. Judge whether it is under virtual environment with  $c$ .
-



## 2.2 Network Time Based Detection

### 2.2.1 Design Idea

The outer resources are used to achieve network time during the program runs NTP protocol to call Internet time. First, when the program starts, the start time  $a1$  of Loop without special instruction is achieved by using NTP protocol to achieve current Internet time. The end time of Loop without special instruction is recorded as  $a2$ . Then we add the special instruction. The start time of Loop with special instruction is  $b1$ . The end time of Loop with special instruction is  $b2$ . Then the total time during the program runs by using the difference  $c = (b2 - b1) / (a2 - a1)$ .

### 2.2.2 Implementation Method

Initialize compile instruction by call NPT protocol to get current network time; Set the counter to 0; Loop till the counter to be the set value (e.g. 1000); When the loop finishes, call the NPT protocol to gain the current network time; difference the two value to get the total time of the program running. If the special instruction of CPUID is removed from the program to get the total network time of another program running. Finally, by judging the ratio of with and without special instruction of CPUID, whether it is under virtual environment is judged. The algorithm is defined as following Algorithm 3.

---

**Algorithm 3:** NetworkTimeBasedDetection()

**Input:** NPT protocol

**Output:** c

1. Call NPT protocol to achieve the current network time  $a1$ ;
  2. Set the integer variable  $i = 0$ . Loop ( $i = i + 1$ ) till  $i = 1000$ ;
  3. Call NPT protocol to achieve the current internet time  $a2$ ;
  4.  $a = a2 - a1$ ;
  5. Call NPT protocol to achieve the current network time  $b1$ .
  6. Set the integer variable  $i = 0$ . Loop { ( $i = i + 1$ ) ;CPUID;} till  $i = 1000$ ;
  7. Call NPT protocol to achieve the current CPU time  $b2$ ;
  8.  $b = b2 - b1$ ;
  9.  $c = b/a$ ;
  10. Judge whether it is under virtual environment with c.
- 

## 2.3 Multi-thread Counter Based Detection

### 2.3.1 Design Idea

This method will use two threads. In the first thread, the loop time will be recorded with the running of program. The second thread will communicate with the first thread when the program starts and the counter is increased. When the counter arrives a fixed value(e.g. 1000), the second thread will communicate with the first thread. The output time of the second thread is the final results for the comparison.

### 2.3.2 Implementation Method

Initialize compile instruction by call NPT protocol to get current network time; Set the counter to 0; Loop till the counter to be the set value (e.g. 1000); When the loop finishes, call the NPT protocol to gain the current network time; difference the two value to get the total time of the program running. If the special instruction of CPUID is removed from the program to get the total network time of another program running. Finally, by judging the ratio of with and without special instruction of CPUID, whether it is under virtual environment is judged. The algorithm is defined as following Algorithm 4.

---

**Algorithm 4:** NetworkTimeBasedDetection()

1. Call *pthread\_create* to create the child process;
  2. In thread2, set the integer variable  $b=0$ . Loop ( $b = b + 1$ ) till  $i=1000$ ; Communicate with thread1 by the function *signal*.
  3. In thread1, set the integer variable  $a=0$ . When the signal is received, Loop  $\{(a = a + 1); CPUID\}$ .
  4. When  $a$  in the thread2 received 1000, communicate with thread1 by the function *signal* and thread1 stops.
  5. Achieved  $a$  and  $b$ ;
  6.  $c = b/a$ ;
  7. Judge whether it is under virtual environment with  $c$ .
- 

## 3 Vulnerability Detection for KVM

DOS attack vulnerability detection algorithm is designed as following Algorithm 5.

---

**Algorithm5:** KVMVulDetect

**Input:** shell

**Output:** vulnerability report

- 1: Execute shell which collects the server information;
  2. Print process whose CPU utility $>0$  to the file CPUUtility.tex; process ID = the first process's ID;
  - 3:  $a =$  CPU utility of the process by Read one line of CPUUtility.tex.
  - 4: If  $a>90$  goto 6 else goto 5;
  - 5: If the end line of CPUUtility.tex, execute sleep; pause 5 seconds, goto 1 else go to 3;
  - 6:  $char[x] =$  command of current process;  $t =$  the times of the process ID exists; execute sleep and pause 1 second; goto 7.
  - 8: loop step1 to step 5 for three times; goto step9;
  - 9: If  $t = 3$  goto step10 else goto step1;
  - 10: Output  $char[x]$  ; print "there is a risk and whether to kill this process";
  - 11: If kill, the process is terminated; else record this process as a secure process and its process ID. If the ID exists in Step 2, process ID; goto 3.
-

## 4 Experiments and Results Analysis

### 4.1 Test Environment

Table 1 gives the hardware and software environments of our experiment.

Hardware environment: Intel Xeon 5160 3.00GHz with multi-cores and 4 threads CPU. DDR2 667MHz memory of 4G. OS: Ubuntu Server 10.04.2; Development environment: C, shellcode, LINUX GCC and VIM.

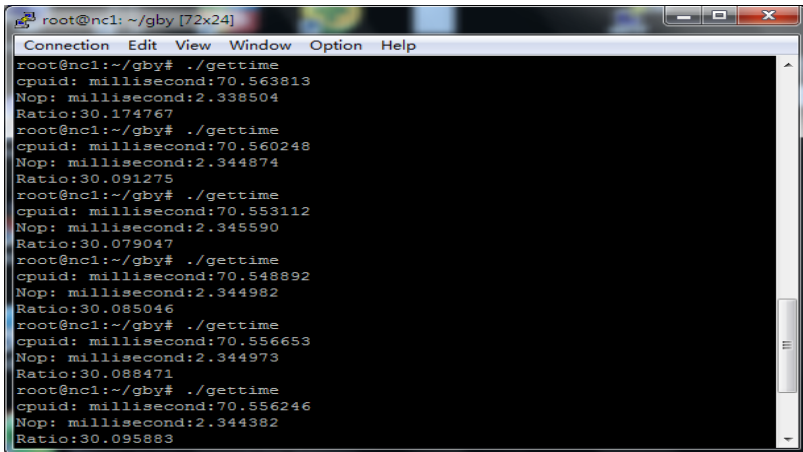
**Table 1.** Hardware and Software Configurations

|                         |  |
|-------------------------|--|
| CPU                     | Intel Xeon 5160 3.00GHz dual core with 4 threads |
| Memory(RAM)             | DDR2 667MHz 4*1G                                 |
| OS                      | Ubuntu Server 10.04.2                            |
| Development Language    | Visual C,Shellcode                               |
| Development environment | Vim, Gcc   |

### 4.2 Function Test of Virtual Environment Detection

#### 4.2.1 Virtual Environment Detection

Experimental process: Run detection program in the LINUX environment and KVM virtual environment in the server; analysis the results.



```

root@nc1: ~/gby [72x24]
Connection Edit View Window Option Help
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.563813
Nop: millisecond:2.338504
Ratio:30.174767
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.560248
Nop: millisecond:2.344874
Ratio:30.091275
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.553112
Nop: millisecond:2.345590
Ratio:30.079047
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.548892
Nop: millisecond:2.344982
Ratio:30.085046
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.556653
Nop: millisecond:2.344973
Ratio:30.088471
root@nc1:~/gby# ./gettime
cpuid: millisecond:70.556246
Nop: millisecond:2.344382
Ratio:30.095893

```

**Fig. 1.** Detection results in Linux

```
root@test160: ~ [72x24]
Connection Edit View Window Option Help
root@test160:~# ./gettime
cpuid: millisecond:34.394721
Nop: millisecond:13.480446
Ratio:2.551453
root@test160:~# ./gettime
cpuid: millisecond:34.059890
Nop: millisecond:14.564377
Ratio:2.338575
root@test160:~# ./gettime
cpuid: millisecond:34.983212
Nop: millisecond:14.360094
Ratio:2.436141
root@test160:~# ./gettime
cpuid: millisecond:33.590742
Nop: millisecond:13.720242
Ratio:2.448262
root@test160:~# ./gettime
cpuid: millisecond:34.477879
Nop: millisecond:12.875669
Ratio:2.677754
root@test160:~# ./gettime
cpuid: millisecond:34.038413
Nop: millisecond:14.523170
Ratio:2.343732
```

Fig. 2. Detection results in KVM

Results : As is shown in Fig.2 , c both with or without CPUID fluctuates around 30 in Linux and fluctuate between [2,3] under KVM. The value is different and effective to differentiate the virtual and physical environments.

### 4.2.2 DoS Attack Simulation

In KVM, we open DoS attack process and investigate the change of CPU utility in the server using TOP command and record once five seconds. The result is as following Fig. 3

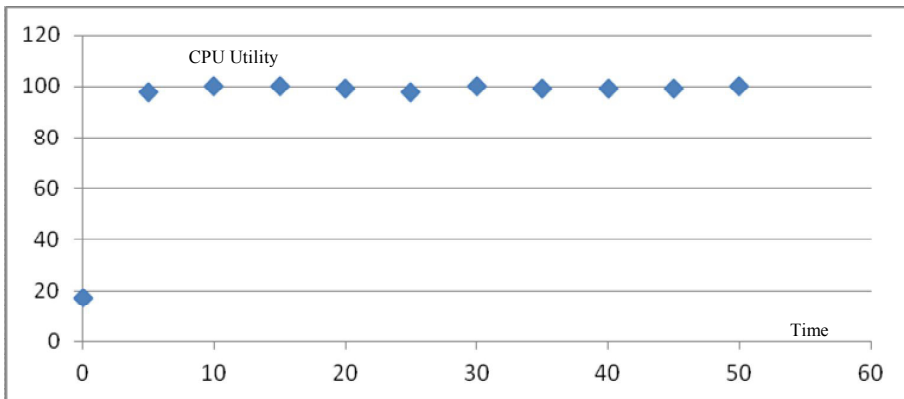


Fig. 3. Effect of DOS Attack to Virtual Machine Communication



As is shown in Fig.5, communication can finish after the fork process in the attacked virtual machine is closed.

From this experiment, we can see that DOS is successfully emulated.

### 4.2.3 KVM Vulnerability Detection

#### (1) Simulate DOS attack

Test procedure: Run the DOS attack simulation process in KVM virtual machine. Then, we record the CPU utility every 5 seconds using TOP.

Test results : we find that the CPU utility is above 98% except the time the program initiates.It shows that DOS simulation (i.e.,CPU resources are mostly occupied simulation )results are successfully achieved.

#### (2) Detect DOS attack

Test procedure: Repeat the running of the DOS attack close process in KVM virtual machine. Run the vulnerability detection program. Observer the CPU utility and report the attack to the

Test results: As is shown in Table 2, when the DOS attack process is running, our program can successfully detect the risky status and report to user. When the DOS attack process is closed and CPU utility is relatively low, there is no report to user. This result verified the effectiveness of our detection software and stability of our results.

**Table 2.** Function test results

| Status | Open | Close | Open | Close | Open | Close | Open | Close | Open |
|--------|------|-------|------|-------|------|-------|------|-------|------|
| CPU%   | 100  | 57    | 99   | 27    | 100  | 78    | 98   | 45    | 100  |
| Report | Y    | N     | Y    | N     | Y    | N     | Y    | N     | Y    |

## 4.3 Performance Test

### 4.3.1 KVM Virtual Environment Detection

Our basic idea for virtual environment detection is that the difference of Linux and KVM environments can be detected by comparing the running time . (1)CPU cycle counter based detection method: under Linux system, the value is distributed between [30,35] with or without special instruction ( CPUID ) .Under KVM virtual environment , the value is distributed between [2,4] with or without special instruction (CPUID) . These results are relatively stable.(2)multithreads counter based detection method: under Linux system, the value is distributed between[31,34]and [15,18] with or without special instruction (CPUID) .Under KVM virtual environment , the value is distributed between [2,5] with or without special instruction (CPUID) . These results fluctuated greatly and are not stable.

Environment adaptability test: Just because the above tests are performed under single server, to avoid the randomness, we also performed the following tests in different platforms. As is shown in the Table 3 and Table 4, the three algorithms can apply to different environments.

**Table 3.** Comparison of Different virtual environment methods

| Comparison             | CPU cycle based method | Network time based method | Multithreads counter based method |
|------------------------|------------------------|---------------------------|-----------------------------------|
| Stability              | High                   | Middle                    | Relative Low                      |
| Accuracy               | High                   | Middle                    | High                              |
| Anti-attack difficulty | Low                    | Middle                    | High                              |

**Table 4.** Detection accuracy

| # of created processes | 1  | 2  | 5  | 10 | 15 | 20 | 50  | 80 | 100 |
|------------------------|----|----|----|----|----|----|-----|----|-----|
| CPU%                   | 19 | 22 | 45 | 67 | 92 | 99 | 100 | 99 | 100 |
| Report                 | N  | N  | N  | N  | N  | N  | N   | Y  | Y   |

**4.3.2 Vulnerability Real-Time Detection**

Test procedure: Open limited threads in virtual machine and run vulnerability detection program. Record CPU utility and report to user of the attacks. From Table 3 we can find out that, even CPU utility is up to 98% in a short time, the detection software will not report. Only CPU utility remains high, the detection software will report user and suggest to deal with the vulnerability.

**5 Conclusions and Future Work**

This paper introduced a detection method of KVM’s virtual environment and vulnerability. Both function and performance test results verified its effectiveness. We are now research on more automatic vulnerability detection method and vulnerability detection method for XEN .

**Acknowledgments.** We express our gratitude for the related students in Beihang University who participated in our project in [13].

**References**

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. *J. Mol. Biol.* 147, 195–197 (1981)
2. May, P., Ehrlich, H.-C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) *Euro-Par 2006. LNCS*, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco (1999)

4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, <http://www.ncbi.nlm.nih.gov>
7. Karger, P.A.: Is Your Virtual Machine Monitor Secure? In: Third Asia-Pacific Trusted Infrastructure Technologies Conference, October 14-17, p. 5 (2008)
8. Wu, H.Q., Ding, Y., Winer, C., Yao, L.: Network security for virtual machine in cloud computing. In: 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), November 30-December 2, pp. 18–21 (2010)
9. Christopher, T., Maria, H., Chad, L.: Virtualization Detection: New Strategies and Their Effectiveness (Ph.D. Thesis). Minneapolis, MN 55455: University of Minnesota (2010)
10. Peter, F.: Attacks on Virtual Machine Emulators. Symantec Advanced Threat Research (2008)
11. Mueller, S.: KVM Security Comparison (November 02, 2009), [http://www.atsec.com/downloads/white-papers/kvm\\_security\\_comparison.pdf](http://www.atsec.com/downloads/white-papers/kvm_security_comparison.pdf)
12. Bazargan, F.A., Yeun, C.Y., Zemerly, J.: Understanding the security challenges of virtualized environments. In: 2011 International Conference for Internet Technology and Secured Transactions (ICITST), December 11-14, pp. 67–72 (2011)
13. Sun, Y.K.: Detection of KVM's Virtual Environment and Vulnerability. Bachelor Thesis. Supervised by Ruan, L. (2012)



# Scalability Tests of a Finite Element Code on Hundreds of Thousands Cores and Heterogeneous Architecture

Jiangyong Ren, ChaoWei Wang, Yingrui Wang, and Rong Tian

HPC Research Center, Institute of Computing Technology,  
Chinese Academy of Sciences, 100190 Beijing, China  
{remjiangyong, wangchaowei, wangyingrui, rongtian}@ncic.ac.cn

**Abstract.** A multi-scale finite element method code, msFEM, is tested on Jaguar and Nebulae, two petaflops computers that were listed as #1 and #2 on the Top500 list of June 2010 at the time of the tests. The flat MPI version of msFEM is scaled from 20K up to 200K CPU cores on Jaguar, delivering 70% parallel efficiency at the 200K cores with a finite element model of eight millions of degrees of freedom. GPU versions, in both double precision and mixed precision coded through MPI+OpenMP+CUDA hybrid programming, 900 GPU nodes on Jaguar and 1500 GPU nodes on Nebulae, achieving remarkable 90+% parallel efficiency on the systems. The mixed-precision GPU version delivers further 1.5 times of speedup over the fully double precision version with no significant implementational cost. The large-scale tests support that the msFEM runs efficiently on petaflops computers and is highly potential for domain applications at extreme-scale.

**Keywords:** Tens of Thousands Cores, Scalability, GPU, Mixed Precision, Finite Element Method.

## 1 Introduction

The frontier of computational physics and engineering is to address the challenge of high fidelity modeling and simulation of real complex systems, the need to completely transform the discipline of computer simulation into predictive science [1]. A real physical problem usually is an interaction process involving multiple physics (i.e., multi-physics) and the interaction crossing a broad spectrum of spatial and temporal scales (i.e., multi-scale). As petaflops computers become readily available, computational scientists are stepping into a completely new era, a times when the coupling of multi-physics that has not been feasible before now may become affordable. Direct numerical simulation (DNS), which, in a broad sense, means a simulation with least phenomenological assumption and parameterization in the whole process, is commonly accepted as the “converged” solution and a major path leading to the predictive science and it causes broad attention in the community of computational science. For example, DNS can be conceived as a commonly shard core feature of the 6 early exascale science codes selected on Titan, the #1 computer

on the Top500 list of November 2012 [2]. The six codes are respectively S3D for combustion [3], CAM-SE for climate [4], DENOVO for nuclear reactor designs [5], LAMMPS for molecular dynamics [6], PELOTRAN for computational geo-science [7], and WL-LSMS for first principle calculation [8].

Material strengthening is the centuries-long standing and ultimate goal of material science. The material strengthening, in nature, is to introduce various kinds of defects (e.g., point, line, planar and volume defects etc) to pin dislocation. However, material strengthening invariably leads to concomitant decreasing plasticity and toughness, showing a strength-toughness trade off. Quantitative understanding the link between multiscale microstructures and properties of materials (Fig. 1) is fundamentally crucial for improving fracture toughness while keeping high strength [9][10].

The promise of computational science in the extreme-scale computing era is to reduce and decompose macroscopic complexities into microscopic simplicities with the expense of high spatial and temporal resolution of computing. Direct combination of 3D microstructure data sets and 3D large-scale simulations provides transformational opportunity for the development of a comprehensive understanding of microstructure-property relationships in order to systematically design materials with specific desired properties.

We have developed a framework simulating the ductile fracture process zone in microstructural detail (Fig. 2) [11]-[19]. The experimentally reconstructed microstructural dataset is directly embedded into a continuum mesh model to improve the simulation fidelity of microstructure effects on fracture toughness. For the first time, the linking of fracture toughness to multiscale microstructures in a realistic 3D numerical model in a direct manner has been accomplished (Fig. 2 and Fig. 3) [19]. Reported in the paper is the scalability tests of the developed multiscale simulation code, msFEM, up to 200K CPU cores and 900 GPU nodes on Jaguar and 20K CPU cores and 1500 GPUs cards on Nebulae, the two petaflops computers [2].

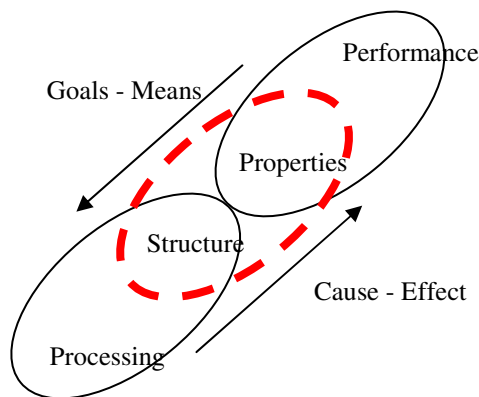
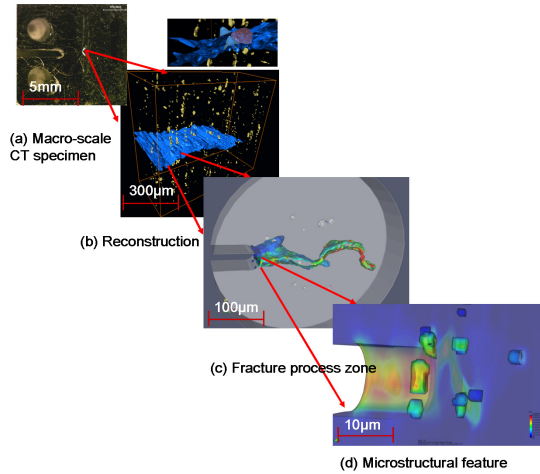
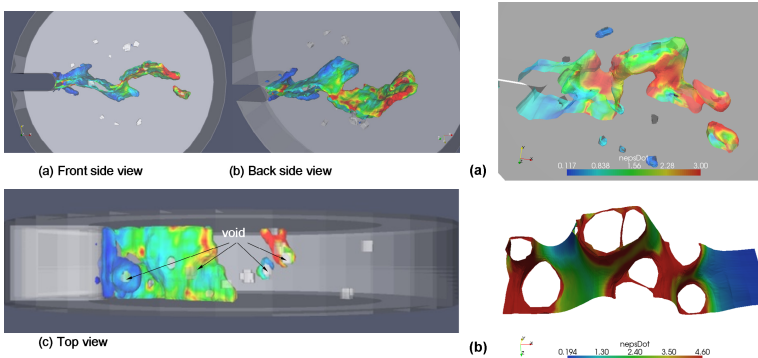


Fig. 1. Missing structure-property link in material designs [10]



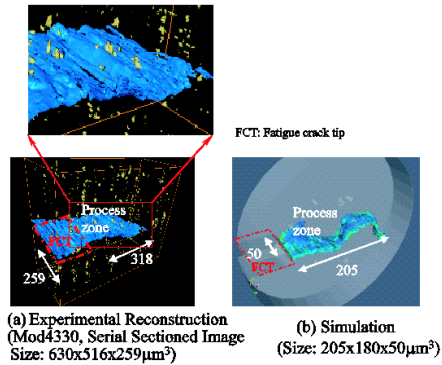
**Fig. 2.** Three dimensional microstructure reconstructions and simulations of fracture process zone [17][18][19]. (a) and (b) show the crack tip specimen and microstructure reconstruction providing the microstructures within the fracture process zone and crack opening displacement (COD) versus the applied load, respectively. Using high performance computing, a 3-dimensional microstructure simulation ((c), (d)) reveals clearer micro-structural features and interplay during the development of the fracture process zone and provides a deeper understanding of the effects of microstructures on materials properties.



**Fig. 3.** Microstructural features of fracture process zone revealed through numerical simulation.

## 2 msFEM—A Multi-scale Finite Element Method Code

The physical “multiscale nature” of extreme-scale domain applications determines the time resolution needed to be resolved has to be small. When the physics at the small scale is dominant, which is usual in multiscale phenomena, the small time step is not only a requirement of the numerical stability, but more importantly also the need to capture the rapid evolving physics at sub-scale. The sub-scale rapid evolving physics



**Fig. 4.** The experimental verification

and the excellent scalability of explicit methods explain the current situation that around 2/3 applications on today's petaflops computers worldwide are either explicit FEM/FDM or MD/particle simulation, for example on Titan [20].

msFEM is an explicit code for multiscale analyses of material deformation, which is a 3D, parallel, nonlinear, large-deformation, multiscale code written in C++. The code is based on gradient plasticity theories [21]-[23] and the framework of generalized finite element methods [24]-[28]. The continuum model employed can be switched between the classical continuum model and the multi-resolution continuum model [11]-[19]. The generalized finite element framework allows an arbitrary number of degrees of freedom (dof) per node. The code has been used on BlueGene/L, Dawning 5000A, Nebulae and Jaguar (refer to [2] for the details of the computers).

An explicit FEM is to solve explicitly Newton's second law,  $Ma=F$ , where  $M$  is the mass matrix and  $F$  is the vector of force. The most of calculation is spent on computing  $F$ , approximately occupying 90-94% of the overall time-to-solution. The update of acceleration, velocity, and displacement occupies about 1-2%, data IO around 2-3%, and the rest around 3%.

## 2.1 Implementations of msFEM

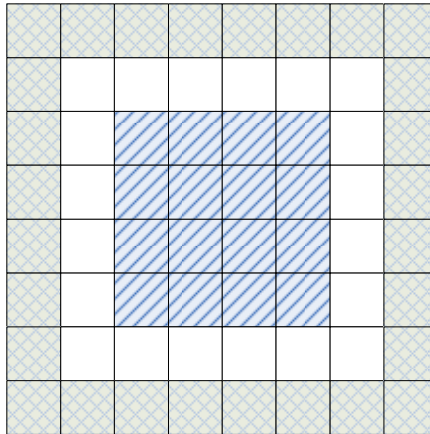
msFEM has three different parallel implementations: (1) flat MPI version, msFEM\_CPU; (2) CPU+GPU double precision version, msFEM\_GPU\_double; (3) CPU+GPU mixed precision version, msFEM\_GPU\_mixed. The three implementations are detailed as follows:

1. msFEM\_CPU: each CPU core runs a MPI process based on a traditional flat MPI programming model;
2. msFEM\_GPU\_double, developed for GPU accelerated clusters. Each compute node runs a MPI process, the cores of CPU and GPU are collectively managed by multiple threads. It is based on a MPI+OpenMP+CUDA hybrid parallel programming model. All float point operations are done in double precision in this version.

3. msFEM\_GPU\_mixed, a mixed precision version of msFEM\_GPU\_double. The mixed precision is an acceleration strategy on vector units such as SSE and GPU using low precision for the majority of float point operations while maintaining the accuracy of the solution of application. The purpose of the mixed precision is to accelerate the time-to-solution by fully taking advantage of the compute power of low precision over high precision on vector units and increasing the throughput of network and memory access by doubling the usage efficiency of bandwidth.

## 2.2 Parallelization and Acceleration Techniques in msFEM

msFEM decomposes the domain of interest using METIS [29], a graph partitioning open source code. Each sub-domain is assigned to and computed by one MPI process. Fig. 5 illustrates such a sub-domain without losing generality. The finite elements of each sub-domain are grouped into two parts: an internal zone composed of the 4x4 elements in the center in the figure and a halo zone composed of the outer two layers of elements. The internal zone is called internal because data are self dependent and data exchange with neighboring processes is not needed. In the halo zone, the outermost layer of elements is called a ghost layer and the second to the outermost is called a boundary layer. Stepwise calculation in an explicit FEM includes two parts: (1) to compute on the internal zone, (2) to send the boundary layer to neighboring processes, to receive the data into the ghost layer from neighboring processes and to compute on the boundary layer after the communication is done with success.



**Fig. 5.** Partitioned sub-domain of a FE mesh model

Parallelization is mainly concerned about:

### 1. Domain partition

There are two ways to partition a mesh: node-based partition and element-based partition. For an explicit method, the element-based partition leads to a shorter communication boundary and hence less elements that require inter-subdomain data exchange. The element-based partition is adopted in msFEM.

## 2. Asynchronous communication

Overlapping communication with computation is a major optimization strategy in a parallel finite element code. The detailed procedure is to first start non-blocking communication of the halo data exchange before the computation on the internal zone so that the communication of the halo zone can be overlapped by the computation on the internal zone. When the internal zone contains an enough number of elements, the computation on the internal zone can well hide the time spent on the data exchange of the halo zone.

## 3. Terabyte data IO

Aside the communication bottleneck, data IO becomes a new bottleneck when the number of MPI processes is beyond tens of thousands. The sequential data IO is definitely out of consideration for a simulation at scale. On the other hand, once the number of MPI processes exceeds several tens of thousands (for example 40K MPI processes), process-wise IO operations exert huge pressure on a parallel file system, resulting in, for example, the contention of metadata server in a parallel file system like Lustre on Jaguar. msFEM avoids the contention by collecting IOs operation into a small set of the MPI processes involved in a simulation.

### 2.2.1 GPU Acceleration

GPU acceleration follows several quite standard techniques:

1. Packed data transfer between CPU and GPU. This includes to re-form data-structure to improve memory access speed;
2. Asynchronous execution. Data to be computed on GPU is partitioned. Data transfer and computation is pipelined to hide the time of data transfer between CPU and GPU.
3. Memory optimization, including register, constant memory, texture memory.

### 2.2.2 Mixed Precision Computation

Mixed precision in computation can increase throughput of memory access and communication while doubling FLOPs per second. A mixed precision algorithm is to fully take advantage of low precision computation while not sacrificing the numerical accuracy of the application solution. This has been shown advantageous on GPU [41][43][44][45], FPGA [42].

The idea of mixed precision can be traced back to the iterative refinement [30]-[39] in solving a system of linear equations, which is first developed by Wilkinson in 1948 [30].

The iterative refinement can be stated as (1)(2)(3)(4)

$$\mathbf{x}^{(0)} = \mathbf{0} . \quad (1)$$

$$\mathbf{d}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} . \quad \text{compute residual in } \textit{high} \text{ precision} \quad (2)$$

$$\mathbf{A}\mathbf{c}^{(k)} = \mathbf{d}^{(k)} . \quad \text{solve equation system in } \textit{low} \text{ precision} \quad (3)$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{c}^{(k)} . \quad \text{accumulate solution in } \textit{high} \text{ precision} \quad (4)$$

Wilkinson and his collaborators [30] verified that if matrix  $A$  is not ill-conditioned, iterative refining  $x$  using mixed precision can reach the same precision done by fully high precision computation. For the above iterative refinement, 90% float point operations can be done with low precision while not affecting the precision of solution. The effectiveness of the mixed precision calculation is dependent on whether there is precision loss. Langou et al. (2006) applied the mixed precision algorithm to dense matrices and verified the effectiveness of the mixed precision algorithm on Cell and other popular CPU architectures [36]. Goddeke et al. (2008) employed the iterative refinement in a multigrid method to solve a large system of linear equations [45]. Those work showed that the solution obtained by using the mixed precision algorithm is the same as that obtained using fully high precision.

To the best knowledge of the authors', the mixed precision idea is mostly applied to solving a system of linear equations (for examples [30]-[51]) and remains yet investigated in an explicit solve. We have extended the idea to an explicit finite element method [52][53][54]. The mixed precision explicit FEM is described as follow:

$$\mathbf{m}\Delta\mathbf{a}^{k+1} = \beta\Delta\mathbf{f}^{k+1} . \quad (5)$$

$$\mathbf{a}^{k+1} = \mathbf{a}^k + \Delta\mathbf{a}^{k+1}/\beta . \quad (6)$$

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \mathbf{a}^{k+1}\Delta t . \quad (7)$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{v}^{k+1}\Delta t . \quad (8)$$

The basic idea of the above mixed precision algorithm is to solve an incremental form of Newton's second law using low precision and to update the acceleration using high precision to avoid precision loss (5)(6)(7)(8). A scaling parameter  $\beta$  is introduced to avoid the underflow of small incremental values  $\Delta\mathbf{a}$  and  $\Delta\mathbf{f}$ .  $\beta$  is chosen to be  $\|\mathbf{f}\|/n$ , where  $n$  is the length of the vector  $\mathbf{f}$  [53][54].

### 3 Tests and Results

msFEM is tested on Jaguar and Nebulae, which are, respectively, installed in Oak Ridge National Lab. and Chinese National Center for HPC at Shenzhen. The two computers were respectively listed as #1 and #2 on the Top500 list of June of 2010, the time of the tests having been done.

Jaguar is equipped with AMD Opteron 6274 CPUs and NVIDIA Tesla 2050 GPUs (in the time of tests, not the GPU is replaced by Tesla 2090). The whole system is composed of 18688 compute nodes (299008 CPU cores). At the time of the tests, Jaguar underwent an upgrade to Titan and the whole system contains two partitions: a CPU partition containing 17728 compute nodes and a GPU partition containing 960 GPU-accelerated compute nodes. msFEM\_CPU, the flat MPI version, has been tested on the CPU partition, while msFEM\_GPU, both the full and the mixed precision

versions, have been tested on the GPU partition. Each compute node of Nebulae is equipped with an Intel X5650 CPU and a NVIDIA Tesla C2050 GPU. The available nodes at the time of the test are 2000, each equipped with a GPU processor. The tests of msFEM\_CPU use CPUs only while the tests of msFEM\_GPU are co-processed by both CPUs and GPUs. Those codes are equivalent in their physical model and the target of domain application. The msFEM codes, by different architecture-oriented implementations, can well reveal a better matched parallelization scheme with today's petaflops computer architecture to achieve the same application target.

### 3.1 Tests of msFEM\_CPU

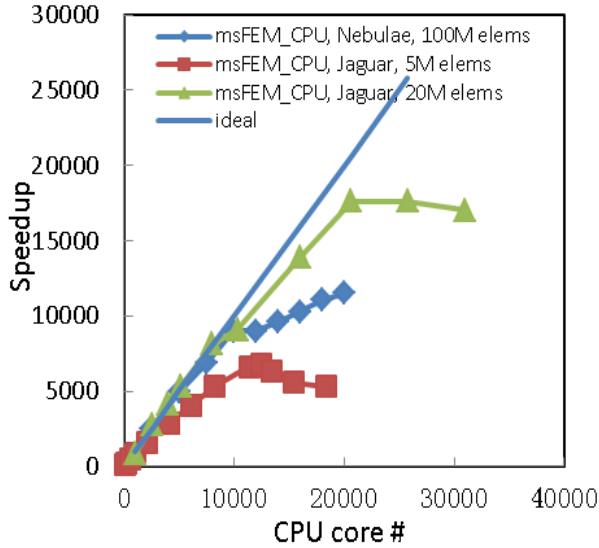
The finite element method implemented in the msFEM code computes 9 dofs per node in 3D (which is 3 usually), which is more communication sensitive than the usual finite element method. The tested mesh model on Nebulae is a mesh of 100M tetrahedral elements. Time to solution is an averaged value of 10 steps. The time to solution versus the number of CPU cores is listed in Table 1 and plotted in Fig. 6.

Two mesh models are tested on Jaguar: a mesh of 5M tetrahedral elements and a mesh of 20M tetrahedral elements. The test results are also plotted in Fig. 6 for a comparison with those obtained on Nebulae. It is seen that speedup saturates at 25800 CPU cores for "Jaguar(20M elems)" and 13468 CPU cores for "Jaguar(5M elems)", respectively. The reason is that the number of elements allocated to each CPU core at the two critical points is too small such that the size ratio of the internal zone to the halo zone is too small to largely overlap the time of communication. As the increase of the model size from 5M to 20M elements, the speedup curve converges toward the ideal case, signaling a good weak scalability of the msFEM code. msFEM\_CPU shows good strong scalability on Nebulae with linear speedup within the scale of 10K CPU cores. Beyond the scale, speedup slows down but still delivers a linear scalability. Based on the plots in Fig. 6, a comparison of the performance of two petaflops systems is also made possible: the both computers show similar performance within the scale of 10K CPU cores but Jaguar outperforms Nebulae at the scale beyond 10K CPU cores. This is also intuitively consistent with user experience with the tests on the systems.

**Table 1.** Strong scalability tests of msFEM\_CPU on Nebulae

| # of processes | Time | Parallel efficiency(%) |
|----------------|------|------------------------|
| 2500           | 11.5 | 100                    |
| 5000           | 5.8  | 99                     |
| 7500           | 4.2  | 91                     |
| 10000          | 3.2  | 89                     |
| 12000          | 3.2  | 74                     |
| 14000          | 3.0  | 68                     |
| 16000          | 2.8  | 64                     |
| 18000          | 2.6  | 61                     |
| 20000          | 2.5  | 57                     |





**Fig. 6.** Strong scalability of msFEM\_CPU(2500-20K CPU cores)

In order to test msFEM\_CPU's performance with an extremely large mesh model, the code has been tested up to 200K CPU cores on Jaguar (Table 2 and Fig. 7). The test model is a mesh of 2.7 billion, approximately 8 billion of dofs. The computation at the scale of 200K CPU cores utilizes around 2/3 computing resources of Jaguar. Table 2 lists the test results from 20K to 200K CPU cores. The parallel efficiency at 200K CPU cores is 71%. As shown in Fig. 7, msFEM\_CPU is able to deliver an approximately linear speedup up to 80K CPU cores with the mesh model of 8 billion of DOFs; from 80K to 200K CPU cores the rate of speedup decreases but speedup is still evident. In summary, msFEM\_CPU shows a good strong scalability up to 200K CPU cores on Jaguar and the code utilized efficiently 2/3 computing resources of the petaflops system.

**Table 2.** Strong scalability tests of msFEM\_CPU on Jaguar up to 200K CPU cores

| # of processes | Time | Speedup |
|----------------|------|---------|
| 20000          | 70.9 | 1.00    |
| 40000          | 34.5 | 2.05    |
| 80000          | 18.2 | 3.89    |
| 100000         | 15.2 | 4.66    |
| 160000         | 11.0 | 6.44    |
| 200000         | 10.0 | 7.09    |

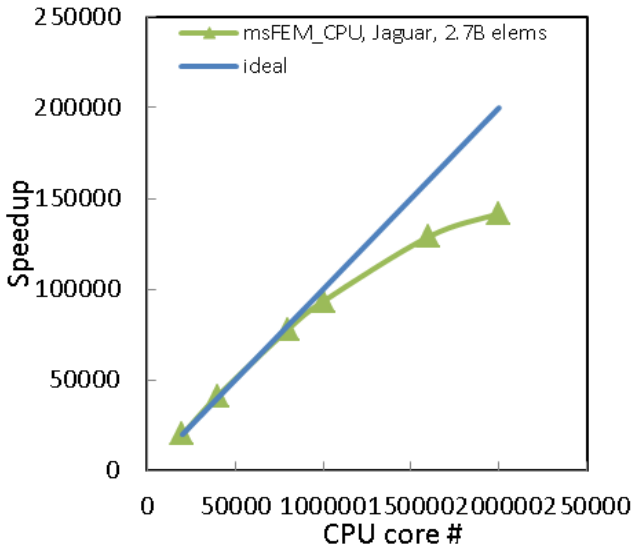


Fig. 7. Strong scalability of msFEM\_CPU(20K-200K CPU cores)

### 3.2 Tests of msFEM\_GPU

msFEM\_GPU has two different GPU implementations: full precision and mixed precision. The GPU implementations have been tested on Jaguar and Nebulae. The tests are not completed at the same time and hence the models used in the tests on the two machine are also not exactly the same. But the model size is approximately the same, around 100M tetrahedral elements. Test results are listed in Table 3 and plotted in Fig. 8. The test on Jaguar is done with Jaguar's GPU partition. Each compute node is equipped with one GPU card. Each compute node runs one MPI process (therefore the number of MPI processes on each node is only 1/16 of that of the flat MPI runs). One CPU core is assigned solely for data transfer and management and the rest 15 CPU cores are responsible for computation. The tests used up to 900 compute nodes or 900 GPU processors, which is almost the entirety (the whole partition contains 960 compute nodes with GPU) of the GPU partition of Jaguar.

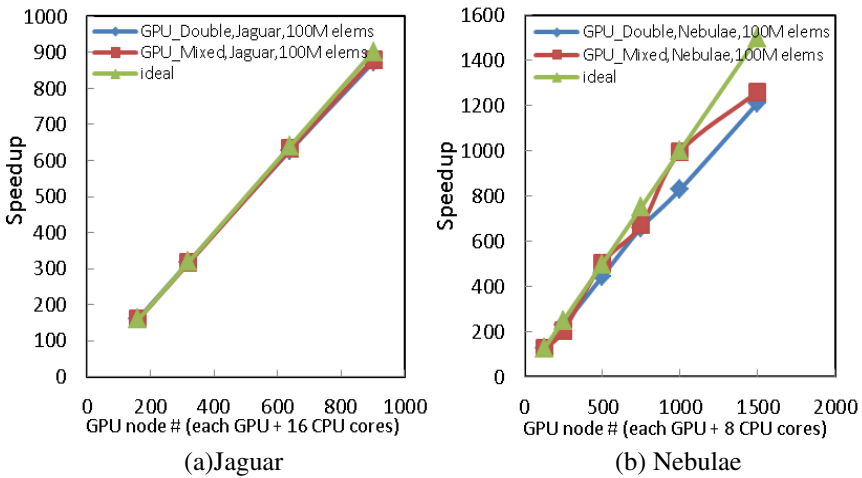
With regard to the time-to-solution, msFEM\_GPU\_double is 14 times faster than msFEM\_CPU, msFEM\_GPU\_mixed is 19 times faster than msFEM\_CPU. The mixed precision version gains extra 1.5 times speedup over the double precision implementation. The final solution of application obtained using the mixed precision algorithm maintains 10 effective numbers of decimal digits compared with that obtained by the double precision version[53][54].

It can be concluded from the above tests that:

1. Hybrid parallel computing, CPU-GPU co-processing, shows excellent parallel efficiency (90+%), due to significant increase in parallelization grain size on a GPU-accelerated compute node;
2. On the same hardware and the same accuracy of application solution, the mixed precision algorithm offers extra 1.5 times speedup with no significant implementational cost [53][54].

**Table 3.** Strong scalability tests of msFEM\_GPU codes on Jaguar

| # of MPI processes | Time(second/step) |             | Parallel efficiency |             | Speedup      |             | Speedup of mixed/double |
|--------------------|-------------------|-------------|---------------------|-------------|--------------|-------------|-------------------------|
|                    | Double prec.      | Mixed prec. | Double prec.        | Mixed prec. | Double prec. | Mixed prec. |                         |
| 160                | 26.82             | 18.00       | 1.00                | 1.00        | 1            | 1.00        | 1.49                    |
| 320                | 13.45             | 9.09        | 1.00                | 0.99        | 1.99         | 1.98        | 1.48                    |
| 640                | 6.82              | 4.55        | 0.98                | 0.99        | 3.93         | 3.96        | 1.50                    |
| 902                | 4.91              | 3.27        | 0.97                | 0.98        | 5.46         | 5.50        | 1.50                    |



**Fig. 8.** Strong scalability tests of msFEM\_GPU codes

### 3.3 Comparisons and Discussions

The tests in 3.1 and 3.2 are based on the same application algorithm but the different parallelization schemes. So a comparison between the results obtained by msFEM\_CPU and msFEM\_GPUs well reveals the difference in the parallelization model used. The GPU version of msFEM uses a MPI+X programming model (where X is the openMP+CUDA) while the CPU version is a flat MPI one. Clearly the MPI+X model shows much better scalability than the flat MPI; no matter on Jaguar or Nebulae (Fig. 8) the two ways of parallelization shows critically different scalability—the MPI+X is always much better than the flat MPI for the given approximately same size of mesh model. Clearly, the good scalability is attributed to the remarkably increased parallelization grain size on a GPU accelerated compute node.

## 4 Concluding Remarks

The scalability tests of the multi-scale code “msFEM” on Jaguar and Nebulae, two petaflops computers, show msFEM is highly scalable. msFEM\_CPU is scalable from

20K up to 200K CPU cores and delivers 71% parallel efficiency at the scale of 200K CPU cores; its GPU implementations achieve linear speedup with 90+% parallel efficiency at the scale of 1500 GPU nodes. On the same hardware and the same solution of application, the mixed precision algorithm offers extra 1.5 time speedup over the double precision implementation, with no significant implementational cost. The implementations of a CPU-only version and a CPU+GPU co-processing version for the same numerical method and the same physical model enable a direct comparison between the two programming models, the flat MPI and the MPI+X. It is clearly shown that the MPI+X is much better scalable due to the increasing parallelization grain size affordable on a GPU accelerated compute node. This observation motivates a further discussion on the scalability of the flat MPI model. When moving from today's petaflops computers to tomorrow's exascale systems, we will expect a dramatic increase in parallelism. A billion ways of parallelism in conjunction with runtime imbalance of millions of compute nodes [1], the flat MPI model will not scale down to use within future CPUs with hundreds or thousands of lightweight cores, in particular for those bulk synchronization intensive applications—for example implicit methods. This bulk synchronization operation resembles to frequently stop the whole population of China for one single thing, which is obviously not practical. Hence, a natural idea is to structure the flatten world of the flat MPI model into a hierarchy of several layers to reduce and finally to reduce a single-dimensional synchronization into a multiple level of synchronization. Fortunately, the GPU-accelerated compute node is already on the path, leading to a hierarchy of parallelism. As such, domain applications should also be structured in the similar way to match with the hardware evolution trend.

In summary, the MPI+X hybrid parallel model will be much better scalable as largely increasing parallelism grain size affordable by an individual compute node. The increased grain size reduces the number of MPI processes and hence reduces the volume/expense of bulk synchronization. With this reason, we should expect a shift in both computer hardware architecture and domain application algorithm framework from a single-dimensional, flatten world of parallelism into multiple level of hierarchy of parallelism.

**Acknowledgements:** The work was financially supported by “100 Talent Program” of Chinese Academy of Sciences and National Foundation of Sciences of China (grand numbers: 11072241, 11111140020, 91130026). This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725, under the Director's Discretion (DD) Project “MAT028” from 2010-2012. This research used resources of China National Center of HPC at Shenzhen.

## References:

1. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, Office of Science, DOE (2010)
2. <http://www.top500.org>

3. Sankaran, R.: Porting S3D turbulent combustion software to accelerator based systems. Titan Summit. August 15-17, JICS Auditorium, Building 5100, ORNL, USA (2011)
4. Archibald, R.: Progress Towards Accelerating CAM-SE on Hybrid Multi-Core Systems. Titan Summit. August 15-17, JICS Auditorium, Building 5100, ORNL, USA (2011)
5. Joubert, W.: Porting the Denovo Radiation Transport Code to Titan: Lessons Learned. Titan Summit. August 15-17, JICS Auditorium, Building 5100, ORNL, USA (2011)
6. Tharrington, A.: LAMMPS: Code Transformations in preparing for Titan. Titan Summit. August 15-17, JICS Auditorium, Building 5100, ORNL, USA (2011)
7. <http://ees.lanl.gov/pflotran/>
8. Eisenbach, M.: Preparing WL-LSMS for First Principles Thermodynamics Calculations on Accelerator and Multicore Architectures. Titan Summit. August 15-17, JICS Auditorium, Building 5100, ORNL, USA (2011)
9. Olson, G.B.: Designing a new material world. *Science* 288(5468), 993–998 (2000)
10. Olson, G.B.: Computational design of hierarchically structured materials. *Science* 277(5330), 1237–1242 (1997)
11. McVeigh, C., Liu, W.K.: Multiresolution continuum modeling of micro-void assisted dynamic adiabatic shear band propagation. *Journal of the Mechanics and Physics of Solid* 58(2), 187–205 (2010)
12. McVeigh, C., Vernerey, F., Liu, W.K., Brinson, C.: Multiresolution analysis for material design. *Computer Methods in Applied Mechanics and Engineering* 195, 5053–5076 (2006)
13. McVeigh, C., Vernerey, F.J., Liu, W.K., Moran, B., Olson, G.B.: An Interactive microvoid shear localization mechanism in high strength steels. *Journal of the Mechanics and Physics of Solids* 55(2), 224–225 (2007)
14. McVeigh, C.: Ph.D. thesis, Northwestern University (2007)
15. McVeigh, C., Liu, W.K.: Linking microstructure and properties through a predictive multiresolution continuum. *Computer Methods in Applied Mechanics and Engineering* 197, 3268–3290 (2008)
16. McVeigh, C., Liu, W.K.: Multiresolution modeling of ductile reinforced brittle composites. *Journal of the Mechanics and Physics of Solids* 57, 244–267 (2009)
17. Tian, R., Moran, B., Liu, W.K., Olson, G.B.: Multiscale fracture simulator. Dynamic Microstructure Design Consortium (ONR Contract: N00014-05-C-0241) Base Final Report (2008)
18. Tian, R., Liu, W.K., Chan, S., Olson, G.B., Tang, S., Wang, J.S., Jou, H.J., Gong, J.D., Moran, B.: Linking Microstructures to Fracture Toughness—predictive 3D process zone simulations. The D 3-D Annual PI Review, Evanston, IL, March 23-25 (2009)
19. Tian, R., Chan, S., Tang, S., Kopacz, A.M., Wang, J.-S., Jou, H.-J., Siad, L., Lindgren, L.-E., Olson, G., Liu, W.K.: A multi-resolution continuum simulation of the ductile fracture process. *Journal of the Mechanics and Physics of Solids* 58(10), 1681–1700 (2010)
20. <http://www.olcf.ornl.gov/event/cray-technical-workshop-on-xk6-programming/>
21. Aifantis, E.C.: On the role of gradients in the localization of deformation and fracture. *International Journal of Engineering Science* 30(10), 1279–1299 (1992)
22. Hill, R.: Elastic properties of reinforced solids: some theoretical principles. *Journal of the Mechanics and Physics of Solids* 11(5), 357–372 (1963)
23. Hill, R.: On constitutive macro-variables for heterogeneous solids at finite strain. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 326(1565), 131–147 (1972)
24. Tian, R., Yagawa, G.: Generalized node and high-performance elements. *International Journal for Numerical Methods in Engineering* 64, 2039–2071 (2005)

25. Tian, R., Yagawa, G., Terasaka, H.: Linear dependence problems of partition of unity based generalized FEMs. *Computer Methods in Applied Mechanics and Engineering* 195, 4768–4782 (2006)
26. Tian, R.: A PU-based 4-node quadratic tetrahedron and linear dependence elimination in three dimensions. *International Journal of Computational Methods* 3, 545–562 (2006)
27. Tian, R., Matsubara, H., Yagawa, G.: Advanced 4-node tetrahedrons. *International Journal for Numerical Methods in Engineering* 68, 1209–1231 (2006)
28. Tian, R., Yagawa, G.: Allman’s triangle, rotational dof and partition of unity. *International Journal for Numerical Methods in Engineering* 69, 837–858 (2006)
29. <http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>
30. Wilkinson, J.H.: *Rounding Errors in Algebraic Processes*. Prentice-Hall (1963)
31. Moler, C.B.: Iterative refinement in floating point. *J. ACM* 14(2), 316–321 (1967)
32. Jankowski, M., Woniakowski, H.: Iterative refinement implies numerical stability. *Journal BIT Numerical Mathematics* 17(3), 303–311 (1977)
33. Higham, N.J.: *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, Philadelphia (2002)
34. Demmel, J.W.: *Applied Numerical Linear Algebra*. SIAM Press (1997)
35. Demmel, J., Hida, Y., Kahan, W., Li, X.S., Mukherjee, S., Riedy, E.J.: Error bounds from extra precise iterative refinement. Technical Report No. UCB/CSD-04-1344, LAPACK Working Note 165 (February 2005)
36. Langou, J., Langou, J., Luszczek, P., Kurzak, J., Buttari, A., Dongarra, J.: Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In: *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (2006)
37. Kurzak, J., Dongarra, J.: Implementation of mixed precision in solving systems of linear equations on the Cell processor. *Concurrency and Computation: Practice and Experience* 19(10), 1371–1385 (2007)
38. Buttari, A., Dongarra, J., Langou, J., Langou, J., Luszczek, P., Kurzak, J.: Mixed precision iterative refinement techniques for the solution of dense linear systems. *Int. J. High Perform. Comput. Appl.* 21, 457–466 (2007)
39. Buttari, A., Dongarra, J., Kurzak, J., Luszczek, P., Tomov, S.: Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy. *ACM Transactions on Mathematical Software (TOMS)* 34(4) (2008)
40. Taiji, M., Narumi, T., Ohno, Y., Futatsugi, N., Suenaga, A., Takada, N., Konagaya, A.: Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations. In: *Proc. Supercomputing* (2003)
41. Göddeke, D., Strzodka, R., Turek, S.: Accelerating double precision FEM simulations with GPUs. In: *Proceedings of ASIM 2005 - 18th Symposium on Simulation Technique* (2005)
42. Strzodka, R., Göddeke, D.: Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In: *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2006)*, pp. 259–268 (2006)
43. Strzodka, R., Göddeke, D.: Mixed precision methods for convergent iterative schemes. In: *Proceedings of the 2006 Workshop on Edge Computing Using New Commodity Architectures*, p. D–59–60 (2006)
44. Göddeke, D., Strzodka, R., Turek, S.: Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, Special Issue: Applied Parallel Computing 22(4), 221–256 (2007)

45. Gddeke, D., Strzodka, R.: Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations (part 2: Double precision GPUs). Technical report, Technical University Dortmund (2008)
46. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J.W., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. SIAM, <http://www.netlib.org/lapack/>
47. Li, X.S., Demmel, J.W., Bailey, D.H., Henry, G., Hida, Y., Iskandar, J., Kahan, W., Kang, S.Y., Kapur, A., Martin, M.C., Thompson, B.J., Tung, T., Yoo, D.J.: Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software (TOMS)* 28(2) (2002)
48. Gddeke, D., Strzodka, R., Turek, S.: Performance and accuracy of hardware-oriented native-,emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, Special Issue: Applied Parallel Computing 22(4), 221–256 (2007)
49. Gddeke, D., Wobker, H., Strzodka, R., Mohd-Yusof, J., McCormick, P., Turek, S.: Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU. Accepted for Publication in the *International Journal of Computational Science and Engineering* (2008)
50. Strzodka, R., Gddeke, D.: Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In: *FCCM 2006: Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 259–270 (2006)
51. Kurzak, J., Dongarra, J.J.: Implementation of mixed precision in solving systems of linear equations on the CELL processor. *Concurrency and Computation: Practice and Experience* 19(10), 1371–1385 (2007)
52. Tian, R.: Co-design thinking towards exascale computing. *Information Technology Letter* 70(3), 50–63 (2012)
53. Liu, J., Wang, C., Ren, J., Tian, R.: A mixed precision explicit finite element algorithm on heterogeneous architecture and its CUDA implementation. *Computer Science* 39(6), 293–296 (2012)
54. Liu, J.: A mixed precision GPU acceleration algorithm and its application to FEM. MS thesis of Graduate School of Chinese Academy of Sciences (2011)

# Author Index

- Chang, Libo 66  
Chen, Hui-Xing 26  
Cui, Lizhen 75
- Deng, Junyong 66  
Du, Huimin 66  
Du, Zhenlong 89
- Feng, Chunsheng 1
- Han, Jungang 66  
He, Wei 75  
Hou, Ji 40  
Huang, Guangxin 66
- Jiang, Lin 66  
Jiang, Shengyi 55  
Jiao, Ge 132  
Jun-Min, Wu 12
- Kang, Lixia 110
- Li, Kenli 26, 118, 132  
Li, Lang 132  
Li, Tao 66  
Li, Xiaoli 89  
Liu, Hui 132  
Liu, Yongzhong 110
- Qin, Yunchuan 118
- Ren, Jiangyong 151  
Ren, Ju 99  
Ren, Yuxing 110  
Ruan, Li 140
- Shen, Kangkang 89  
Shi, Lin 26  
Shu, Shi 1  
Su, Huayou 99  
Sun, Yikai 140
- Tang, Yazhe 110  
Tian, Rong 151
- Wang, ChaoWei 151  
Wang, Yi 132  
Wang, Yingrui 151  
Wen, Mei 99  
Wu, Meiling 55  
Wu, Xing 40
- Xia, Mingxing 110  
Xiao, Limin 140  
Xiao, Lingzhi 66  
Xiao, Qi 118  
Xiao-Dong, Zhu 12  
Xiao-Yu, Zhao 12  
Xiu-Feng, Sui 12  
Xu, YuMing 132  
Xue, Yungang 99
- Yang, Xiaojian 89  
Ying-Qi, Jin 12  
Yue, Xiaoqiang 1
- Zhang, Chunyuan 99  
Zhang, Wu 40  
Zhu, Mingfa 140  
Zhuo, Shaojian 40