

PARA'04
WORKSHOP ON STATE-OF-THE-ART
IN SCIENTIFIC COMPUTING

June 20-23, 2004
Complementary Proceedings

The organizers and editors
Jack Dongarra
University of Tennessee and
Oak Ridge National Laboratory
and
Kaj Madsen and Jerzy Waśniewski
Technical University of Denmark

The main sponsor
Informatics and Mathematical Modelling
Technical University of Denmark

Other sponsors
HP, NAG, Comsol, Sun, UNI•C, Microsoft, and IBM.

Preface

Introduction

The PARA workshops in the past have been devoted to parallel computing methods in science and technology. There have been seven PARA meetings to date: PARA'94, PARA'95 and PARA'96 in Lyngby, Denmark, PARA'98 in Umeå, Sweden, PARA'2000 in Bergen, Norway, PARA'02 in Espoo, Finland, and PARA'04 again in Lyngby, Denmark. The first six meetings featured lectures in modern numerical algorithms, computer science, engineering, and industrial applications, all in the context of scientific parallel computing.

This meeting in the series, the PARA'04 Workshop with the title “State of the Art in Scientific Computing”, was held in Lyngby, Denmark, June 20-23, 2004. The PARA'04 Workshop was organized by Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory, and Kaj Madsen and Jerzy Waśniewski from the Technical University of Denmark. The emphasis here was shifted to High-Performance Computing (HPC). The ongoing development of ever more advanced computers provides the potential for solving increasingly difficult computational problems. However, given the complexity of modern computer architectures, the task of realizing this potential needs careful attention. For example, the failure to exploit a computer's memory hierarchy can degrade performance badly. A main concern of HPC is the development of software that optimizes the performance of a given computer.

The high cost of state-of-the-art computers can be prohibitive for many workplaces, especially if there is only an occasional need for HPC. A solution to this problem can be network computing, where remote computing facilities are exploited via the internet.

PARA'04 featured invited talks, contributed talks, minisymposia, and software and hardware vendors. The first day, June 20, was devoted to two parallel tutorials. The minisymposia and contributed talks during the main part of the Workshop, June 21-23, were scheduled in parallel sessions. All invited and contributed talks were noncommercial. The Workshop attracted 230 speakers from all over the world.

The PARA'06 Workshop with the title “State-of-the-Art in Scientific and Parallel Computing” will be held in Umeå (Sweden) on June 17-21, 2006.

Tutorials

Validated scientific computing using interval analysis was organized by *George F. Corliss* from Marquette University (USA). This tutorial gave an introduction to concepts and patterns of interval analysis. It was assumed that the participants had had a first course in scientific computation, including floating-point arithmetic, error analysis, automatic differentiation, Gaussian elimination, Newton's method, numerical optimization, and Runge-Kutta methods for ODE's. The tutorial included lectures, with examples in MATLAB and Sun's Fortran 95, and a set of supervised, hands-on exercises.

Automatic differentiation was organized by *Andrea Walther* from the Technical University of Dresden (Germany). This tutorial gave a detailed introduction to the chain rule based technique of automatic differentiation (AD) that provides first and higher derivatives without incurring truncation errors. Several examples illustrated the theoretical results. Some AD tools, selected as a reasonably representative sample, were tested in supervised, hands-on exercises.

Key speakers

Richard P. Brent, Oxford University Computing Laboratory (UK), *Fast and reliable random number generators for scientific computing*. Fast and reliable pseudo-random number generators are required for simulation and other applications in scientific computing. Richard outlined the requirements for good uniform random number generators, and described a class of generators having very fast vector/parallel implementations with excellent statistical properties.

Bernd Dammann and **Henrik Madsen**, the Technical University of Denmark (Denmark), *High Performance Computing and the importance of code tuning - some practical experiences from program tuning at the DTU HPC Center*. This talk gave a short overview of the High Performance Computer installation at the Technical University of Denmark (DTU), as well as a summary of some code tuning experiments. It is easy to reduce the run time of an application for a given problem by buying a computer with a faster CPU (higher clock frequency). However, very often the same or even better speed-up of the code can be achieved by analyzing and tuning the code - without the need to invest in new hardware.

Jack Dongarra, the University of Tennessee and Oak Ridge National Laboratory (USA), *High performance computing trends and Self Adapting Numerical Software (SANS) - effort*. In this talk Jack looked at how high performance computing has changed over the last 10 years and predicted future trends. In addition, he advocated the need for self adapting software.

Iain Duff, the Rutherford Appleton Laboratory (UK) and CERFACS (France), *Partitioning and parallelism in the solution of large sparse systems*. Iain first reviewed the various levels of parallelism that are available in the direct solution of large sparse linear systems. He also briefly considered iterative as well as direct methods in this study.

Fred Gustavson, the IBM T.J. Watson Research Center (USA), *Ideas for high performance linear algebra software*. In this talk Fred presented several ideas for the development of sequential and parallel HPC dense linear algebra software. The main results were obtained from the Algorithms and Architecture Approach.

Per Christian Hansen, the Technical University of Denmark (Denmark), *Large-scale methods in inverse problems*. Inverse problems arise in geophysics, tomography, image deblurring and many other areas where the goal is to compute interior or hidden information from exterior data. This talk presented a survey of numerical methods and paradigms suited for large-scale inverse problems.

Bo Kågström, the University of Umeå (Sweden), *Recursive blocked algorithms and hybrid data structures for dense matrix library software*. Matrix computations are both fundamental and ubiquitous in computational science and its vast application areas. Along with the development of more advanced computer systems with complex memory hierarchies, there is a continuing demand for new algorithms and library software that efficiently utilize and adapt to new architecture features.

John K. Reid, the Rutherford Appleton Laboratory (UK), *Fortran is getting more and more powerful*. There is much happening just now with respect to Fortran. The features of Fortran 2003 have been chosen and the standard is very near completion. John is the Convener of the ISO Fortran Committee.

Peter Sloot, the University of Amsterdam (The Netherlands), *Scientific computing in the Grid: A biophysical case study*. Workers at the University of Amsterdam

conducted computer simulation experiments in pre-operative planning of vascular reconstruction with a physician in the experimental loop. Peter showed new results from numerical simulations of blood flow with 3D cellular automata.

Zahari Zlatev, National Environmental Research Institute (Denmark), *Large-scale computations with the Unified Danish Eulerian Model*. The Unified Danish Eulerian Model (UNI-DEM) is a mathematical model for performing different comprehensive studies related to damaging effects from high pollution levels in Denmark and Europe. The model is described by a system of partial differential equations (PDEs).

Minisymposia

Interval methods, organized by *Luke Achenie*, University of Connecticut (USA), *Vladik Kreinovich*, University of Texas at El Paso (USA), and *Kaj Madsen*, Technical University of Denmark (Denmark). In many practical problems there is a need to (a) solve systems of equations and inequalities, and/or (b) optimize some performance measure. The results obtained by conventional algorithms are either local or cannot be guaranteed. Interval analysis provides guaranteed approximations of the set of all the actual solutions of the problem. This ensures that no solution is missed. There were 21 speakers in this minisymposium.

Trends in large scale computing, organized by *Scott B. Baden*, University of California at San Diego (USA). Software infrastructure for large scale computation often fails to realize the full potential afforded by technological advances, and the result is lost opportunities for making scientific discovery. This minisymposium examined two important issues in software infrastructure for large scale computation: achieving scalability, and optimization through specialization. There were 5 speakers in this minisymposium.

High performance linear algebra algorithms, organized by *Fred G. Gustavson*, IBM T.J. Watson Research Center (USA), and *Jerzy Waśniewski*, Technical University of Denmark (Denmark). The algorithms of Linpack and Eispack and later LAPACK and ScaLAPACK have stood the test of time in terms of robustness and accuracy. The focus of this minisymposium was on explaining high performance versions of these algorithms. There were 7 speakers in this minisymposium.

Substructuring, dimension reduction and applications, organized by *Zhaojun Bai*, University of California (USA) and *Rencang Li*, University of Kentucky USA. There are a variety of reasons to go for substructuring and dimension reduction in scientific computations and applications. Substructuring makes it possible to solve large and seemingly intractable computational problems by some kind of divide-and-conquer technique. It also offers a general methodology for parallelization. There were 12 speakers in this minisymposium.

Parallel processing in science and engineering, organized by *Adam W. Bojańczyk*, Cornell University (USA). This minisymposium concerned selected aspects of parallel and distributing computing as they arise in engineering. Both non-traditional applications as well as relevant software tools were presented. There were 9 speakers in this minisymposium.

Distributed computing: tools, paradigms and infrastructures, organized by *Beniamino Di Martino*, *Rocco Aversa*, Second University of Naples (Italy), and *Laurence Tianruo Yang*, Francis Xavier University (Canada). The Minisymposium presented re-

cent advances in distributed computing technology, methodology and tools. The presentations featured a variety of topics ranging from mobile and location-aware computing to skeletons and high-level parallel languages, from programming environments and tools for Grid applications' development and tuning, to distributed monitoring and security issues. There were 9 speakers in this minisymposium.

High-performance computing in earth and space science, organized by *Peter Messmer*, Tech-X Corporation at Boulder (USA). High-performance computing facilities enable simulations of physical phenomena with ever increasing fidelity and accuracy. The range of resolved scales in a single simulation, as well as the number of physical processes included, yield results that can be directly compared with observational data. There were 7 speakers in this minisymposium.

Advanced algorithms and software components for scientific computing, organized by *Padma Raghavan*, Pennsylvania State University (USA). This minisymposium concerned algorithms for sparse linear systems solution and function approximation and their implementation using advanced software architectures. Discussions emphasized the role of such techniques for improving the performance of long-running PDE-based simulations. There were 7 speakers in this minisymposium.

Software engineering and problem solving environments for scientific computing, organized by *José C. Cunha*, Universidade Nova de Lisboa (Portugal) and *Omer F. Rana*, Cardiff University (UK). The emergence of computational grids in the last few years provides new opportunities for the scientific community to undertake collaborative and multi-disciplinary research. The aim of this minisymposium was to bring together experts who have experience in developing software tools to support application scientists, and those who make use of these tools. There were 5 speakers in this minisymposium.

Runtime software techniques for enabling high-performance applications, organized by *Masha Sosonkina*, Iowa State University (USA). Parallel computing platforms are advancing rapidly, both in speed and size. However, often only a fraction of the peak hardware performance is achieved by high-performance scientific applications. One way to cope with the changeability of hardware is to start creating applications able to adapt themselves "on-the-fly". The talks of the minisymposium discussed this issue from both the application-centric and system-centric viewpoints. There were 6 speakers in this minisymposium.

Sparse direct linear solvers, organized by *Sivan Toledo*, Tel-Aviv University (Israel). The matrices of most of the systems of linear algebraic equations arising from scientific and engineering applications are sparse. This minisymposium dealt with some modern algorithms for sparse direct linear solvers. There were 12 speakers in this minisymposium.

Treatment of large scientific models, organized by *Krassimir Georgiev*, Bulgarian Academy of Science (Bulgaria) and *Zahari Zlatev*, National Environmental Research Institute (Denmark). The exploitation of new fast computers in the effort to avoid non-physical assumptions and, thus, to develop and run more reliable and robust large scientific models was the major topic of this minisymposium. There were 9 speakers in this minisymposium.

Performance evaluation and design of hardware-aware PDE solvers, organized by *Markus Kowarschik* and *Frank Hülsemann*, University of Erlangen-Nuremberg (Germany). In an ideal situation, all performance optimization of computationally intensive software would take place automatically, allowing the researchers to concentrate on the development of more efficient methods rather than having to worry about performance. However, for the time being, the need to identify and remove the performance bottlenecks of computationally intensive codes remains. As an example of a class of computationally intensive problems, this minisymposium concentrates on the numerical solution of PDEs. There were 7 speakers in this minisymposium.

Computationally expensive methods in statistics, organized by *Wolfgang Hartmann*, SAS Institute Inc. (USA) and *Paul Somerville*, University of Central Florida (USA). A two-dimensional data set with N observations (rows) and n variables (columns) and large scale data requires intensive computational work. Of course there may be even more dimensions of the data set. There were 5 speakers in this minisymposium.

Approaches or methods of security engineering (AMSE), organized by *Taihoon Kim* and *Ho Yeol Kwon*, Kangwon National University (Korea). Security engineering software is needed for reducing security holes. The talks presented a number of methods for designing such software. There were 16 speakers in this minisymposium.

Contributed talks

Some contributed talks were added to the minisymposium sessions. The rest were organized in the following independent sessions: two sessions of “Grid and network”, two sessions of “HPC applied to security problems”, two sessions of “Clusters and graphics”, one session of “HPC applied to cryptology”, one session of “ODEs, PDEs and automatic differentiation”, one session of “Computer tools”, and a special session of “Computer vendors”.

The Workshop Proceedings

The proceedings of the Para’04 Workshop are divided into two complementary books, this (ISBN: 87-643-0041-2 and ISSN: 1601-2321), and the following Springer book:

- J. Dongarra, K. Madsen and J. Waśniewski (Eds.)
 - ▶ Proceedings of the Para’04 Workshop on State-of-the-Art in Scientific Computing, Lyngby, Denmark, June, 2004.
 - ▶ Springer Lecture Notes in Computer Science, number 3732.

Acknowledgments

The organizers are indebted to the Para’04 sponsors, whose support was vital to the success of the workshop. The main sponsor was the Department of Informatics and Mathematical Modelling of the Technical University of Denmark. The other sponsors were:

- HP High Performance Computing USA,
- NAG Numerical Algorithms Group LTD UK,
- Comsol A/S Denmark (MATLAB distributor),
- Sun Microsystems Denmark,
- UNI•C Danish Computing Center Denmark,
- Microsoft Denmark, and

- IBM International Business Machines Denmark.

The organizers would like to thank Kirsten Probst for her excellent work as Para'04 secretary. Dorthe Thøgersen, Henrik Krogh and other staff of the conference also provided very valuable help.

We are very grateful to Professor Ho Yeol Kwon from the Kangwon National University, Electrical and Computer Engineering Department for taking many photos during the PARA'04 conference. These are available at the PARA'04 URL

<http://www.imm.dtu.dk/~jw/para04/>.

Thanks are also due to Vincent A. Barker for his kind assistance in the preparation of both the workshop and these proceedings.

The Para'04 conference ended on June 23 2004. The evening of June 23, the eve of St. Hans Day, is celebrated in Denmark by the lighting of bonfires. We are indebted to Arriva Denmark A/S for making available two boats for the Para'04 participants, from which we could see some of the bonfires and admire beautiful Copenhagen.

Finally, we would like to thank the Para'04 referees for their careful evaluation of the workshop papers.

Jack Dongarra
Kaj Madsen
Jerzy Waśniewski

Table of Contents

Masking Latency with Data Driven Program Variants	1
<i>Scott B. Baden</i>	
Dynamic Code Generation and Component Composition in C++ for Optimising Scientific Codes at Run-time	2
<i>Olav Beckmann and Paul H J Kelly</i>	
Communication Strategies for Parallel Cooperative Ant Colony Optimization on Clusters and Grids	3
<i>Siegfried Benkner, Karl F. Doerner, Richard F. Hartl, Guenter Kiechle and Maria Lucka</i>	
Fully Self Organized Public Key Management for Mobile Ad Hoc Network	13
<i>Daeseon Choi, Seunghun Jin, Hyunsoo Yoon</i>	
Real Time Intrusion Detection System for Malformed Packet Attacks	23
<i>Eun-Yeung Choi, Hyun-Sung Kim, Kee-Young Yoo</i>	
Predicting Protein-Protein Interactions in Parallel	24
<i>Yoojin Chung, Sang-Young Cho and Chul-Hwan Kim</i>	
On the Parallelization of the Lattice-Boltzmann Method	33
<i>Salvatore Filippone, Nicola Rossi, Gino Bella and Stefano Ubertini</i>	
Parallel and Distributed Techniques for Extracting Large Ontologies as a Resource in a Grid Environment	43
<i>Andrew Flahive, Mehul Bhatt, Carlo Wouters, Wenny Rahayu, David Taniar and Tharam Dillon</i>	
Adaptive Fuzzy Active Queue Management	51
<i>Mahdi Jalili-Kharaajoo, Mohammadreza Sadri and Farzad Habibipour Roudsari</i>	
Inversion and Division Architecture in Elliptic Curve Cryptography over $GF(2^n)$ (Not accepted yet)	59
<i>Jun-Cheol Jeon, Kyo-Min Ku, Kee-Young Yoo</i>	
Efficient On-the-fly Detection of First Races in Nested Parallel Programs	69
<i>Keum-Sook Ha, Yong-Kee Jun,, Kee-Young Yoo</i>	
New Architecture for Inversion and Division over $GF(2^m)$	79
<i>Kyeoung Ju Ha, Kyo Min Ku, and Kee Young Yoo</i>	

Solving Linear Systems on Cluster Computers with High Accuracy	83
<i>Carlos Amaral Hölbíg, Paulo Sérgio Morandi Jr., Bernardo Frederes Krämer Alcalde, Tiarajú Asmuz Diverio, Dalcidio Moraes Claudio</i>	
Finite Fields Multiplier based on Cellular Automata	91
<i>Hyun-Sung Kim, and Il-Soo Jeon</i>	
Efficient Systolic Architecture for Modular Multiplication over $GF(2^m)$	98
<i>Hyun-Sung Kim and Sung-Woon Lee</i>	
Analyzing the Safety Problem in Security Systems using SPR Tool	105
<i>Il-Gon Kim, Jin-Young Choi, Peter D. Zegzhda, Maxim O. Kalinin, Dmitry P. Zegzhda, In-Hye Kang, Pil-Yong Kang and Wan S. Yi</i>	
A CBD-based SSL Component Model	115
<i>Young-Gab Kim, Lee-Sub Lee, Dongwon Jeong, Young-Shil Kim, Doo-Kwon Baik</i>	
Exponentiation over $GF(2^m)$ For Public Key Crypto System using Cellular Automata	123
<i>Kyo Min Ku, Kyeong Ju Ha, and Kee Young Yoo</i>	
Area Efficient Multiplier based on LFSR Architecture	129
<i>Jin-Ho Lee, and Hyun-Sung Kim</i>	
Efficient Authentication and Key Agreement for Client-Server Environment	136
<i>Sung-Woon Lee, Hyun-Sung Kim, and Kee-Young Yoo</i>	
New Efficient Digit-Serial Systolic AB^2 Multiplier & Divider in $GF(2^m)$	143
<i>Won-Ho Lee, Kee-Young Yoo</i>	
Parallel Co-Processor for Ultra-fast Line Drawing	153
<i>Pere Marès, Antonio B. Martínez and Joan Aranda</i>	
The Design Patterns of Performance-Decision Factors in High-Speed NIDS	163
<i>Jongwoon Park, Keewan Hong, Kiyoong Hong, Dongkyoo Kim, Bongnam No</i>	
Scalable Race Visualization for Debugging Message-Passing Programs	173
<i>Mi-Young Park, So-Hee Park, Su-Yun Bae, and Yong-Kee Jun,</i>	
Hierarchical Structures for Multi-Resolution Visualization of AMR Data	183
<i>Sanghun Park</i>	
The Development of Domain Specific Languages From Scientific Libraries	193
<i>Daniel Quinlan</i>	
A method to Derive the Cache Performance of Irregular Applications on machines with Direct Mapped Caches (Before referee)	194
<i>Carsten Scholtes</i>	

<i>Preface</i>	IX
Interfacing C++ member functions with C libraries <i>Kurt Vanmechelen and Jan Broeckhove</i>	204
Compilation Techniques for a Chip-Multiprocessor with Two Execution Modes <i>Chao-Chin Wu</i>	211
Explicit Formulas and Library of Images of Electromagnetic Fields for Anisotropic Materials <i>Valery Yakhno, Tatyana Yakhno and Mustafa Kasap</i>	221
Author Index	231

Masking Latency with Data Driven Program Variants

Scott B. Baden

University of California, San Diego
Department of Computer Science and Engineering
9500 Gilman Drive, MC 0114
La Jolla, CA 92093-0114 USA
baden@cs.ucsd.edu

Abstract. Application performance is sensitive to technological change, in particular to effects that have, over time, raised the cost of communication relative to computation. A general approach for tolerating the cost of communication is elusive. The difficulty is that the programmer must partition and schedule computations in order to mask latencies, but the exact strategy and policy depends on the application, the system, and even dynamic operating conditions. I present the notion of a *canonical program variant* that relies on flexible scheduling of coarse grain data flow tasks to reorder computations dynamically. This approach is similar to classic data flow and to instruction level parallelism. It eliminates the need to hard code the strategy for overlapping communication with computation, enabling the application to respond dynamically to data dependent and environmental conditions.

Acknowledgments

This work was partially supported under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

Dynamic Code Generation and Component Composition in C++ for Optimising Scientific Codes at Run-time

Olav Beckmann¹ and Paul H J Kelly¹

Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2AZ, United Kingdom
www.doc.ic.ac.uk/~ob3, phjk

Abstract. The TaskGraph Library is a C++ library for dynamic code generation and component composition, which combines specialisation with dependence analysis and restructuring optimisation. A TaskGraph represents a fragment of code which is constructed and manipulated at run-time, then compiled, dynamically linked and executed. The TaskGraph Library is implemented purely in C++, using macros and operator overloading to define a simplified, C-like sub-language that is used for constructing TaskGraphs. The internal representation for generated code is SUIF-1, which facilitates using a range of analysis and restructuring passes.

We view the TaskGraph library as a research tool that can support effective large-scale computation in two distinct ways: dynamic component modification, leading to code that can tune itself to a particular platform at run-time, and optimisation of dynamic component composition, where the internal structure of software components is adapted at run-time to their calling context.

Communication Strategies for Parallel Cooperative Ant Colony Optimization on Clusters and Grids^{*}

Siegfried Benkner¹, Karl F. Doerner², Richard F. Hartl², Guenter Kiechle³ and Maria Lucka¹

¹ Institute for Software Science, University of Vienna
Nordbergstrasse 15, A-1090 Vienna, Austria
{sigi, lucka}@par.univie.ac.at

² Institute for Management Science
Bruenner Strasse 72, A-1210 Vienna, Austria
{karl.doerner, richard.hartl}@univie.ac.at

³ Salzburg Research Forschungsgesellschaft mbH
Jakob Haringer Strasse 5/II, A-5020 Salzburg
guenter.kiechle@salzburgresearch.at

Abstract. In this paper we study different parallel variants of Ant Colony Optimization (ACO) for solving the Vehicle Routing Problem. We propose a new parallelization strategy which is based on a cooperation of multiple ant colonies and which combines fine-grained with coarse-grained parallelism. Moreover we outline the realization of a Grid service for ACO using the Vienna Grid Environment.

1 Introduction

The Vehicle Routing Problem (VRP) involves the construction of a set of vehicle tours starting and ending at a single depot and satisfying the demands of a set of customers, where each customer is served by exactly one vehicle and neither vehicle capacities nor maximum tour lengths are violated. Therefore no efficient exact solution methods are available, and the existing solution approaches are of heuristic nature. In this article we focus on solving the VRP using different parallel versions of Ant Colony Optimization (ACO). Based on the observation of real ant's foraging behavior ACO was developed as a graph-based, iterative, constructive meta-heuristic by Dorigo et al. [11]. The main idea of ACO is that a population of computational ants repeatedly builds and improves solutions to a given instance of a combinatorial optimization problem. From one generation to the next a joint memory is updated that guides the work of the successive populations. The memory update is based on the solutions found by the ants and more or less biased by their associated quality.

Recently some possible parallelization strategies for ACO have been proposed, which can be classified into *fine-grained* and *coarse-grained* strategies [12]. In fine-grained parallelization strategies usually several artificial ants of a colony are assigned to each

^{*} This work was supported by the Special Research Program SFB F011 "AURORA" of the Austrian Science Fund FWF.

processor and therefore frequent information exchange between the small sub-colonies of ants (i.e. an information exchange between the processors) takes place [9, 14, 22]. Coarse-grained parallelization schemes run several colonies in parallel. This strategy is also referred to as *multi colony approach*. The information exchange among colonies is done at certain intervals or numbers of iterations [3, 15]. The important questions in implementing the multi colony approach are when, which and how information should be exchanged among the colonies. Stuetzle [21] studied the effect on solution quality when applying independent runs of the algorithm without communication in comparison to one longer run of the algorithm.

In our work we consider both fine-grained and coarse-grained parallelization strategies, as well as a combination of these two strategies. To our best knowledge this is the first work which presents speed-up and efficiency for a fine-grained, coarse-grained and mixed parallelization strategy for the Savings based ACO algorithm. First preliminary results for the fine-grained strategy of one problem class were already shown in our previous paper [10].

We evaluate the effectiveness of the different parallelization strategies on PC clusters and outline the realization of a Grid service for Ant Colony Optimization.

2 Savings based ACO algorithms for the VRP

The Savings based ACO algorithm published in [19] and repeated here mainly consists of the iteration of three steps: (1) generation of solutions by ants according to private and pheromone information; (2) application of a local search to the ants' solutions, and (3) update of the pheromone information.

Solutions are constructed based on the well known Savings Algorithm due to Clarke and Wright [7]. In this algorithm the initial solution consists of the assignment of each customer to a separate tour. After that for each pair of customers i and j the following savings values are calculated:

$$s_{ij} = d_{i0} + d_{0j} - d_{ij}, \quad (2.1)$$

where d_{ij} denotes the distance between locations i and j , the index 0 denotes the depot, and s_{ij} represent the savings of combining two customers i and j on one tour contrary to serving them on two different tours. In the iterative phase, customers or partial tours are combined by sequentially choosing feasible entries from the list of saving values. A combination is infeasible if it violates either the capacity or the tour length constraints. The decision making about combining customers is based on a probabilistic rule taking into account both savings values and the pheromone information. Let τ_{ij} denote the pheromone concentration on the arc connecting customers i and j telling us how good the combination of these two customers i and j was in previous iterations. In each decision step of an ant, we consider the k best combinations still available, where k is a parameter of the algorithm which we will refer to as 'neighborhood' below. Let Ω_k denote the set of k neighbors, i.e. the k feasible combinations (i, j) yielding the largest savings, considered in a given decision step, then the decision rule is given by

equation (2.2).

$$\mathcal{P}_{ij} = \begin{cases} \frac{s_{ij}^\beta \tau_{ij}^\alpha}{\sum_{(h,l) \in \Omega_k} s_{hl}^\beta \tau_{hl}^\alpha} & \text{if } (i, j) \in \Omega_k \\ 0 & \text{otherwise.} \end{cases} \quad (2.2)$$

In (2.2), \mathcal{P}_{ij} is the probability of choosing to combine customers i and j on one tour, while α and β bias the relative influence of the pheromone trails and the savings values, respectively. This algorithm results in a (sub-)optimal set of tours through all customers, once no more feasible savings values are available.

The used pheromone update rule was proposed in [2] and its pheromone management centers around two concepts borrowed from Genetic Algorithms, namely ranking and elitism to deal with the trade-off between exploration and exploitation. In [19] this paradigm was used for solving the VRP. Thus, we will just briefly depict the pheromone update scheme here. Let $0 \leq \rho \leq 1$ be the trail persistence and E the number of elitists. Then, the pheromone update scheme can formally be written as

$$\tau_{ij} := \rho \tau_{ij} + \sum_{r=1}^{E-1} \Delta \tau_{ij}^r + E \Delta \tau_{ij}^* \quad (2.3)$$

First, the best solution found by the ants up to the current iteration is updated as if E ants had traversed it. The amount of pheromone laid by the elitists is $\Delta \tau_{ij}^* = 1/L^*$ if (ij) belongs to the best solution so far, 0 otherwise, where L^* is the objective value of the best solution found so far. Second, the $E - 1$ best ants of the current iteration are allowed to lay pheromone on the arcs they traversed. The quantity laid by these ants depends on their rank r as well as their solution quality L^r , such that the r -th best ant lays $\Delta \tau_{ij}^r = (E - r)/L^r$ on the arcs they traverse. Arcs belonging to neither of those solutions just face a pheromone decay at the rate $(1 - \rho)$, which constitutes the trail evaporation.

A solution obtained by the above mentioned procedure can then be subjected to a local search in order to ensure local optimality. In our algorithms we sequentially apply the *swap* neighborhood [17] between tours to improve the clustering and the 2-opt algorithm [8] within tours to improve the routing.

3 Granularity of Parallelization

Our goal in parallelization of the ACO algorithm is on the one hand to speed up its execution and on the other hand to improve the solution quality by exploiting the possibilities provided by parallel architectures. We have considered three different parallelization strategies: a coarse-grained, a fine-grained, and a mixed strategy which exploits both coarse-grained and fine-grained parallelism.

The coarse-grained parallelization strategy is based on a multi-colony approach where several colonies of ants cooperate in finding good solutions. We use a homogeneous approach where the different ant colonies have the same behavior. The information exchange between the colonies is based on the exchange of single solutions.

With the fine-grained parallelization strategy a colony of ants is partitioned into multiple subcolonies, depending on the number of processors available. Every subcolony uses the same pheromone matrix. After all ants in a subcolony have found their solutions, a local best solution is established. From these values the global best solution and the E best solutions for the whole colony are found. The pheromone matrix for every colony is updated separately.

The mixed parallelization strategy attempts to combine the advantages of the fine-grained and coarse-grained strategies. With this strategy the MPI processes are organized into two hierarchical levels, where on the higher level several colonies of ants exploit coarse-grained parallelism by pursuing a multi-colony approach, while on the lower level each of these colonies is split into multiple subcolonies in order to exploit fine-grained parallelism. For example, if 8 MPI processes are used and 4 colonies of ants are processed in parallel, each colony is partitioned into two subcolonies which again are processed in parallel.

3.1 Fine-grained Parallelization

Our performance analysis of the fine-grained approach is based on the larger instances of the 14 classic VRP benchmarks [6]. These instances range from 75 to 199 customers and comprise both problems where all customers are randomly located in the plane and problems where customer located in several clusters. Furthermore, some instances are just capacity constrained while others also feature restrictions on the maximum tour length. The experiments with the parallelized version were run on a Beowulf cluster with 16 compute nodes, each equipped with four 700 MHz Pentium III Xeon processors and connected via Myrinet. We report detailed results on speedup with different numbers of processors.

In Figure 1 we present average results on the speedup over 4 benchmark problem instances ranging from 75 to 199 customers. The experiments clearly show satisfying speedups up to 32 processors. The results are averaged over 5 runs for each problem instance.

For example for the problems with 199 customers, on 8 processors a sixfold speed up with an efficiency of more than 70% and on 32 processors a speedup of 12 with an efficiency of approximately 40% was obtained.

The result is different when we consider the speedup of the 75 customer problem instance and the usage of 16 processors. Here we have a maximum speedup of 4.88 (with an efficiency of 60%), which is smaller than the speedup on the larger problems. The reason for the reduced speedup on the smaller problem is that the required runtime in constructing a solution is also smaller than for the larger problems and therefore the pheromone update which requires communication consumes more runtime compared to the total amount of used runtime. This speedup is reached by using 4 processors per node and two nodes. We have different speedup when we use a different assignment of our 8 processors. The speedup is reduced to 4.26 when we execute the program on 4 nodes and on each node only 2 of the processors are used - when we use 8 nodes (on each node 1 processor) then the speedup is reduced to 2.61.

In comparison to that we consider the speedup of the larger problem instances with 199 customers. Here the effect is not so drastic - it is even a contrary effect. When we

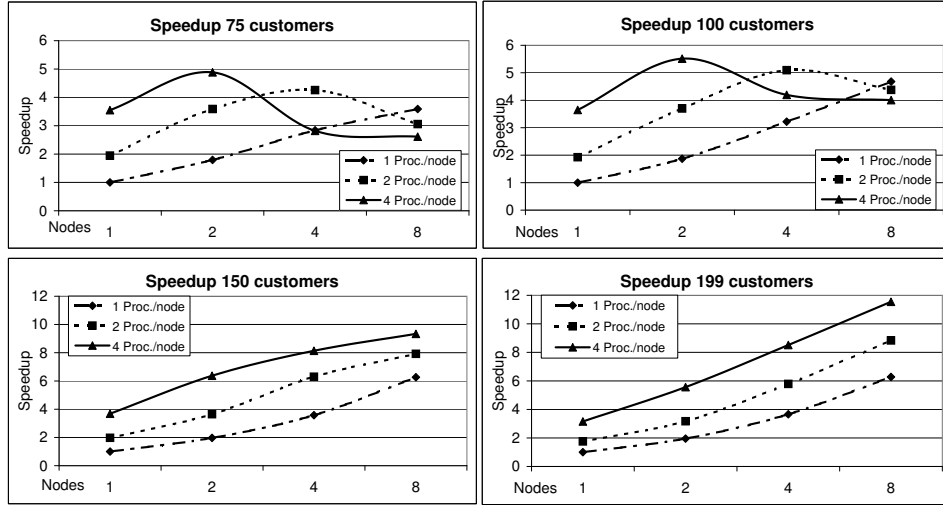


Fig. 3.1. Speedup - Fine Grained Approach

use 4 processors per node and two nodes we have a speedup of 5.56 - this speedup increases to 5.79 by using 4 nodes (2 processors an each node). We have even a further increase in speedup when we use 8 nodes with one processor on each node to 6.28.

The reason for the different behavior in speedup of the problems with the different problem sizes lies in cache effects. For the smaller problem we have a gain in speedup when we use all the available processors on one node - the communication is faster and the memory requirements are not so large as for the problems with more customers where the pheromone matrix is larger.

Therefore, for smaller problem instances it is recommendable to use all the processors on one node. For the problems with 150 customers the speedup for 8 processors on varying number of nodes is almost the same. For larger problems it is favorable to use fewer processors per node in order to get a better speedup on our hardware architecture.

On our results we can also see that for the smaller instances we get a reduction in the speedup. When too many processors are used the speedup reduces from 4.88 to 2.61 with the number of processors increasing from 8 to 32.

3.2 Coarse-grained and Mixed Parallelization

In Figure 2 performance results of our mixed parallelization strategy using 4, 8 and 16 processors for 2 colonies are shown both in terms of execution times and solution quality, where the exchange of solutions between the colonies was performed every *ex* iterations (denoted on the x-axis). The solution quality is denoted on the y-axis. Note that a smaller number represents a better solution quality. In the variant *gb* only the global best solution is exchanged, whereas in the variant *gb + el* also the solutions of the elitists in the population with the global best solution are broadcast to the other colony. Moreover, we have also implemented the two variants where the converged

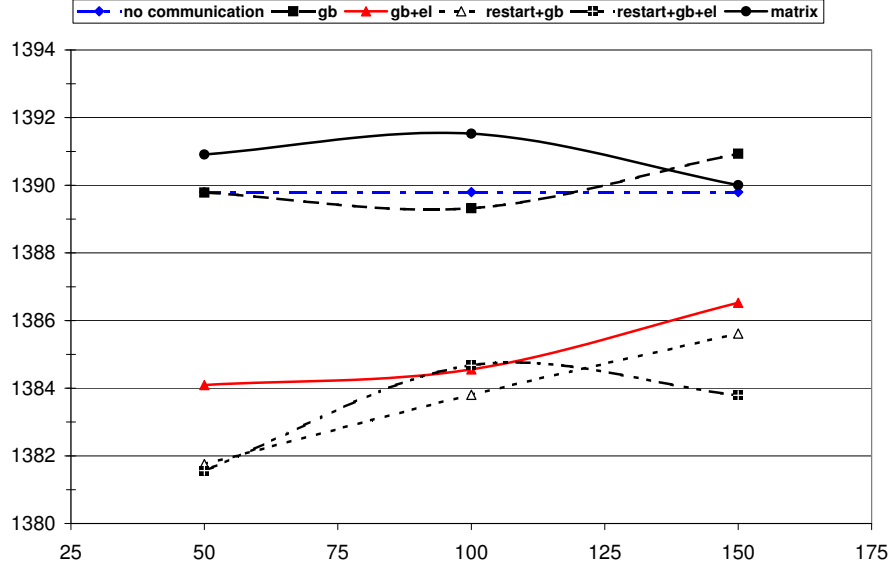


Fig. 3.2. Computational Results - Mixed Approach for the Problem with 199 Customers

pheromone matrices are reset and only the globalbest and elitist solutions are updated in the new initialized matrix. Another version of our algorithm exchanges the whole pheromone matrix of the population where the globalbest solution was found. We apply the algorithm for two times the number of customers iterations.

<i>variant</i>	<i>runtime</i>	<i>globalbest solution</i>
<i>no communication</i>	120.13	1389.80
<i>gb</i>	147.47	1389.76
<i>gb + el</i>	149.27	1384.09
<i>restart + gb</i>	206.29	1381.74
<i>restart + gb + el</i>	199.44	1381.56
<i>matrix</i>	201.18	1390.26

Table 1. Runtime of Cooperative Colonies for the Problem with 199 Customers

The average solution quality of two independent populations is provided as a yardstick. In this case no communication between the two populations occurs and the better result of the two populations is reported (variant *no communication*). By using pheromone information exchange we can easily improve the solution quality. The so-

lution quality without communication is 1389.8. We can increase the solution quality to 1381.56 if we exchange the global best and the elite solutions every 50 iterations between the two colonies and reset the pheromone matrix. By communicating only the global best solution we achieve almost the same solution quality. The runtime is increased by resetting the pheromone matrix from about 150 seconds to over 200 seconds. (see Table 1). A good tradeoff between solution quality and runtime can be the strategy with no reset of the matrix and the communication of the global best and elite solutions (sol. qual. 1384.09, runtime 149.27 sec.). The communication of the whole pheromone matrix leads to a decrease in solution quality as well as worse runtime behavior.

4 Grid Services for Ant Colony Optimization

In the following we outline the provision of our parallel ant colony optimization codes as Grid Services using the Vienna Grid Environment (VGE) [1, 23].

4.1 Vienna Grid Environment

VGE is based on a service-oriented architecture and has been built on top of existing standard Grid and Web Services technologies. Under VGE, parallel applications available on various HPC platforms, may be exposed via WSDL [25] as services and securely accessed by multiple remote clients over the Internet using SOAP [20]. VGE comprises a generic Grid service provision framework, a Grid client environment, one or more service registries, and a certificate authority.

The VGE service provision framework automates the task of transforming existing applications into Grid services without forcing the user to cope with the details of Web Services and Grid technologies. VGE services encapsulate native HPC applications available on clusters or other parallel hardware, and offer a common set of operations for data transfer, job execution, job monitoring, error recovery, and application-level quality of service support.

The VGE Grid client environment automates the provision of Web-based user interfaces to Grid services and offers a high-level application programming interface (API) for the development of advanced client-side applications which gives developers full control over service discovery, QoS negotiation and job handling while keeping the underlying implementation details hidden from view.

VGE service registries maintain a list of service providers and the services they support and are usually accessed during service selection and QoS negotiation.

The VGE Certificate Authority is in charge of creating user and service provider certificates in accordance with the Internet X.509 PKI Certificate Policy and Certification Practices [5]. X.509 certificates serve as digital identities in the VGE system and are used to provide transport layer security utilizing SSL connections via https and message layer security based on WS Security [26]. VGE adopts a purely client-driven approach for accessing services, i.e. all interactions of a client with services are initiated by the client and neither call-backs nor notification mechanisms are used. As a consequence, there is no need for tunneling holes through site firewalls or any other special site security compromises.

VGE services support a model and process for agreeing dynamically various QoS properties, including service completion time, cost and others. If for a VGE service the optional QoS support mechanisms are employed, a user or client-application may choose from a set of service providers the one which can satisfy a request subject to certain QoS constraints, eg. within a certain deadline.

4.2 Provision of HPC Applications as Grid Services

Within VGE the transformation of HPC applications into Grid services is based on the concept of *generic application services*. A generic application service provides common operations for data transfer, remote job management, error recovery, and QoS support. These operations are customized for a specific application by means of an XML application descriptor, which includes information about the input/output files and the script for initiating job execution. A generic application services is realized as a Java component, which is transformed automatically into a Web Service with corresponding WSDL descriptions.

In order to provide a parallel ACO code as a Web/Grid Service, the application has to be pre-installed on some Grid host and a job-script for starting the application as well as a corresponding XML application descriptor have to be provided. Using VGE, a corresponding service is generated and deployed within the VGE hosting environment, which is based on Apache Tomcat/Axis. As a result of deployment, the ACO code is exposed as a service and accessible over the Internet via SOAP. The VGE client environment may then be used to automatically generate from the XML application descriptor a Web-based user interface for accessing ACO services from remote machines. Alternatively, the VGE Grid client API may be used to develop a customized client application. VGE client applications usually run on PCs or workstations connected to the Internet and make use of the VGE client API for communicating with services through the VGE middleware. The client side applications handle the creation of service input data, the job execution, and the post-processing of service output data.

By offering parallel ACO algorithms as Grid services it is possible to solve multiple vehicle routing problems concurrently based on transparent on-demand access to clusters or other parallel hardware available in a Grid infrastructure.

Note that currently a VGE Grid service as outlined above does not employ multiple Grid sites (clusters) for solving a single VRP. For the fine-grained parallelization strategies the use of multiple Grid sites may not be feasible due to the high latencies over the Internet. However, for coarse-grained parallelization strategies with minimal communication, a parallelization over multiple Grid sites (e.g. by using MPICH-G2 [13]) may be beneficial.

5 Conclusion

In this paper we investigated different parallel versions of Ant Colony Optimization. While most of the work published on parallel ACO focuses on solution quality and competitiveness effects, we have analyzed the effects of different parallelization strategies both with respect to solution quality and efficiency. Moreover, we have outlined the

provision of ACO codes as Grid services using the Vienna Grid Environment. A more detailed evaluation of ACO Grid services and corresponding strategies for solving large scale real world problem instances in real time will be subject of future work.

References

1. S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. VGE - A Service-Oriented Environment for On-Demand Supercomputing. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.
2. B. Bullnheimer, R. F. Hartl, and Ch. Strauss. A New Rank Based Version of the Ant System: a computational study. *Central European Journal of Operations Research*, 7(1):25–38, 1999.
3. B. Bullnheimer, G. Kotsis, C. Strauss. Parallelization Strategies for the Ant System. In R. Leone et al., editors, *High Performance Algorithms and Software in Nonlinear Optimization*, pages 87–100, 1998. Dordrecht:Kluwer Academic.
4. J. Cao, F. Zimmermann. "Queue Scheduling and Advance Reservations with COSY", Proceedings of the International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, 2004
5. S. Chokhani, W. Ford, R. Sabett, C. Merrill, S. Wu. *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*, <http://www.ietf.org/rfc/rfc3647.txt>, The Internet Society, 2003
6. N. Christofides, A. Mingozzi, and P. Toth. The Vehicle Routing Problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, Chichester, 1979. Wiley.
7. G. Clarke, and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12:568–558, 1964.
8. G. A. Croes. A Method for Solving Traveling Salesman Problems. *Operations Research*, 6:791–801, 1958.
9. P. Delisle, M. Krajecki, M. Gravel, C. Gagne. Parallel Implementation of an Ant Colony Optimization Metaheuristic with OpenMP. In *Proceedings of the 3rd European Workshop on OpenMP, EWOMP2001*, Barcelona, Spain, September 2001.
10. K. F. Doerner, R. F. Hartl, G. Kiechle, M. Lucka, M. Reimann. Parallel Ant Systems for the Capacitated Vehicle Routing Problem. In *Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004*, number LLNS 3004 in Lecture Notes in Computer Science, pages 72–83, Coimbra, Portugal, April 5–7, 2004. Springer.
11. M. Dorigo, and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
12. M. Dorigo, T. Stuetzle. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In F. Glover, and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 251–285, January 2003. Kluwer.
13. N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing (JPDC)*, Vol. 63, No. 5, pp. 551–563, May 2003.
14. D. Merkle, and M. Middendorf. Fast Ant Colony Optimization on Runtime Reconfigurable Processor Arrays. *Genetic Programming and Evolvable Machines*, 3(4):345–361, 2002.
15. M. Middendorf, F. Reischle, H. Schmeck. Multi Colony Ant Algorithms. *Journal of Heuristics*, 8:305–320, 2002.
16. MPI: A Message-passing Interface Standard Version 1.1. MPI Forum, 1995.

17. I. H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operations Research*, 41:421–451, 1993.
18. T. K. Ralphs. Parallel Branch and Cut for Capacitated Vehicle Routing. *Parallel Computing* 29:607–629, 2003.
19. M. Reimann, M. Stummer, and K. Doerner. A Savings Based Ant System for the Vehicle Routing Problem. In W. B. Langdon, et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2002*, pages 1317–1325, San Francisco, 2002. Morgan Kaufmann.
20. SOAP Version 1.2. <http://www.w3.org/TR/soap/>
21. T. Stuetzle. Parallelization Strategies for Ant Colony Optimization. In A. E. Eiben et al., editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, PPSN-V*, number LNCS 1498 in Lecture Notes in Computer Science Number, pages 722–731, Amsterdam, 1998. Springer.
22. E. G. Talbi, O. Roux, C. Fonlupt, D. Robillard. Parallel Ant Colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 17:441–449, 2001.
23. The Vienna Grid Environment. <http://www.par.univie.ac.at/project/vge>
24. WebServices - Axis. <http://ws.apache.org/axis/>
25. Web Services Description Language (WSDL) 1.1, <http://www.w3.org/TR/wsdl>
26. Web Service Security. SOAP Message Security 1.0, OASIS Standard 200401, March 2004

Fully Self Organized Public Key Management for Mobile Ad Hoc Network

Daeseon Choi¹, Seunghun Jin¹, and Hyunsoo Yoon²

¹ Electronics and Telecommunications Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350 KOREA
{sunchoi, jinsh}@etri.re.kr

² Department of EECS, Korea Advanced Instituted of Science and Technology,
373-1 Kusung-dong, Yuseong-gu, Daejeon 305-701 KOREA
hyoon@camars.kaist.ac.kr

Abstract. As there is no central authority or fixed infrastructure in mobile ad hoc network, key management should be performed by the nodes themselves, which means that each distributed peer nodes perform the role of authority. If there is no priori trust relationship between nodes in the network, peer nodes have to make trust relationship by themselves. It is the “fully” self organized key management problem. There must be no dependency of any infrastructure, any central server, any secret share dealer and any initial trust relationship between nodes even from the initial boot strapping stage. And transitivity of trust must be minimized. In this paper we propose a fully self organized public key management scheme supporting all those requirements and limitations. In this approach, all peer nodes can issue public key certificate of other node. Certificate is issued after authenticating physical entity when two peer nodes encounter physically. Trustworthiness of the issuer of the certificate is evaluated by replying party. As there is no priori trust, trustworthiness of the issuer is evaluated from reputation about the issuer. Reputation made by other nodes reflects other node’s experience about the issuer. Collected reputation reflects trustworthiness of the issuer in the whole network. So it is possible to decide trustworthiness without any priori trust.

1 Introduction

The MANET (Mobile Ad hoc NETWORK) is a network that consists of distributed wireless mobile hosts that provide networking function such as routing by themselves. There is no fixed infrastructure and no previously defined configuration or topology. And wireless communication has inherent vulnerability.

There are 3 kinds of security related issues that must be considered due to those characteristics of the MANET[1]. First one is the secure routing. Secure routing is to provide correct routing. To provide it, integrity and authenticity of routing information must be guaranteed. And authenticity of source, intermediate, and destination nodes is also important. Second issue is the co-operation enforcement. To make a MANET work well, every node has to provide its networking function to the other nodes. To provide networking function, they have to use their resources such as battery power. There might be some selfish node that only uses network and not provide their resources.

Co-operation enforcement is a problem to enforce those selfish nodes to cooperate and provide resources for other node's benefit. The last issue is the key management. Key management is the problem of how to create and exchange securely the cryptographic key of each node. What make the key management in MANET different from that one in conventional network is that there is no security infrastructure such as KDC, CA and RA. And there is a priori trust issue. A MANET is organized by one of 2 kinds of manners. In the managed environment, nodes start to work having priori trust information about other nodes from the initial stage. Example of this case is network of military personals who know each other. But in the open environment, there is no trust information about others at initial stage. Example of this case is civil application. This paper handles the issue of open environment key management. It is defined as fully self organized key management problem. To be "fully" self organized key management, there must be no dependency of any infrastructure, any central server, or any secret share dealer. In addition to those infrastructure issues, some requirements or limitation can be specified as follows.

- **Fully distributed:** There must be no assumption about special role of any node or group.
- **Bootstrapping from scratch:** There must not be any initial trust relationship between nodes even from the initial boot strapping stage.
- **Minimized transitivity of trust:** Long chain of trust by normal nodes is not appropriate.
- **Fast usable:** If a node can not communicate with other node from the beginning, it is very impractical.

As far as we know, there is no previous research that confirms to all these requirements of fully self organized key management for MANET in open environment. In this paper we propose a new approach that solves all these problems.

Our approach uses public key certificate, which is a largely accepted and proven technology for key management. But it requires a certificate issuer that is trusted by everybody. In our approach, any node can issue certificate, if it wants. It is possible that there are many certificate issuers in a network. The relying party who receives a certificate has to decide the trustworthiness of the certificate based on the trustworthiness of the issuer. Because, there is no priori trust or assigned role as a certificated authority, the trustworthiness of an issuer can only be decided based on experience about the issuer. The experience consists of relying party's own and other node's experience. The experience of other nodes is called reputation.

The rest of the paper is organized as follows. In section 2, we briefly introduce some previous research related to key management in MANET. Section 3 describes our approach in detail. In section 4, we analyze and evaluate feasibility of our approach. Finally, section 5 draws a conclusion.

2 Related Works

In the previous works, there have been 3 kinds of approaches for key management in MANET. They are methods using symmetric cryptography, approaches based on

public key, and context based authentication approach⁴es. For using symmetric key, each node should know and store all other node's key from the beginning. And to store all other node's keys, large storage is required. Some mobile nodes may have very limited storage. As it requires previous acquisition of communicating party, it is very difficult to adapt to dynamic topology change. So this approach is not appropriate to open environment.

There are various methods using public key. As there is no unique CA, distributed CA was proposed[2][3]. A group of nodes works as a CA. This method is based on threshold cryptography. Signatures from nodes are collected. If the number of signers exceeds a threshold, signature is accepted as valid. To get a certificate, the node has to contact group of nodes. In this method, to distribute authority, the private key share for signing certificate must be distributed at initial stage. And every node in the network should know the corresponding public key. The other method using public key is a web of trust[4]. That method is also used in wired environment with the name of PGP[5]. In this method, there is no group of authority and no need of distribution of private key. Every node can issue a certificate to the nodes that they trust. In this point, our method is similar to this method. But the issued certificate is only trusted by someone who directly or indirectly trusts the issuer. To indirectly trust the issuer, there must be a chain of trust. In other words, this method raises problem of transitive trust. And there must be some previous trust relationship with other nodes, even though they are very small portion of all nodes. To solve the problem of transitive trust, a newly proposed method makes use of mobility of node as a way to make direct trust between each node[6]. Even though this method does not make transitive trust, nodes that have not met with each other cannot communicate with each other. It needs very long time for every node to meet each other, if there is no transitive trust. Public key based methods that do not use certificate are id-based cryptography and cryptography based id. Id-base cryptography is a method that makes public key from identifier[7]. So there is no need of certificate that guarantees the relation between id and public key. So there is no need to evaluate the trustworthiness of the certificate issuer. But, to verify the public key generated from identifier, there must be a trusted third party. Cryptography based id is identifier generated from key[8][9]. It shares benefit of id based crypto and does not need TTP. But identifier generated from cryptographic key is not human readable and memorable.

Third approach is authentication based on context[10]. Shared password method is one of them. People in the specific location share temporary password. This method has limitation of coverage.

All of these previous approaches do not meet every requirements of fully self organized key management problem.

3 Reputation Based Distributed Certification

3.1 Overview

In our approach, when two nodes meet each other at the first time, they exchange their public key. After verifying the other's id and private key proof of possession, a node

stores the other's id and public key. If any node has certificate, it also exchange it. We call this exchange a handshaking. What makes this physical meeting possible is the mobility of nodes in the MANET. After handshaking, a node issues a certificate to the other. When the other node already has a certificate for the public key, the certifier adds its signature to the certificate. So there can be more than one signature in a certificate. The certification is not mandatory. It is done by some volunteering nodes. The motivation for volunteering is that the certification is a way to gain credits in a network. Any node can broadcast the certificate revocation message.

When two nodes that have never hand-shaken before communicate each other, they exchange their certificates. The relying party validates received certificate. The trustworthiness of a certificate is dependent on the trustworthiness of the certifiers who remain their signatures in the certificate. Because there is no priori trust, trustworthiness of the certifiers is evaluated based on different information. The information we use is the relying party's own experience about the certifiers and the reputation about the certifiers in the network. Reputation is the experience of the other nodes in the network. In the following sections, we describe in detail the evaluation metrics of trust worthiness, some operations, and operating scenario of our schemes.

3.2 Metrics

The trustworthiness of a certificate is defined as a summation of the certifiers' trustworthiness. The trustworthiness of a certifier is a metric about correctness of certificates that is issued or signed by the certifier. It is calculated based on two types of the experience of the nodes that handle the certificate. First one is the experience of a node whose certificate is issued or signed by the certifier. The node can verify that the content of its certificate is correct. If it found something wrong, it can recognize that the certifier is wrong. The second one is the experience of a node that verifies the content of the other's certificate during handshaking. The node can compare the submitted id, POP and public key with the content of the submitted certificate. If something wrong was found, that means all signers of the certificate are wrong. This experience of node i about node j is represented as $E(i, j)$. The initial value of E is 0. Whenever a node views a correct certificate, it increases its E values about the signers by one. When it finds an incorrect certificate, it sets the E values about the signers as -1 . If the E value is already negative, the node decreases E value by one. It is difficult to accumulate credit but easy to lose it. What the E value means is that a node can gain credit by issuing or signing certificates.

There are many cases that only a node's own experience is not enough to evaluate the trustworthiness of a certifier. A node can reference the reputation about the certifier in the network. The reputation about a certifier is defined as follows.

$$R(j) = \text{average}(E(r, j) \cdot rw), r \in \text{Set of Reputator}$$

$$rw = \frac{E(i, r)}{\text{Average}(E)} : \text{Reputator Weight}$$

Reputation is the weighted mean of other nodes' experience. The other node is called reputer . It is not necessary to get reputation from all nodes. And the reputation is

weighted by the trustworthiness of the reputer itself. The trustworthiness of a reputer is calculated based on only the node's E values. The reason of not including the reputation of the reputer is to prevent recursion.

The trustworthiness of a certifier $Ts(i, j)$ is the summation of the E value and the R value as shown in the following.

$$Ts(i, j) = E(i, j) + R(j)$$

The trustworthiness of a certificate Tc is defined as follows.

$$Tc = \sum_{s \in S} Ts(j, s) - RW \cdot \sum_{v \in V} Ts(i, v)$$

S : Set of Signer

V : Set of Revokers

RW : Revocation Weight

The trustworthiness of a certificate is summation of all signers' trustworthiness, from which summation of all revokers' trustworthiness is subtracted. Unlike certification, revocation message is broadcasted by the revoker. Like reputation, the trustworthiness of the revoker is included in calculation. So the signer or revoker that has negative trustworthiness work in the opposite way.

A node trusts a certificate if the trustworthiness of a certificate exceeds a threshold. The threshold of a node is defined by each node itself and the value changes as time elapses. At the bootstrapping stage, there is not enough experience and reputation. So at this stage, the threshold is set very low and it increases as time goes by.

3.3 Stored Data & Operations

A node stores the following data.

- Handshake List : $\{\{id, pub_key, key_id, hs_time\}, \dots\}$
- Experience List : $\{\{id, E\}, \dots\}$
- Reputation List : $\{\{reputer_id, E, reputation_time\}, \dots\}$
- Revocation List : $\{\{revoker_id, subject_id, key_id\}, \dots\}$

For Reputation List, reputation time is used to classify an obsolete record. But the records of negative E value are not obsolete.

Some required operations and message format are defined in the followings.

Handshake

```
Handshake(Id, pub key, pop, previous cert)
HandshakeResponse(ack, Certificate(option))
```

When two nodes meet each other physically, they exchange Handshake message. After verifying the information in the message, a node may issue or sign a certificate optionally. The certificate is returned and verified by the certificate owner. The Handshake message includes a node id, a public key of the node, proof of possession of matching private key, and the certificate(if there is). Verifying Handshake message consists of authenticating id, and comparing public key with pop. Verified information is stored in the Handshake List. If there is a certificate in the Handshake message, the content of the certificate is compared with the other information. E values about Signers are updated based on the result of comparison. If something goes wrong, the verifier accuses all signers and revokes the certificate. Accusation and revocation operations are defined in the followings. After all verification, the node may issue or sign a certificate. When a certificate is issued or signed, the certificate owner compares the content of the certificate with the information that it sends before. If there is something wrong, it updates its Experience List and accuses the certifier. Then, the node removes the signature from the accused certifier.

Reputation Query

```
ReputationRequest(certifier id)
ReputationResponse(sender id, certifier id, E, signature)
```

When a node needs, it sends the ReputationRequest messages to other nodes. The nodes to be queried are selected based on the sender node's Experience List. The n numbers of nodes of most high E value are chosen. If there is not enough information in the own Experience List, neighbor nodes are chosen.

Revocation

```
Revocation(sender id, subject id, key id, signature)
```

Any node can broadcast the Revocation message. As mentioned before, Revocation message is reflected based on the trustworthiness of the sender. The revocation message includes sender id, certificate owner's id, key id and the sender's signature. The receiving node stores the information in its Revocation List.

Accusation

```
Accusation(sender id, certifier id, signature)
```

Revocation is about a certificate. Accusation is about a certifier that issued or signed certificates. The Accusation message is also broadcasted. The message includes sender id, accused certifier id and the sender's signature. The receiving node updates Reputation List.

Certificate Validation When a node communicates with other nodes that are not in the Handshake List, it sends its certificate and receives the other's certificate. The node validates the received certificate. Except the general process of the certificate validation, the trustworthiness of certificate T_c is evaluated. The node calculates T_c based on information in its Experience List, Reputation List and Revocation List. If the calculated T_c value does not exceed threshold, the node send Reputation Query to collect more information. After recalculating the T_c value, if the T_c value is still not enough, the node does not accept the certificate.

3.4 Scenario

With the metrics and operations that are defined in the previous section, the nodes of a MANET use and manage public key system. The operating scenario is very simple. At the bootstrapping stage, the nodes initialize their stored data and clock. They move and meet other nodes. They handshake, certify, verify or accuse some one. They increase their threshold for accepting a certificate. When a new node joins the network, it asks its neighbor nodes' experience. Based on the reputation, the joining node meets the most trusted nodes in the network and gets certified.

The Fig. 1 shows some example of these key management operations. Fig. 1 (a) shows Handshake operation including certification. Node e issues certificates to the node c and node d . After verifying issued certificate, the two nodes update their experience about node e . As shown in the Fig. 1(b), node c handshakes with node b and node c . Node b and node d verify node c 's certificate and update experience about issuer node e . Node c transfers its certificate to the node a as in the Fig. 1(c). As node a does not have data for validating the node c 's certificate that is issued by node e , node a queries reputation to the node b and node d . The two queried nodes send their experience to node a . Now node a has reputation data such as $\{b, e, 1\}$, $\{d, e, 2\}$. Average reputation is 1.5. If node a 's threshold at this time is 1, node a accepts the certificate issued by node e . As shown in this example, a node can evaluate a certificate, even when the node has never met the owner or the issuer of certificate. Fig. 1(e) shows Accusation and Revocation operations. If node c , node d , and node e conspire and node e makes a wrong certificate, node b can detect that the node e issues the wrong certificate when node b and node c handshake. Node b executes Accusation and Revocation operations.

Then node a has Reputation List and Revocation List as shown in the Fig. 1(e). When node a evaluate node c 's certificate, the average reputation is 0.5 that is below the node a 's threshold 1. So the node a does not accept the certificate issued by node e .

4 Analysis

4.1 Security

To strengthen the security of key management, our approach includes a method for accusing malicious certifier and revoking false certificate. There were no previous researches that considered these factors for key management in MANET. In our method, information about malicious certifier is broadcasted as an accusation message. As there

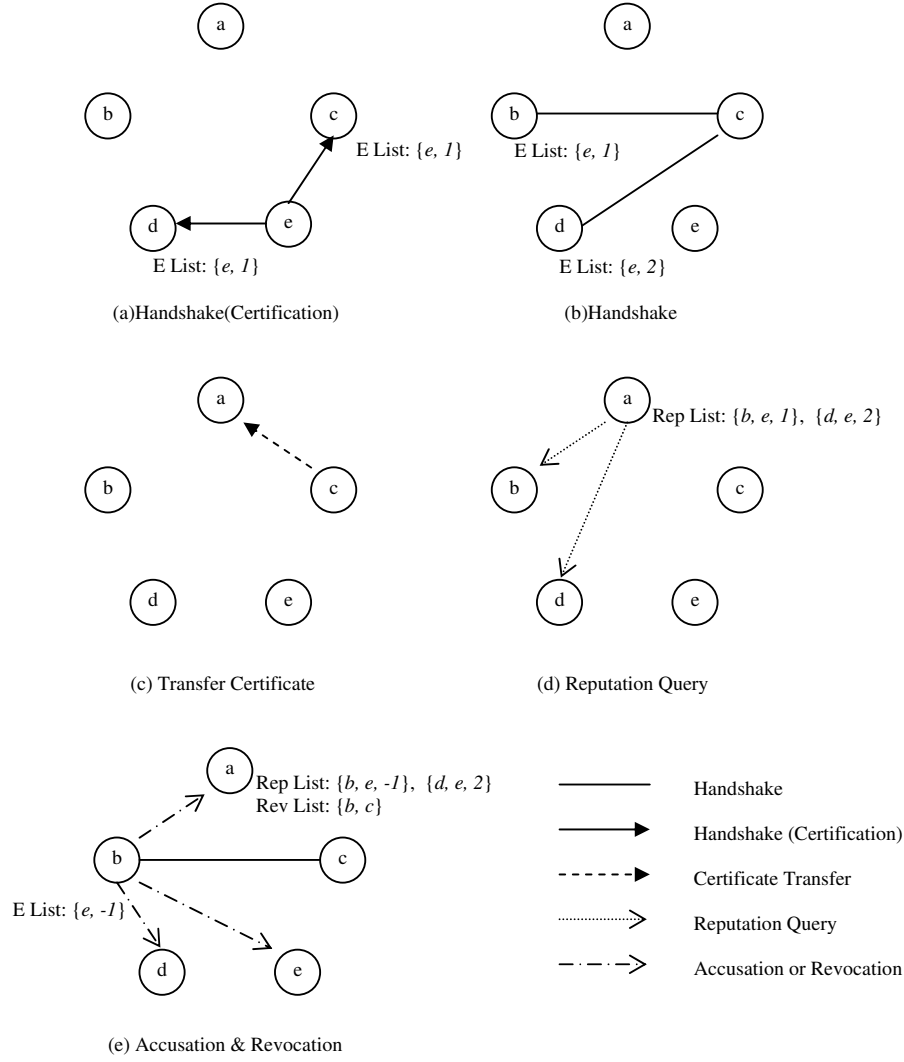


Fig. 3.1. Examples of key management operations

can be a malicious accuser that broadcasts false accuse message, trustworthiness of the accuser is also considered in evaluating trustworthiness of a certifier. If some nodes with high trustworthiness or many nodes repute positively about the issuer, the accusation that is send by lower trusted accuser affect not much in evaluation. Not accuser but all repusers' trustworthiness is considered. Revocation is also considered in validating a certificate. Like accusation, revocation is also accepted based on the trustworthiness of the revokers to prevent false revocation. It will take a certain amount of time for infor-

mation about malicious nodes to propagate. To show how fast our scheme can exclude malicious nodes, we are conducting some experiment with a simulation

In our scheme, there are two ways for a certificate owner to do for increasing trustworthiness of his own certificate. The one is to get signed by many certifiers. And the other is to get certified by the highly trustworthy certifiers. The way to become a highly trusted certifier is to do many certifications. Therefore, highly trusted certifier will get more and more certification requests (Handshake) and their trustworthiness will increase more and more. As time elapses, a few certifiers with highly trustworthiness will emerge. This is one of some ways for a Trusted Third Party to be made in the human society. It is fully self organized manner without no priori information or trust. So we call our scheme “fully self organized”.

4.2 Requirement Conformation

In this section we show that how our approach solves four requirements defined in the section 1.

- **Fully distributed:** In our scheme, there is no previous assumption about role of any node.
- **Bootstrapping from scratch:** In our scheme, every node has empty data storage at initial stage. There is no initial trust relationship.
- **Minimized transitivity of trust:** There is no transitivity of trust. Every node evaluates trustworthiness of certifier by itself with just referencing other nodes' experience. There is no certificate chain as shown in the example described in the section 3.
- **Fast usable:** Every node that has obtained a certificate issued can communicate securely with other nodes that it has never met before. This fact is shown in the example in the section 3. If every node issues a certificate, it takes just one step of move for every node to meet some node and get certified. If only small portion of nodes issues certificate, it takes some steps of move. But it is much small time in comparison with the time took for every node to meet each other.

5 Conclusion

In this paper, we proposed a new approach for key management in MANET environment. In our approach, nodes can evaluate the trustworthiness of a certificate issuer base on its own experience and reputation about the issuer in the network. The authority of certifying node is given by all participating nodes. Accusation of malicious certifier and revocation of false certificate are used to maintain security in the network. Our method does not require any priori trust or any assumption of role of a node. It is fully self organized trust relationship. Moreover, our method makes it possible for every node to communicate securely from the initial deployment stage of a network.

Our scheme can be applied to the other security problems of the MANET in addition to key management. With our scheme, any type of certifier's trustworthiness can be evaluated. For example, self-organized attribute certifier is useful for privilege management and other security operations. For about cooperation enforcement problem, the

certification of cooperation degree can be used as a kind of incentive for cooperation. We think that our scheme can be a comprehensive solution for many security problems in MANET. We are now conducting a research for extending our scheme to the integrated solution.

References

- [1] Refik Molva and Pietro Michiardi. Security in Ad hoc Networks. In Proceedings of the Personal Wireless communication, 2003.
- [2] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad hoc networks. In Proceedings of the 9th International Conference on Network Protocols (ICNP), 2001.
- [3] L. Zhou and Z. Haas. Securing ad hoc networks. IEEE Network, 13(6), 1999.
- [4] Srdjan Cakun, Jean-Pierre Hubaux and Jeff Hodges. Self-organized public-key management for mobile ad hoc networks. IEEE Transactions on Mobile Computing, 2(1), 2003.
- [5] P. Zimmermann. The Official PGP User's Guide. MIT Press, 1995.
- [6] Srdjan Cakun, Jean-Pierre Hubaux and Levente Buttyan. Mobility Helps Security in Ad hoc Networks. In Proceedings of MobiHoc, 2003.
- [7] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. SIAM Journal of Computing, 32(3), 2003.
- [8] G. Montenegro and C. Castelluccia. Statistically unique and cryptographically verifiable (SUCV) In Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS), 2002.
- [9] G. O'Shea and M. Roe. Child-proof authentication for MIPv6 (CAM). ACM Computer Communications Review, 2001.
- [10] N. Asokan and P. Ginzboorg. Key agreement in ad hoc networks. Computer Communications, 23, 2000.

Real Time Intrusion Detection System for Malformed Packet Attacks

Eun-Yeung Choi¹, Hyun-Sung Kim², Kee-Young Yoo¹

¹ Department of Computer Engineering at Kyungpook National University
Daegu, Korea, 702-701
sionchoi@infosec.knu.ac.kr, yook@knu.ac.kr

² Department of Computer Engineering, Kyungil University, Korea, 712-701
kim@kiu.ac.kr

Abstract. The number of bypassing attacks related to malformed packets continues to increase, along with more intelligent and skillful hacking techniques. Most existing intrusion detection systems (IDSs) are unable to detect IP fragmentation attacks, as they could not support a packet reassembly method. Therefore, to cope with these problems, the current paper³ proposes a network-based IDS that can efficiently detect attacks based on malformed packets. The system is mainly composed of 5 components : Information collecting agent(IA), Simple analyzing agent(SA), Fragment analyzing agent(FA), Collaboration agent(CA) and Decision engine(DE). The IA extracts the important features from network packets. The SA analyzes simple attacks using packet header information, while the FA detects IP fragmentation attacks using an efficient algorithm. The CA collects not only collecting information at the SA and the FA but also collecting other strange information related to the malformed packet. The DE judges whether or not an intrusion has occurred on the basis of information gathered from target systems by CAs.

The paper is collected separately. The figures could cause difficulty in printing. It is attached in the end in the printed report. It is placed separately in the on-line version of the report. We are sorry for this inconvenience.

³ Corresponding author : Kee-Young Yoo

Predicting Protein-Protein Interactions in Parallel

Yoojin Chung, Sang-Young Cho and Chul-Hwan Kim *

Computer Science & Information Communications Engineering Division
Hankuk University of Foreign Studies, Yongin, Kyonggi-do, Korea
{chungyj, sycho, redfoot}@hufs.ac.kr

Abstract. In general, the interactions between proteins are fundamental to a broad area of biological functions. In this paper, we try to predict protein-protein interactions in parallel on a 17-node PC-cluster using a parallel incremental support vector machine. According to the experiments, we obtained an average speed-up of 4 with an 5-node cluster with 86.6% accuracy, 89.8 % precision and 84.5 % recall. To our knowledge, it is the first try to predict protein-protein interactions in parallel.

1 Introduction

A major post-genomic scientific pursuit is to describe the functions performed by the proteins encoded by the genome. One strategy is to first identify the protein-protein interactions in a proteome, then determine pathways and overall structure relating to these interactions, and finally to statistically infer functional roles of individual proteins. Although huge amount of genomic data are at hand, current experimental protein interaction assays must overcome technical problems to scale-up for high-throughput analysis. In this paper, we try to predict protein-protein interactions in parallel on a 17-node PC-cluster using a parallel incremental support vector machine (SVM for short) [4].

The interactions between proteins are fundamental to the biological functions such as regulation of metabolic pathways, DNA replication, protein synthesis, etc [1]. But, biologists used experimental techniques to study protein interactions such as two-hybrid screens. And these experimental techniques are labor-intensive and potentially inaccurate. Thus, many bioinformatic approaches has been taken to predict protein interactions.

Among them, Bock and Gough [2] first proposed a method to predict protein interactions from primary structure and associated physicochemical features using SVM. It is based on the following postulate: knowledge of the amino acid sequence alone might be sufficient to estimate the propensity for two proteins to interact and effect useful biological function. The postulate is suggested by the virtual axiom that sequence specifies conformation [5]. Here, primary structure of a protein is the amino acid sequence of the protein.

* This work was supported by grant NO. R01-2003-000-10860-0 from the Basic Research Program of the Korea Science& Engineering Foundation.

In this paper, we try to predict protein-protein interactions using only one associated physicochemical feature among amino acid's diverse features such as hydrophobicity, polarity, aroma, charge etc. and we get approximately 94% accuracy, 99% precision, and 90% recall in average when using hydrophobicity feature, which is better than the result of Bock and Gough.

And, it still takes much time to train huge biological data using SVM. But, to our knowledge, there is no try to predict protein-protein interactions in parallel till now. In this work, we get the following results.

- We do diverse experiments using our sequential method to confirm that our method is reasonable.
- We try to predict protein-protein interactions in parallel using a parallel incremental SVM [4] and measure speedup and performances of accuracy, precision and recall on various configurations.

2 Our sequential method

SVM learning is one of statistical learning theory, it is used many recent bioinformatic research, and it has the following advantages to process biological data [2]:

- SVM generates a representation of the nonlinear mapping from residue sequence to protein fold space [7] using relatively few adjustable model parameters.
- SVM provides a principled means to estimate generalization performance via an analytic upper bound on the generalization error. This means that a confidence level may be assigned to the prediction and alleviates problems with overfitting inherent in neural network function approximation [8].

In this paper, we use TinySVM among diverse implementations of SVM and its web site is <http://cl.aist-nara.ac.jp/taku-ku/software/TinySVM>. TinySVM uses many techniques to make large-scale SVM learning practical and thus, it is good for our application with large dimensions of a feature vector of SVM and huge amount of data sets.

SVM is a supervised learning method. Thus we need both positive and negative examples to train SVM. We obtained positive examples (that is interacting proteins) from the Database of Interacting Proteins (DIP for short) and its web site is <http://www.dip.doe-mbi.ucla.edu/>. At the time of our experiments, the DIP database has 15117 entries. And each entry represents a pair of interacting proteins. We make negative examples by using global shuffling to the pairs not in DIP. Here, interacting mean that two amino acid chains were experimentally identified to bind to each other. Our system generate a discrete, binary decision, that is, interaction or no interaction.

Testing sets are not exposed to the system during SVM learning. The database is robust in the sense that it represents protein interaction data collected from diverse experiments. There is a negligible probability that the learning system will learn its own input on a narrow, highly self-similar set of data examples. This enhances the generalization potential of the trained SVM.

	hydro-phobicity	aromatic	small	tiny	aliphatic	polar	positive	charged	negative
accuracy	92.2	86.2	75.2	74.6	72.9	71.1	69.7	66.1	62.6
precision	97.1	93.6	86.8	86.2	84.5	83.2	80.7	76.2	72.3
recall	87.5	77.7	59.4	58.5	56.1	53.0	51.9	46.7	41.2

Fig. 3.1. Comparing 9 features

Now, we will explain how to make a feature vector of SVM. A protein has a various length of an amino acid sequence and its length is from several hundreds to several thousands. Thus, the first step is to normalize the length of each amino acid sequence of a protein. And then, we simply concatenate these two normalized amino acid sequences of protein pair which is interacting or non-interacting. And finally, we replace each amino acid in the concatenated sequence with its feature value.

We evaluate the performance of SVM using accuracy, precision, and recall. Their definitions are as follows.

- Accuracy = $(pp+nn) / (pp+np+pn+nn)$
- Precision = $pp / (pp+pn)$
- Recall = $pp / (pp+np)$,

where p indicates positive, which means there is an interaction, and n indicates negative, which means there is no interaction. And pp is true positive, pn is false positive, np is false negative, and nn is true negative because the right side value is a real value and the left side value is a SVM's predicted value.

The accuracy is the percentage of correct predictions among all the testing set, the precision is the percentage of true positive among all the predicted positive, and the recall is the percentage of true positive among all the real positive testing data. Note that if accuracy is high, recall is an important factor to be a computational screening technique that narrow candidate interacting proteins.

3 Experiments using our sequential method

Now, we will show our experimental results using a sequential method. At the first experiment, we use each of 9 features as a feature vector of SVM and compare the results. 9 features are hydrophobic, aromatic, small, tiny, aliphatic, polar, positive, charged, and negative features. For each features, we do 210 trials. For each try of experiments, we use 4000 pairs of proteins of yeast in DIP database and these data set are selected randomly.

Fig. 3.1 shows the summary of average performances of 9 features. When hydrophobic feature is used, all of accuracy, precision, and recall are best among all the 9 features. And an aromatic feature shows the second performance. And the next is small feature, the next is tiny feature and so on. In Fig. 3.1, from small to negative features, recall is

near or below 50 %. Thus, these are not good criterions to predict protein interactions. Hydrophobic feature shows near 90 % in all performances of accuracy, precision and recall. Thus it is a good criterion to predict protein interactions.

Our results agree with the previous demonstration of sequential hydrophobicity profiles as sensitive descriptors of local interaction sites in Hopp and Woods paper [1]. Thus, we conclude that our feature vector model of SVM to predict protein interactions is reasonable.

To confirm that our model is reasonable, we do 180 trials with increasing the number of training data set of SVM. According to the experiments, as the number of training data increases, all of performances accuracy, precision, and recall are getting better. This results also confirm that our feature vector model of SVM to predict protein interactions is reasonable.

Now, we will show the experimental results comparing kernel models of SVM. For each kernel, we do 180 trials with increasing the number of training data set. When using a linear kernel, average accuracy, precision, and recall are 97.78 %, 98.24 %, and 97.02 %, respectively. When using a polynomial kernel, average accuracy, precision, and recall are 97.70 %, 98.22 %, and 96.88 %, respectively. In both two kernel models, the results are almost the same. But a polynomial kernel takes much more time. And other kernel models of SVM such as neural, RBF and ANOVA show much worse performances and they take much more times than a linear kernel. As a linear kernel is a subset of a polynomial kernel, this result shows that a linear kernel is enough for our protein-protein interaction prediction system. And the fact that linear kernel is the best for our system is very important to parallelize our system. That is explained in the following section.

4 Our parallel method

Now, we will explain our parallel method to predict protein-protein interaction. Fig. 4.2 is the configuration of our 17-node PC-cluster, which consists of one master node and 16 computing nodes. Each PC has a Pentium III processor and they are connected by gigabit ethernet. Each PC has 256M byte memory. Note that memory size is a main constraint to deal with very large dimensions of a feature vector of SVM in our PC-cluster system.

Our sequential protein-protein prediction system use a feature vector of twenty thousand dimensions. And, SVM requires memory more than of n^2 , when n is the number of dimensions of a feature vector of SVM. Thus, our sequential prediction system cannot work in our PC-cluster because of the lack of a memory. To solve this problem, we do experiments using the following two methods and the results are described in the next section.

- We develop a method to reduce dimensions of a feature vector using our feature vector's characteristics.
- We compress each pixel value with high dimensionality and encode it.

To do SVM training in parallel, we mainly refer to Fung and Mangasarian's Incremental SVM [4] and its parallel implementation [6]. Now, we will explain the main idea of incremental SVM.

Number of nodes	16 computing nodes 1 master nodes
Processor	Inter Pentium III coppermine 866Mhz (32KB L1, 256KB L2 cache)
Main memory	256MB SDRAM
Hard disk	30GB EIDE HDD
Network	3com Gigabit Ethernet switch 100 Base-T Ethernet NIC
OS	Redhat Linux
Language	MPICH
Physical size	W 180cm × D 60cm × H 130cm

Fig. 4.2. The configuration of our 17-node PC-cluster

4.1 Incremental SVM

As depicted in Fig. 4.3(a), we consider the problem of classifying m points in the n dimensional input space R^n , represented by the $m \times n$ matrix A , according to membership of each point A_i in the class $A+$ or $A-$ as specified by a given $m \times m$ diagonal matrix D with plus ones or minus ones along its diagonal. For this problem, the standard SVM with a linear kernel is given by the following quadratic program with parameter $\nu > 0$ [4]:

$$\begin{aligned}
 \min_{(\omega, \nu, y) \in R^{n+1+m}} \quad & \nu e^t y + \frac{1}{2} \omega^t \omega \\
 \text{s.t.} \quad & D(A\omega - e\gamma) + y \geq e, \\
 & y \geq 0.
 \end{aligned} \tag{4.1}$$

As depicted in Fig. 4.3(a), ω is the normal to the bounding planes:

$$\begin{aligned}
 x^t \omega &= \gamma + 1 \\
 x^t \omega &= \gamma - 1
 \end{aligned} \tag{4.2}$$

that bound most of the sets $A+$ and $A-$ respectively. The constant γ determines their location relative to the origin. When the two classes are strictly linearly separable, that is the error variable $y = 0$ in (4.1), the plane $x^t \omega = \gamma + 1$ bounds all of the class $A+$ points, while the plane $x^t \omega = \gamma - 1$ bounds all of the class $A-$ points as follows:

$$\begin{aligned}
 A_i \omega &\geq \gamma + 1, \text{ for } D_{ii} = 1, \\
 A_i \omega &\leq \gamma - 1, \text{ for } D_{ii} = -1.
 \end{aligned} \tag{4.3}$$

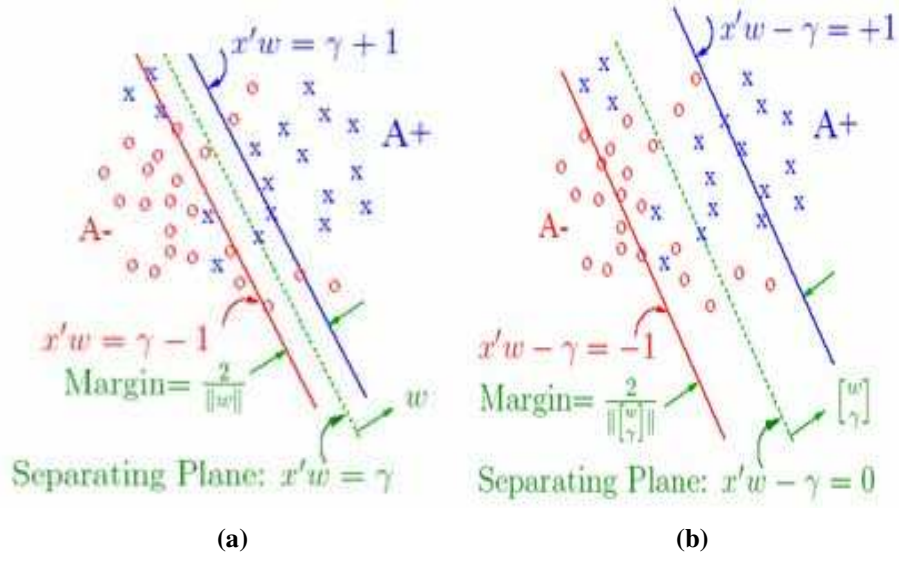


Fig. 4.3. A standard and a proximal SVMs

Consequently, the plane

$$x'\omega = \gamma, \quad (4.4)$$

midway between the bounding planes in (4.2), is a separating plane that separates $A+$ from $A-$ completely if $y = 0$, else only approximately as depicted in Fig. 4.3(a). The quadratic term in (4.1), which is twice the reciprocal of the square of the 2-norm distance $\frac{2}{\|w\|}$ between the two bounding planes of (4.2), maximizes this distance, often called the “margin”. Maximizing the margin enhances the generalization capability of SVM. If the classes are linearly inseparable, which is the case shown in Fig. 4.3(a), then the two planes bound the two classes with a “soft margin” (i.e. bound approximately with some error) determined by the nonnegative error variable y , that is:

$$\begin{aligned} A_i\omega + y_i &\geq \gamma + 1, \text{ for } D_{ii} = 1, \\ A_i\omega - y_i &\leq \gamma - 1, \text{ for } D_{ii} = -1. \end{aligned} \quad (4.5)$$

The 1-norm of the error variable y is minimized parametrically with weight ν in (4.1), resulting in an approximate separating plane as depicted in Fig. 4.3(a). This plane acts as a linear classifier as follows:

$$\text{sign}(x'\omega - \gamma) \begin{cases} = 1, & \text{then } x \in A+, \\ = -1, & \text{then } x \in A-. \end{cases} \quad (4.6)$$

Fund and Mangasarian [4] make a very simple change in the numerical formulation of a standard SVM and changes the nature of an optimization problem of a standard SVM significantly, which is depicted in Fig. 4.3(b) and it is called a proximal SVM. This change can be applied only to a linear kernel of a standard SVM. Two planes, which is represented by black lines in Fig. 4.3(b), are not bounding planes anymore. But, they can be thought of as ‘proximal’ planes, that is, around the planes the points of each class are clustered, and the two planes are pushed as far apart as possible.

Given m data points in R^n represented by the $m \times n$ matrix A and a diagonal matrix D of \pm labels denoting the class of each row of A , the linear proximal SVM generates classifier (4.6) as follows.

- Define $E = [A \ -e]$ where e is an $m \times 1$ vector of ones. Compute

$$\begin{bmatrix} \omega \\ \gamma \end{bmatrix} = \left(\frac{1}{\nu} + E'E \right)^{-1} E' D e \quad (4.7)$$

for some positive γ . Typically γ is chosen by means of a tuning set.

- Classify a new x by using (4.6) and the above solution $\begin{bmatrix} \omega \\ \gamma \end{bmatrix}$.

Then, an incremental SVM can be generated by using a simple procedure to the proximal SVM. An incremental SVM can process data using a divide-and-conquer approach and thus it can be implemented in parallel directly. Tveit and Engum [6] implement it and we use it for our experiments.

5 Experiments using our parallel method

As different values of ν in (4.7) in subsection 4.1 can give difference in accuracy in the classification, we get the following experimental result with various ν values. Because [12] shows that 10 is about the right number of folds to get the best estimate of error, we do our experiments using a 10-fold Cross-Validation [12]. As shown in Table 1, when ν value is 100, we get the best result, i.e., accuracy is 86.8%, precision is 89.8% and recall is 84.5%. We use this ν value in the following experiments.

Table 2 compares the average performances of a standard SVM and an incremental SVM. For each trial, we use 4000 protein pairs and we do 10 trials.

According to the experiments, our method using an incremental SVM shows average performances of 86.6% accuracy, 89.8 % precision, and 84.5 % recall. It is worse than a standard SVM, which has 95.1% accuracy, 97.9 % precision, and 86.8 % recall. But, the result of an incremental SVM is not bad because all of the performances of accuracy, precision, and recall are above 80 %.

Our PC-cluster has 17 nodes but we get experimental results only using 1 node, 3 nodes and 5 nodes, respectively, till now. According to the experiments, all of the performances of accuracy, precision, and recall are the same regardless of the number of nodes as proved in the numerical formulation of an incremental SVM [4]. And we obtained an average speed-up of 2.68 with a 3-node and 4.04 with a 5-node cluster.

ν -value	accuracy	precision	recall
0.001	76.6	74.0	78.0
0.01	76.6	74.2	78.0
0.1	77.4	74.5	79.0
1	80.2	78.6	81.2
10	82.4	81.3	83.2
100	86.6	89.8	84.5
1000	70.2	97.3	63.1
10000	50.9	100	50.4

Table 1. Testing of ν values

	incremental SVM	standard SVM
accuracy	86.6	95.1
precision	89.9	97.9
recall	84.5	86.8

Table 2. Comparing a standard and an incremental SVMs

To use all the proteins in a DIP database in our PC-cluster, we develop a method to reduce dimensions of our feature vector using our feature vector's characteristics. Then, this experimental result shows average performances of 74.1 % accuracy, 71.9 % precision, and 75.6 % recall with 10 trials, which is much worse than both of a standard and an incremental SVMs. This degradation is caused by reduced dimensions of a feature vector of SVM, because reducing dimensions always loses an information.

6 Conclusion

In this work, we showed how to predict protein-protein interactions in parallel using incremental SVM and showed various experimental results. But, the most difficult thing in using a supervised learning method such as a SVM to predict protein-protein interactions is to find negative examples of interacting proteins, i.e., non-interacting protein pairs. If such a database of non-interacting protein pairs are constructed, the better method to predict protein-protein interactions can be developed.

In the near future, we want to improve our parallel protein-protein interaction prediction system by developing an improved SVM for our system. And, we want to develop an effective method to reduce large dimensions of a feature vector of SVM. With experimental validation, further development may produce a robust computational screening techniques that narrow the range of candidate interacting proteins.

References

1. T.P. Hopp and K.R. Woods. Predicting of protein antigenic determinants from amino acid sequences, *Proc. Natl. Acad. Sci. USA*, 78, 3824-3828 (1981).
2. B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*, Garland, New York, 2nd edition, 1989.
3. J. R. Bock and D. A. Gough. Predicting protein-protein interactions from primary structure, *Bioinformatics*, 17:455-460, 2001.
4. G. Fung and O. L. Mangasarian. Incremental support vector machine classification, In *2nd SIAM int'l. Conf. on Data Mining*, SIAM(2002) 247-260.
5. Y. Chung, G. Kim, Y. Hwang, and H. Park. Predicting protein-protein interactions from one feature using svm, In *IEA/AIE'04 Conference Proceedings*, Ottawa, Canada, May 2004.
6. A. Tveit and H. Engum. Parallelization of the Incremental Proximal Support Vector Machine Classifier using a Heap-based Tree Topology, Technical Report, IDI, NTNU, Trondheim, Norway, August 2003.
7. C. B. Anfinsen. Principles that govern the folding of protein chains, *Science*, 81(1):223-230, 1973.
8. T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods-Support Vector Learning*, Chap. 11, MIT Press, Cambridge, MA, 169-184, 1999.
9. P. Baldi and S. Brunak. Bioinformatics: the machine learning approach. In *Adaptive Computation and Machine learning*, MIT press, Cambridge, MA, 1998.
10. C. Bishop. In *Neural networks for pattern recognition*, Oxford University Press, UK, 1996.
11. P. Cohen. In *Empirical methods for artificial intelligence*, Chap. 6.10, MIT Press, 216-219, 1995.
12. I. Witten and E. Frank. In *Data mining: Practical machine learning tools with Java implementations*, Chap. 5.3, Morgan Kaufmann Publishers, 125-127, 2000.

On the Parallelization of the Lattice-Boltzmann Method

Salvatore Filippone, Nicola Rossi, Gino Bella and Stefano Ubertini

University of Rome “Tor Vergata”
Department of Mechanical Engineering and Computing Center
Via Orazio Raimondo 18, I-00173, Rome, Italy
{bella, salvatore.filippone}@uniroma2.it

1 Introduction

Computational fluid dynamics, in its conventional meaning, computes pertinent flow fields in terms of velocity, density, pressure and temperature by numerically solving the Navier-Stokes equations in time and space.

At the turn of the 1980s, the Lattice Boltzmann Method (LBM) has been proposed as an alternative approach to solve fluid dynamics problems [1, 2] and due to the refinements and the extensions of the last years [3–5], it has been used to successfully compute a number of nontrivial fluid dynamics problems, from incompressible turbulence to multiphase flow and bubble flow simulations. The main advantages of LBM with respect to conventional CFD are its simpler mechanism for doing dynamics, its easy numerical implementation and its intrinsic parallelism.

The most severe limitation of the original LB method is the uniform Cartesian grid on which the LBM must be constructed, requiring the approximation of a curved solid boundary by a series of stair steps. This represents a particularly severe limitation for practical engineering purposes especially when there is a need for high resolutions near the body or the walls. Among the recent advances in lattice Boltzmann research that have lead to substantial enhancement of the capabilities of the method to handle complex geometries [3–5], a particularly remarkable option is to use irregular lattices by changing the solution procedure from the original “stream and collide” to a finite volume technique [6–8].

In most applications of LB it is necessary to employ large discretization meshes; thus, it is appropriate to use parallel computing techniques, and this is a major theme of this paper.

The paper is organized as follows: in section 2 we outline the finite volume formulation of the Lattice-Boltzmann equation, in section 3 we discuss some implementation issues, with some experimental results presented in 4; then we draw our conclusions in 6.

2 The Unstructured LB model

LBM takes inspiration from the idea of solving fluid flows through a microscopic kinetic approach, trying to mathematically describe movements and interactions of the small particles that constitute the flow with the assumption that the solute concentrations are sufficiently low not to influence the solvent flow. In this method the collective

degrees of freedom are discrete one-body distribution functions $f(x, v, t)$, representing the probability to find a particle at given point x at time t with a velocity c . The key of the LB method (inherited from its ancestor, the Lattice Gas Cellular Automata) is that only a very limited set of discrete speeds c_i is retained (see fig 2.1).

$$\partial_t f_i + \mathbf{c}_i \cdot \nabla f_i = -\frac{f_i - f_i^{eq}}{\tau} \quad (2.1)$$

The right hand side of eq. 2.1 is called collision function (or operator) and describes molecular collisions via a single-time relaxation towards local equilibrium, f^{eq} , on a typical timescale, τ . For three-dimensional flows there are several cubic lattice models, such as the 15-bit, the 19-bit and the 27-bit, for velocity space discretization. In this paper the 19-bit model is used (Q19D3; see fig. 2.1); thus in eq. 2.1 we have $i = 1, \dots, 19$. This local equilibrium is a (local) Maxwellian expanded to second order in the

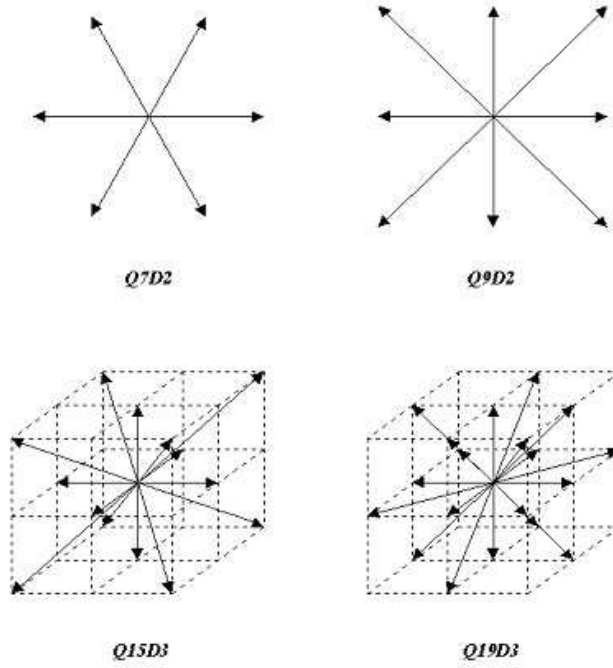


Fig. 2.1. LB discretizations

fluid speed:

$$f_i^{eq} = \rho w_i \left[1 + \beta u_i + \frac{\beta^2}{2} (u_i^2 - u^2) \right] \quad (2.2)$$

where w_i are weighting factors normalized to unit value, $u_i = \mathbf{u} \cdot \mathbf{c}_i$ and $\beta = 1/c_s^2$, c_s being the lattice sound speed, defined by the equation $c_s^2 = \sum_i w_i c_i^2$, ($c_s = 1/\sqrt{3}$ in the present work). In the limit of weak departures from local equilibrium, i.e. small Knudsen numbers, it can be shown through a Chapman–Enskog analysis that LBE recovers the dynamic behaviour of a fluid with the fluid density, the flow velocity and pressure given respectively by:

$$\rho = \sum_i f_i \quad \mathbf{u} = \sum_i \mathbf{c}_i f_i / \rho \quad p = \rho c_s^2 \quad (2.3)$$

Crucial to the LB hydrodynamics is the momentum flux tensor, defined as:

$$\mathcal{P} = \sum_i f_i \mathbf{c}_i \mathbf{c}_i^T$$

The equilibrium component of this tensor controls advection and pressure terms, while the non equilibrium part is in charge of describing dissipative effects. In order to precisely conserve hydrodynamic moments, the set of discrete speeds must be properly defined.

The approach proposed here to numerically solve the LBE is a finite-volume scheme of the cell-vertex type based on a space discretization into tetrahedral elements. The 19 discrete populations associated to each node P of the discrete grid represent the unknowns of the problem. The finite volume over which eq. 2.1 is integrated is defined by means of the set of K elements, which share P as a common vertex. Since the discrete grid is unstructured each node is identified by its coordinates and each element is identified by the connectivity.

The integration of the left hand side of eq. 2.1 over the finite volume k -th portion W_k is approximated as follows:

$$\int_{W_k} (\partial_t f_i + \mathbf{c}_i \cdot \nabla f_i) dV_k = [f_i(P, t + dt) - f_i(P, t)] \frac{W_k}{dt} + \Phi_{i,k} \quad (2.4)$$

where a first-order time marching solution has been chosen and $\Phi_{i,k}$ is the flux associated to the streaming operator of the i -th particle distribution function through the edges of W_k after the application of the Gauss theorem. The sum k runs over the volume obtained by joining the centers of the tetrahedrons with the centers of the triangular edges.

The molecular collision contributions arise from the integration of the collision term over each volume W_k . The resulting collisional flux

$$\Xi_{i,k} = \int_{W_k} \left(\frac{f_i - f_i^{eq}}{\tau} \right) dV_k$$

is computed by calculating the local non-equilibrium distribution function via a linear interpolation.

Therefore, the finite-volume equation takes the following form:

$$f_i(P, t + dt) = f_i(P, t) + \frac{dt}{W_k} \sum_{k=0}^K (\Phi_{i,k} - \Xi_{i,k}) \quad (2.5)$$

where index $k = 0$ denotes the pivotal point P. The detailed expressions of the streaming and collision matrices S_{ik} and C_{ik} are easily obtained by straightforward application of the interpolation rules:

$$f_i(P, t + dt) = f_i(P, t) + dt \sum_{k=0}^K S_{ik} f_i(P_k, t) - \frac{dt}{\tau} \sum_{k=0}^K C_{ik} [f_i(P_k, t) - f_i^e(P_k, t)] \quad (2.6)$$

The equation 2.6 defines a time-marching explicit scheme to compute the steady-state particle distribution functions f_i , $i = 1, \dots, 19$; note that the basic step can be implemented as a series of matrix-vector products, with a coefficient matrix that is constant throughout the simulation.

The following sum rules hold:

$$\sum_{k=0}^K S_{ik} = 0, \quad \sum_{k=0}^K C_{ik} = 1, \quad \forall i$$

These play an important role in the theoretical analysis of the scheme, as detailed in [7].

3 The Implementation

The finite volume formulation outlined in the previous section uses matrix-vector products to advance the simulation through multiple time steps. The formulation has a natural consequence, i.e. the matrices involved are large and sparse: to each volume of the discretization mesh there corresponds a row of the discretized streaming/collision operators, and there are non zero coefficients (i.e. interactions) only in correspondence with physically contiguous volumes. Therefore the code can be implemented by making appropriate use of a standard sparse linear algebra package such as PSBLAS [15], which was originally developed at our university for usage in the context of solution of PDEs by implicit discretization schemes. The library provides the computational kernels normally used in the implementation of sparse iterative solvers, including the matrix-vector product, together with environment handling routines for implementation by means of message-passing on distributed memory architectures. The parallelization is based on domain decomposition techniques, in that the discretization space is allocated to the various parallel processes, while the library handles the data structures necessary to the underlying communication.

The implementation scheme proceeds along the following lines

1. Read the discretization mesh;
2. Build the streaming and collision matrices S and C ;
3. Iterate:
 - (a) Apply boundary conditions;
 - (b) Apply streaming and collision operators;
 - (c) Compute macroscopic quantities by appropriately integrating the population distributions.

In the iteration loop most of the communication among processors takes place during the matrix-vector products implementing the streaming and collision phases; the recovery of macroscopic quantities is done independently for each node of the discretization mesh, thus no communication is needed.

Note that the structure of the mesh does not vary at all during the simulation; this means that the sparse matrices are constant, and thus the setup phase is negligible given that the number of time steps is normally very high.

One of the most critical issues for Lattice Boltzmann techniques is the implementation of the boundary conditions, since the unknowns are the populations, while boundary conditions in CFD are defined as functions of the macroscopic fluid dynamics variables (i.e. velocity and pressure) or their derivatives. The following strategies have been applied for boundary conditions:

- co-volume method [6] for no-slip and free-slip boundary conditions;
- zero-longitudinal pressure or velocity gradients at open boundaries with buffers of ghost nodes [7, 8].

In the co-volume method boundary nodes are treated as fluid nodes with the only difference that the fluxes through boundary edges must be computed)

Using the PSBLAS library facilities of [15] we have obtained very rapidly a workable parallel version of the application, which correctly reproduces the basic physical features of the phenomena of interest.

4 Experimental results

We performed a series of experiments on the reference test case shown in Fig. 4.2. The physical structure under consideration is a square duct with an embedded sphere. The flow is initially at rest (zero speed) and is impulsively started with a uniform velocity profile U_∞ at inlet. The chosen outflow boundary condition is the zero-longitudinal velocity gradient with a prescribed pressure P_∞ . Tests have been carried out successfully for Reynolds number varying from 10 to 100. In order to prove the accuracy of the present Lattice Boltzmann simulation numerical results have been compared with experimental and numerical literature data.

A significant advantage of ULBE compared to traditional LBE and Navier-Stokes is that both pressure and friction contributions to the aerodynamic force are locally available as a linear combination of the particle distribution functions [7, 8]. Upon reaching steady-state, the drag coefficient C_D , a well known parameter used to characterize the aerodynamic force, is measured as:

$$C_D = \frac{2F_z}{\rho U_\infty^2 A}$$

where A is the projection area of the sphere and F_z is the stream wise component of the aerodynamic force acting on the sphere. The numerical values of the drag coefficient, reported in Figure 4.3 over the range $10 \div 100$ of Reynolds numbers, show good agreement with literature data [9–12], both numerical and experimental, since they fit the

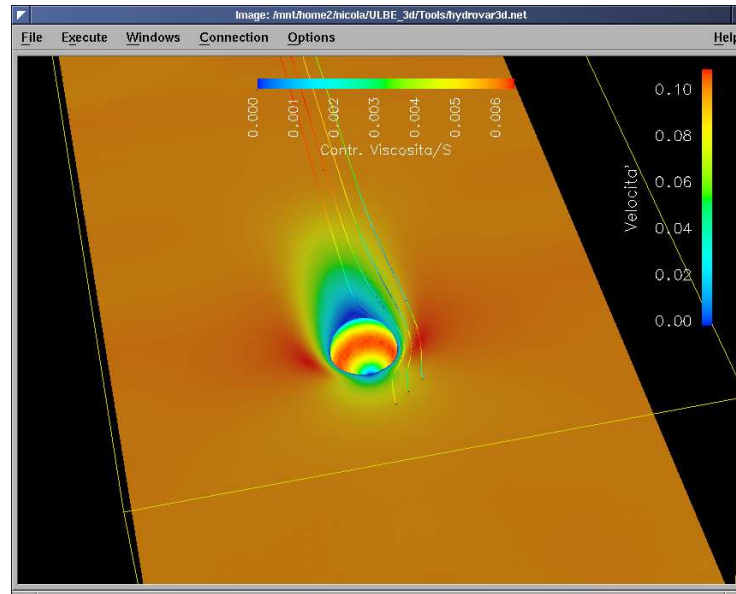


Fig.4.2. Reference test case

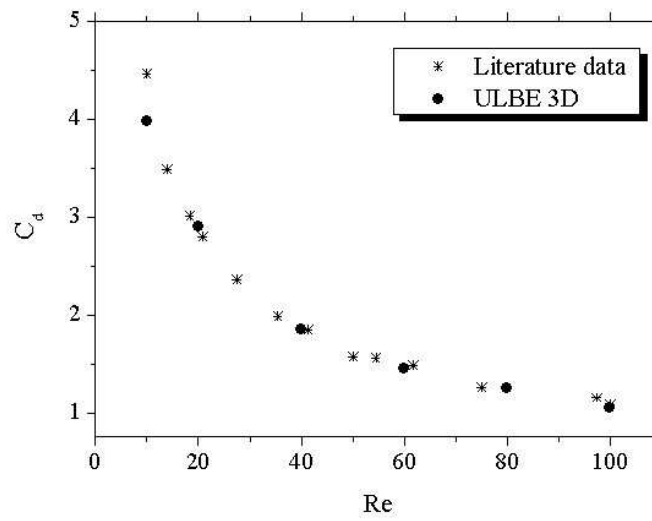


Fig.4.3. Drag coefficient

same curve. Figure 4.4 shows the pressure coefficient C_P as a function of the angular coordinate θ

$$C_P = \frac{2(P - P_\infty)}{\rho U_\infty^2}$$

The pressure coefficient trend, which is simply the pressure at all points around the sphere, has been compared to numerical data found in literature [13, 14].

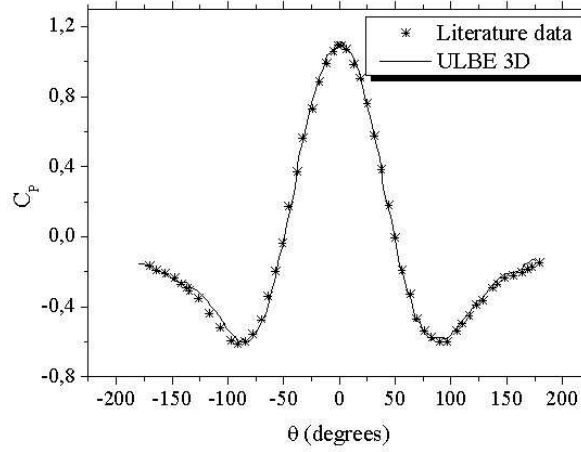


Fig. 4.4. Pressure coefficient

The graph in fig 4.2 shows the velocity field in the fluid, as well as the pressure on the sphere surface. This simulation requires a discretization mesh with 483000 nodes; this is also the row/column size of the sparse matrices involved in the computations. The nonzero pattern is quite irregular, since the discretization mesh is unstructured. The physical results are based on a run of 50000 time steps; in serial computation on an Intel PIV at 2.6 GHz each iteration takes approximately 3 seconds, and the whole simulation more than 40 hours; thus the necessity of parallelization to obtain the good physical results characteristic of the method in a reasonable amount of time.

To estimate the effectiveness of the parallelization we have performed two sets of test runs on a cluster of AMD Athlon at 1.8 GHz connected with a Gigabit Ethernet switch, shown in Table 1, on the same geometry shown in 4.5, but with different number of discretization volumes. The test runs comprised about 300 time steps, and thus are much shorter and more manageable than a complete simulation; they are nonetheless representative, since the amount of computational load does not change throughout the whole run.

Since a complete run would entail about 50000 time steps, we have computed the speedups in Table 1 without accounting for the initialization phase; this is justified, since the value we report in the table is of the order of 100s, and therefore would be negligible

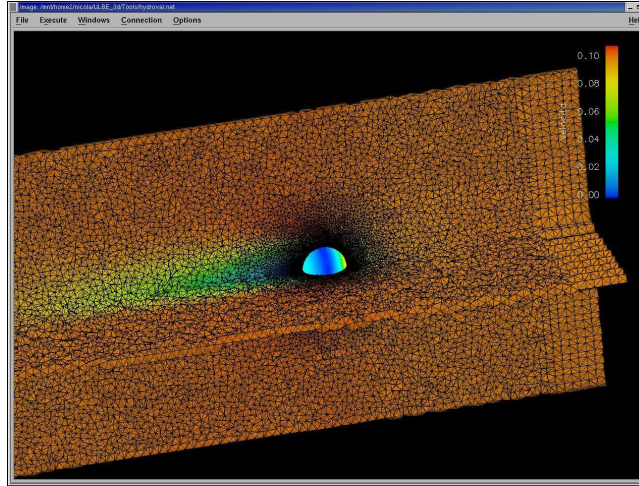


Fig. 4.5. Reference test case: enlarged view of the mesh

Medium (157 K nodes) initialization 110s				
Processors	Total time	Speedup	Avg. time per step	Speedup
1	724	1.00	2.40	1.00
2	389	1.86	1.28	1.88
3	283	2.56	0.93	2.58
4	218	3.32	0.71	3.38
Large (305 K nodes) initialization: 145s				
Processors	Total time	Speedup	Avg. time per step	Speedup
1	1508	1.00	5.01	1.00
2	751	2.01	2.48	2.02
3	616	2.45	2.03	2.47
4	460	3.28	1.50	3.34

Table 1. AMD Athlon, Gigabit Ethernet cluster

in a full run. The speedup on the overall run is somewhat less than the speedup per time-step; this is due to the post-processing routines that have not been parallelized yet. Since they are invoked once every several time steps their impact is not too large.

We are currently testing the code on a cluster of Intel Xeon running at 3 GHz with a Myrinet switch, and the preliminary results are encouraging, even though there still is room for improvement.

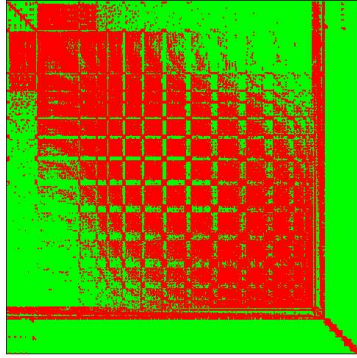


Fig. 4.6. Natural numbering

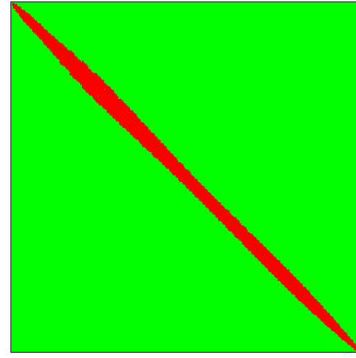


Fig. 4.7. Band reduction numbering

One critical issue in achieving good parallel performance was the use of an appropriate numbering strategy; indeed, the default numbering scheme used by the grid generator, gave rise to the matrix sparsity pattern shown in Fig. 4.6; this pattern causes a tremendous performance penalty if used directly, because:

1. At the serial level it enables little or no cache reuse;
2. At the parallel level a straightforward partition of the matrix will result in too much data communication among processors.

Therefore we applied a standard renumbering tool to reduce the bandwidth, obtaining the pattern in Fig. 4.7; future work will include testing alternative numbering strategies.

5 Conclusions and future work

We have presented an implementation of the Lattice Boltzmann method for unstructured grids based on the formulation of [7, 8]; besides the nice numerical properties, such as the numerical viscosity effects being within second order of accuracy in space, the method readily lends itself to an implementation in terms of standard linear algebra kernel.

It is therefore possible to apply a parallelization strategy based on standard software [15], and this has been verified to be working.

Future developments include both tuning for optimal performance and application to more difficult physical problems, where the computational requirements of the serial version of the method are prohibitive.

References

1. Benzi, R., Succi, S. and Vergassola, M., 1992, The lattice Boltzmann equation: theory and applications, *Phys. Rep.* 222, 145-197.
2. S. Succi, The lattice Boltzmann equation for fluid dynamics and beyond, Oxford University Press, 2001.
3. Filippova, O. and Hanel, D., 1998, Grid refinement for lattice-BGK models, *J. Comp. Phys.* 147, 219
4. Mei, R., Luo, L.-S., Shyy, W., 1999, An accurate curved boundary treatment in the lattice Boltzmann method, *J. Comp. Phys.* 155, 307
5. Bella, G., Ubertini, S. and Bertolino, M., 2003, Computational Fluid Dynamics for Low and Moderate Reynolds Numbers through the Lattice Boltzmann Method, *Int. J. of Comp. and Num. Analysis and Applications*, IJCNAA, vol. 3, No. 1 (2003), pp. 83-115.
6. Peng, G., Xi, H, Duncan, G. and Chou, S.H., 1998, Lattice Boltzmann method on irregular meshes, *Phys. Rev. E*, Vol. 58, No. 4, pp. 4124-4127.
7. Ubertini, S., Bella, G. and Succi, S., 2003, "Lattice Boltzmann Method on Unstructured Grids: Further Developments", *Phys. Rev. E*, vol. 68, 016701.
8. Ubertini, S., 2003, Computational Fluid Dynamics Through an Unstructured Lattice Boltzmann Scheme, Proceedings of IMECE 2003, ASME paper no. IMECE2003-41194.
9. Dongjoo K., Choiy H. Laminar flow past a sphere rotating in the streamwise direction. *J. Fluid Mech.*, 2002 **461**; 365-386.
10. Johnson T.A., Patel V.C. Flow past a sphere up to a Reynolds number of 300. *J. Fluid Mech.*, 1999, **378**; 19-70.
11. Pruppacher H. R., Le Clair B. P., Hamiliec A. E. Some relations between drag and low pattern of viscous flow past a sphere and a cylinder at low and intermediate Reynolds numbers. *J. Fluid Mech.*, 1970, **44**, 781.
12. Mittal, R. A Fourier-Chebyshev spectral collocation method for simulating flow past spheres and spheroids, *Int. J. Numer. Meth. Fluids*, 1999, **30**; 921-937.
13. Rimon Y., Cheng S. I. Numerical solution of a uniform flow over a sphere at intermediate Reynolds numbers. *Phys. Fluids*, 1969, **12**; 949-959.
14. Guelceat U., Aslan A.R. Accurate 3d viscous incompressible flow calculations with the fem. *Int. J. Numer. Meth. Fluids*, 1997, **25**; 985-1001.
15. S. Filippone, and M. Colajanni. PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices. *ACM Trans. Math. Softw.*, 26:527-550, 2000.

Parallel and Distributed Techniques for Extracting Large Ontologies as a Resource in a Grid Environment

Andrew Flahive¹, Mehul Bhatt¹, Carlo Wouters¹,
Wenny Rahayu¹, David Taniar² and Tharam Dillon³

¹ La Trobe University, Australia
{a.flahive, m.bhatt, c.wouters, wenny}@latrobe.edu.au

² Monash University, Australia
David.Taniar@infotech.monash.edu.au

³ University of Technology Sydney, Australia
tharam@it.uts.edu.au

Abstract. This paper presents a system that extracts small, interconnected sub-ontologies, from larger base ontologies, using parallel and distributed techniques. The system has been designed for use in a grid environment as a grid resource that also uses a High Performance Computer as a grid resource for the main processing. The design presented aids the sharing of information between parties without the need for a central ontology location or local processing plant.

1 Introduction

The next era of the Internet is the *Semantic Web*[14]. In this era, the Web will be structured so as to enable companies to share their data resources with other trusted companies through a fast and popular medium. The Semantic Web is gradually becoming a popular way to share information and resources over the Internet through Web Services. In order to harness this emerging technology more and more Web Services are being developed in an attempt to decentralize systems and share computing ability[11]. Not only software but also hardware resources are being integrated into the sharing domain.

The *Semantic Grid* is basically the same as the Semantic Web, except for the fact that the Semantic Grid shares resources in accordance to certain architectures and standard grid infrastructures. The aim of the Semantic Grid is to standardize the use of these structures so that anyone around the world can create their own programs to interact with remote Grid resources easily and with no misinterpretation.

It is becoming more common for a community of entities, that have powerful computing resources or rare computing facilities, to open their valuable resources to the community so that many people can benefit. The idea of *Grid Computing* is to share computer resources in a highly controlled area. Each Grid location has one or more resources that it shares and maintains locally among the wider Grid community[5]. This allows the less-advantaged users, often located in remote locations, to have the same opportunity as the more advantaged users.

Ontologies have emerged as the current most ideal way to store and allow access to large repositories[10, 12] of complex data and relationships. These ontologies have

the potential to grow very large, in excess of millions of concepts and tens of millions of complex relationships. Ontologies are an appropriate structure to enable efficient storage and retrieval of information, however, the problems lie in the sheer amount of processing required to manipulate them.

Sub-ontologies are valid independent ontologies, known as materialized ontologies, that are specifically extracted from very large base ontologies to meet certain requirements. Throughout 2002 and 2003, the authors of this paper implemented a system to extract sub-ontologies based on a number of *Optimization Schemes* (OS's). This system is referred to as *Materialized OntologyView Extractor* and abbreviated to MOVE[2, 3, 16, 17]. MOVE has addressed the problem, of long processing times, by implementing a distributed architecture for the extraction / optimization of a sub-ontology from a large scale, base ontology.

This paper will show how MOVE has been re-designed to be integrated into the Grid as Grid resource. The main aim is to show how such a system can be better utilized by the public if it were available as a Grid resource. Chapter 2 provides some related work about MOVE and current grid infrastructures and applications. Chapter 3 outlines the architecture of the Grid resource that MOVE will be built into. Chapter 4 is an outline of the intended implementation of the Grid resource, including a simulated evaluation of the whole system. Chapter 5 concludes the paper, highlights the main findings and discusses the future of ontology processing in the Grid environment.

2 Related Work

MOVE enables the user to load in any large base-ontology, input their specific criteria for extraction, and produce an optimized view of the base-ontology[17]. The optimized ontology (sub-ontology) contains only the information that the user selected in their criteria and additional required interconnecting pieces of information. It was envisaged [15] that the processing required to extract these sub-ontologies would be very large, thus the system was designed for a High Performance Computer (HPC).

A number of Optimization Schemes (OS's) have been implemented[2, 3], and many others designed[17, 16, 15], in MOVE to help extract semantically correct information. The two optimizations schemes that have been implemented are:

- (RCOS) The Requirement Consistency OS checks for the consistency of the user specified requirements for the target ontology in the form of the labeling. RCOS itself is a combination of four sub-schemes that check for various forms of consistency.
- (SCOS) The Semantic Completeness OS considers the completeness of the concepts, i.e. if one concept is defined in terms of an another concept, the latter cannot be omitted from the sub-ontology without loss of semantic meaning of the former concept. Currently, SCOS consists of three sub-schemes that check for various forms of semantic completeness.

Other optimization schemes that have been designed include:

- (WFOS) Well Formedness Optimization Scheme contains the proper rules to prevent the user labeling being inconsistent. In some cases it might be possible that

the user requirements may contain certain statements that inevitably lead to a solution that is not a valid ontology. WFOS stops this from happening. WFOS is a combination of five sub-schemes.

- (TSOS) Applying the Total Simplicity OS to an existing solution (along with its requirements specification) will result in the smallest possible solution that is still a valid ontology. TSOS achieves this by working not only on the solution, but also its requirements specification. TSOS consists of three sub-schemes.

MOVE is designed specifically to allow other people to integrate their own OS as well as use any of the currently implemented OS's.

The results from the project [3] show that for large ontologies many processors were required to finish the task in a reasonable amount of time (a few minutes). The reliance of this system on multiple processors exceeded our expectation, and because of this, our project would be out of reach for the average user, with a single processor machine.

Clustered computing, at its most basic level, involves two or more computers serving as a single resource[4]. Grid computing is more than this. The resources in the Grid are independently controlled by each local resource provider. These resources are pooled together over some sort of network and can be used by any authenticated user or application on that network. With the emergence of the Grid and Grid Technology[8], it was decided that it would be wise to redesign the MOVE system for the Grid as a Grid resource so that many more users could one day make use of it.

Transferring this highly parallelized system onto the Grid as a Grid resource, will allow people from remote locations to extract sub-ontologies from large base ontologies without requiring their own HPC facility. Users could use the HPC to perform the intensive ontology processing tasks from their location just as if they had their own HPC.

3 System Architecture

The general architecture of the Grid consists of four basic layers[7], the application layer, the middleware layer, the resources layer and the network layer. This architecture is shown in Figure 3.1. The application layer where the user or the initiator runs an program that may require a grid resource. The middleware layer handle the requests from the application programs and determines what resources are required, where to get the resources, and connects the resource to the application. The middleware layer also handles things like security, resource brokering, scheduling and other resource management tasks.

The third layer is where MOVE is located, in the Resources layer. These resources may belong on different platforms, provide different functions, facilities and services. The network layer consists of wires and cables, switches, hubs and routers that connect all of the other layers together.

The implementation of MOVE as a grid resource would not be possible without a pre-defined grid architecture. The best and most common existing grid system is the globus toolkit[13]. It follows the conventions of proper grid architecture by following the Open Grid Services Architecture (OGSA)[6] and the Open Grid Services Infrastructure (OGSI)[9]. It is these standards and conventions that design of the system for the

grid adheres to. These standards define the methods to help set up the communication between the system and available resources (like a HPC). The application programming interface and software development kits as defined in [5] have been crucial for the successful design of the interface to the project in the Grid environment. However, the Globus Toolkit currently requires a third party implementation of a resource broker to search and discover appropriate resources to use.

Open Grid Services Infrastructure (OGSI)[9] is part of the Open Grid Services Architecture (OGSA)[6] that has developed through the Global Grid Forum (GGF)[1] to define Grid Services. Grid Services are Web services that conform to a specific set of conventions. OGSI specifies a set of 'service primitives' that establish a nucleus of behavior common to all Grid/Web services that can be leveraged by system-level services.

Shown in Figure 3.2 is the MOVE system deployed as a resource. MOVE is contained within the OGSI nucleus. This ensures that the Grid resource adhere's to the proper standards. The OGSA handles the overall interface between MOVE and the other applications and resources. Using the OGSA as a means of interaction allows other 3rd party programs to use the resource in a standard way and for MOVE to employ a Grid resource of it's own (The HPC).

One of the major architectural changes that MOVE has to undertake is the addition of a 'Resource Determination' module. This module is shown in Figure 3.2 as reference point 3. The purpose of this module is to determine what resources are required by MOVE to complete the ontology processing as efficient as possible. In the past we just told the HPC how many processors it should use to complete the processing.

The module's main purpose is interface between MOVE and the Resource Broker. The module determines the optimum number of processors it should use and the minimum specifications of the processors and provides the Resource Broker with this

Layers of the Grid

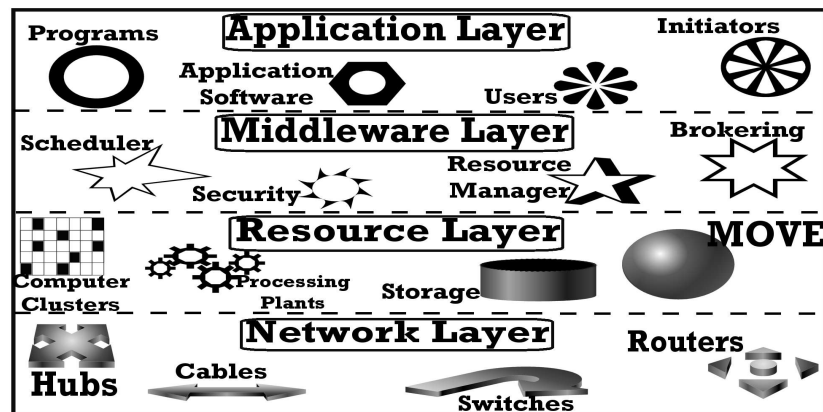


Fig. 3.1. Basic Grid Architecture

information (point 4 in Figure 3.2). Based on this information the resource broker finds suitable resources (a HPC), transfers the work to be done and sends the data back to move (point 5 in Figure 3.2).

MOVE has been designed using the OGSA. The OGSA acts as the main interface standard between resources, resource brokers and applications. The OGSI is the blueprint that the Grid Resource is constructed on. The 'Resource Determination' module interacts between the resource Broker and MOVE using proper OGSA methods to transport the information. MOVE is then able to collect the base-ontology from the application, processes the ontology as best it knows how and returns the solution to the initiating application.

4 Experimental Simulation

MOVE is designed to be implemented on a server that can be accessed through the grid network and have access to other grid resources. MOVE has been designed to use a HPC as a grid resource just as it would if the program were run at the HPC facility. It performs the processing tasks using a number of available processors at the HPC facility and returns the results back to MOVE.

A user's local ontology manipulating program requires a sub-ontology to enable faster searching and more focused information. The program then searches 'The Grid' for a suitable resource that matches it's requirements through a resource broker. Once MOVE is discovered, the program sends the ontology to MOVE which analyzes the ontology to be extracted. MOVE gathers the required information from the user and proceeds to start the Optimization Schemes.

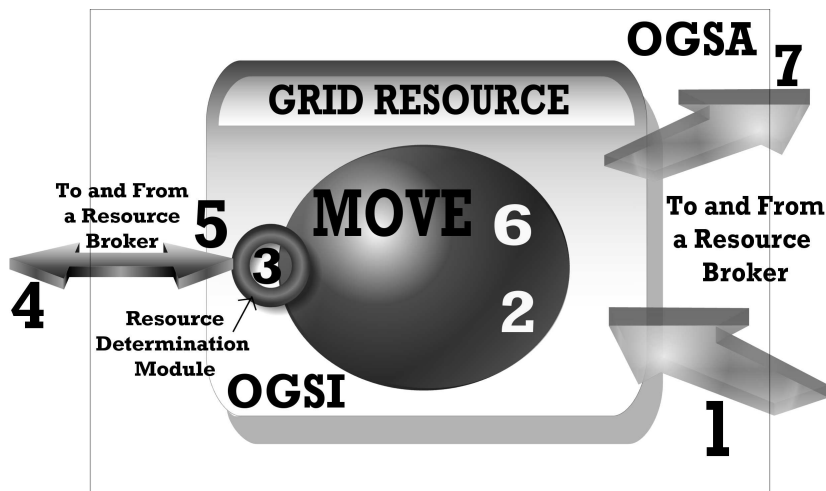


Fig. 3.2. MOVE Resource Architecture

At this point MOVE realizes that the task is too big for the local resources it houses and therefore sets out to discover the required number of processors. Once a suitable HPC has been discovered, MOVE uses the processors as if it were it's own. MOVE completes the task and presents the extracted sub-ontology to the remote program. Ideally all of this happens without the user realizing the the bulk of the processing has happened at a distant location. The user may only notice a slight lag as the program itself if MOVE has to search and discover appropriate grid resources to use.

Figure 4.3 shows the experimental results when running MOVE using the Semantic Completeness Optimization Scheme (SCOS). It shows the results from the local Cluster Environment (HPC) as well as the simulated results from MOVE as implemented as a Grid Resource in Grid Environment using the HPC as a Grid Resource.

The graphs in Figure 4.3 indicate that there is some loss between the original local cluster system and the new Grid system. The lower of the two plots of each pair of the same sized ontology, shows the time taken when the system was deployed locally on the HPC cluster. The higher plot of each of the pairs shows the expected time taken for the Grid implementation of the same system. The system is deployed on a server and uses the same HPC but as a Grid resource instead of a local cluster. The only difference, in the results, is the time taken to transfer the information across the medium between the server and the HPC. The time lag is consistent, depending on the size of the ontology that needs to be transferred. As the ontology grows larger, so too does the gap between the local version of the system and the Grid version.

The graphs indicates a trade-off that may need to be considered between allowing access to more users or timely results. The systems administrator may choose to restrict access to the MOVE resource to only those users with a high bandwidth connection during peak times. They may also like to restrict the size of the ontology during peak

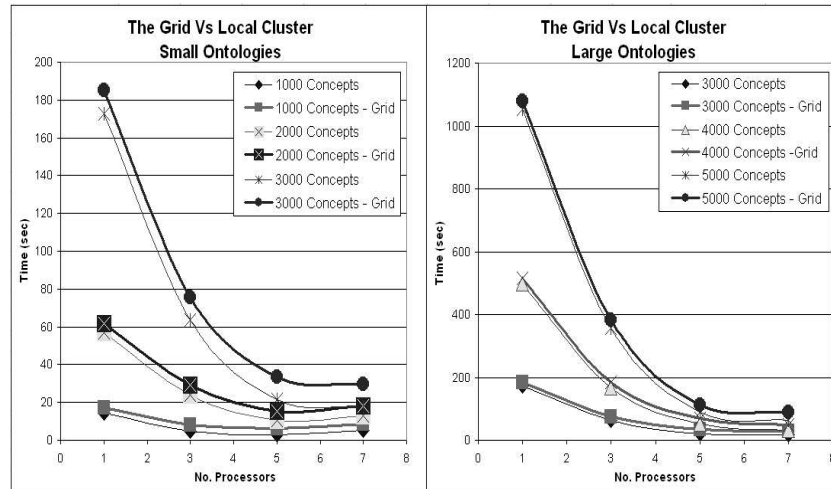


Fig. 4.3. Grid Evaluation

times so that the resource can be more efficiently shared between users. These are all scheduling issues. By providing this evaluation, future schedulers may be able to be more effectively tuned to suit the specifications of the advertised resource and the requirements of the user.

5 Conclusion

This paper presented the design and simulation of a system that uses parallel and distributed techniques on the Grid as a Grid resource. The system has been transformed into a Grid resource that uses a HPC as a Grid resource to perform the required tasks. The preliminary results shown in this paper indicate that the MOVE system can be implemented on the Grid as a resource in a Grid environment and prove successful. This means that under-resourced users can perform the most computationally extensive tasks on large ontologies. If the remote user hadn't have had access to MOVE as a Grid resource then they would not have been able to perform the extensive ontology processing required.

More projects like this are needed to make full use of the Grid Environment that the world is moving toward. As the Grid environment expands and evolves more and more resources will be developed for use on the Grid. There are endless ways that the grid can help the general community. In the future MOVE will be extended to include other tasks like merging large ontologies, synchronizing them and perform other general maintenance operations. The need for ontology processing in a parallel environment will be essential to allow all processing of large ontologies. The Grid will bring the computing power to the masses, as few people have the resources to perform these tasks on their own.

References

1. The Global Grid Forum. <http://www.gridforum.org/>, 2004.
2. M. Bhatt, A. Flahive, C. Wouters, W. Rahayu and D. Taniar. A Distributed Approach to Sub-Ontology Extraction. In *IEEE CS Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA'04)* (Fukuoka, Japan, March 2004).
3. M. Bhatt, C. Wouters, A. Flahive, W. Rahayu and D. Taniar. Semantic completeness in sub-ontology extraction using distributed methods. In *International Conference on Computational Science and its Applications (ICCSA'04)* (Italy, 2004), vol. ICCSA'04, Springer-Verlag Heidelberg.
4. C. Bookman. *Linux Clustering: Building and Maintaining Linux Clusters*. New Riders, 2003.
5. Foster, I.T.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, Springer-Verlag (2001) 1–4
6. I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG*, Global Grid Forum, June 22, 2002.
7. G. Francois. Grid Cafe. <http://gridcafe.web.cern.ch/>, 2004.

8. S. Mukherjee, J. C. A. Mustafi. Grid Computing: The Future of Distributed Computing for High Performance Scientific and Business Applications. In *IWDC (2002)*, S. Das and S. Bhattacharya, Eds., vol. 2571 of *Lecture Notes In Computer Science*, Springer-Verlag Heidelberg, pp. 339–342.
9. ogsi wg. Open Grid Services Infrastructure (OGSI) Version 1.0. Version 1.0 ed. Global Grid Forum, 2003.
10. J. Pan, S. Cranefield and D. Carter. A Lightweight Ontology Repository. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems (2003)*, pp. 632–638.
11. B. Plateau. The grid: Challenges and research issues. In *Advances in Computing Science - ASIAN 2002. Internet Computing and Modeling, Grid Computing, Peer-to-Peer Computing, and Cluster Computing : 7th Asian Computing Science Conference, Hanoi, Vietnam, December 4-6, 2002. Proceedings (2002)*, J. H. G. Goos and van Leeuwen, Eds., vol. Volume 2550 / 2002, Springer-Verlag Heidelberg, pp. 13–14.
12. Prof. D. Sleeman, Dr. D. Robertson, Dr. S. Potter and Dr. M. Schorlemmer. Ontology Extraction for Distributed Environments, in Knowledge Transformation for the Semantic Web. In *Frontiers in Artificial Intelligence and Applications 95*. IOS Press, 2002, pp. 80–91.
13. University of Chicago. The Globus Alliance. <http://www.globus.org/>, 2004.
14. W. 3. C. org. W3C World Wide Web Consortium. *W3C Website* (2004). <http://www.w3c.org>.
15. C. Wouters, T. Dillon, W. Rahayu and E. Chang. A Practical Walkthrough of the Ontology Derivation Rules. *DEXA2002* (2002), 259–268.
16. C. Wouters, T. Dillon, W. Rahayu, E. Chang and R. Meersman. Ontologies on the MOVE. In *Proceedings of 9th international conference on Database Systems for Advanced Application (DASFAA 2004)* (April 2004), Lecture Notes in Computer Science, Springer Verlag.
17. C. Wouters, T. Dillon, W. Rahayu, E. Chang and R. Meersman. A Practical Approach to the Derivation of Materialized Ontology Views. In *Web Information Systems*, T. D. and R. W., Eds. Idea Group Publishing, 2004, pp. 191–226.

Adaptive Fuzzy Active Queue Management

Mahdi Jalili-Kharaajoo, Mohammadreza Sadri and Farzad Habibipour Roudsari

Young Researchers Club, Islamic Azad University and Iran Telecommunication Research
Center, Tehran, Iran mahdijalili@ece.ut.ac.ir

Abstract. As an enhancement mechanism for the end-to-end congestion control, Active Queue Management (AQM) can keep smaller queuing delay and higher throughput by proposing fully dropping the packets at the intermediate nodes. Comparing with RED algorithm, although PI controller for AQM designed by Hollot improves the stability, it seems other methods to design of robust controllers may lead to better results. Moreover, the transient performance of PI controller is not perfect, such as the regulating time is so long. In order to overcome this drawback, in this paper, a novel adaptive fuzzy logic based controller is designed for Active Queue Management (AQM) in TCP/AQM networks. From control point of view, it is rational to regard AQM as a typical regulation system. Recently many AQM algorithms have been proposed to address performance degradations of end-to-end congestion control. However, these AQM algorithms show weaknesses to detect and control congestion under dynamically changing network situations. A simulation study over a wide range of IP traffic conditions shows the effectiveness of the proposed controller in terms of the queue length dynamics, the packet loss rates, and the link utilization.

1 Introduction

A typical information exchange over the Internet is guaranteed by several intermediate nodes (routers) which direct packets originated by the sender to the receiver over links with limited bandwidths. Each router has a finite buffer for storing packets exceeding the total capacity of the link. When the packet net flow exceeds the buffer size the link becomes congested causing a so-called packet drop to occur. Namely, the packet is lost and the sender required to transmit it again.

TCP congestion control mechanism, while necessary and powerful, are not sufficient to provide good service in all circumstances, specially with the rapid growth in size and the strong requirements to Quality of Service (QoS) support, because there is a limit to how much control can be accomplished at end system. It is needed to implement some measures in the intermediate nodes to complement the end system congestion avoidance mechanisms. Active Queue Management (AQM), as one class of packet dropping/marketing mechanism in the router queue, has been recently proposed to support the end-to-end congestion control in the Internet [1-5]. It has been a very active research area in the Internet community. The goals of AQM are (1) reduce the average length of queue in routers and thereby decrease the end-to-end delay experienced by packets, and (2) ensure the network resources to be used efficiently by reducing the packet loss that occurs when queues overflow. AQM highlights the

tradeoff between delay and throughput. By keeping the average queue size small, AQM will have the ability to provide greater capacity to accommodate nature-occurring burst without dropping packets, at the same time, reduce the delays seen by flow, this is very particularly important for real-time interactive applications. RED [6,7] was originally proposed to achieve fairness among sources with different burst attributes and to control queue length, which just meets the requirements of AQM. However, many subsequent studies verified that RED is unstable and too sensitive to parameter configuration, and tuning of RED has been proved to be a difficult job [8-10].

Fuzzy logic controllers have been developed and applied to nonlinear system for the last two decades [11]. The most attractive feature of fuzzy logic control is that the expert knowledge can be easily incorporated into the control laws [12].

The intuition and heuristic design is not always scientific and reasonable under any conditions. Of course, since Internet is a rather complex huge system, it is very difficult to have a full-scale and systematic comprehension, but importance has been considerably noted. The mathematical modeling of the Internet is the first step to have an in-depth understanding, and the algorithms designed based on the rational model should be more reliable than one original from intuition. In some of the references, the nonlinear dynamic model for TCP flow control has been utilized and some controllers like PI and Adaptive Virtual Queue Algorithm have been designed for that [13-17]. In the research, we will apply a fuzzy controller to design the AQM system for congestion avoidance. The simulation results show the superior performance of the proposed controller in comparison with classic PI controller.

2 TCP flow control model

In [13], a nonlinear dynamic model for TCP flow control has been developed based on fluid-flow theory. This model can be stated as follows

$$\frac{dW(t)}{dt} = \frac{1}{R(t)} - \frac{W(t)W(t-R(t))}{2R(t)}p(t-R(t)); \frac{dq(t)}{dt} = \frac{N(t)}{R(t)}W(t) - C(t) \quad (2.1)$$

The definition of the parameters can be found in [2,13].

We believe that the AQM controller designed with the simplified and inaccurate linear constant model should not be optimal, because the actual network is very changeable; the state parameters are hardly kept at a constant value for a long time. Moreover, the equations (1) only take consideration into the fast retransmission and fast recovery, but ignore the timeout mechanism caused by lacking of enough duplicated ACK, which is very usual in burst and short-lived services. In addition to, there are many non-respective UDP flows besides TCP connections in networks; they are also not included in equations (1). These mismatches in model will have negative impact on the performance of controller designed with the approach depending with the accurate model. For the changeable network, the robust control should be an appropriate choice to design controller for AQM. The above nonlinear and time-varying system was approximated as a linear constant system by small-signal linearization about an operating point [2,5,13] (Fig. 1), where

$$K(t) = \frac{[R(t)C(t)]^3}{[2N(t)]^2}; T_1(t) = R(t); T_2(t) = \frac{R^2(t)C(t)}{2N(t)} \quad (2.2)$$

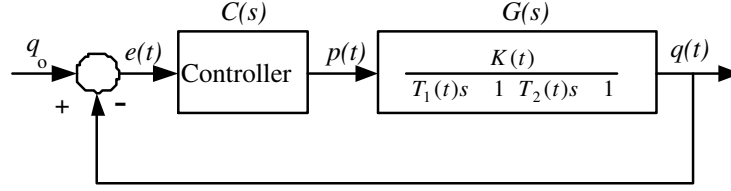


Fig. 2.1. Block diagram of AQM control system

To describe the system in state space form, suppose that $x_1 = e$; $x_2 = \frac{de}{dt}$, so the plant depicted in Fig. 1 is described by a second order system as

$$\frac{dx_1}{dt} = x_2; \frac{dx_2}{dt} = -a_1x_1 - a_2x_2 - b + F \quad (2.3)$$

where

$$a_1 = \frac{1}{T_1T_2}; a_2 = \frac{T_1 + T_2}{T_1T_2}; b = \frac{K}{T_1T_2}; F = \frac{d^2q_0}{dt^2} + \frac{T_1 + T_2}{T_1T_2} \frac{dq_0}{dt} + \frac{q_0}{T_1T_2} \quad (2.4)$$

3 Design of fuzzy controller

Fuzzy logic control (FLC) has been demonstrated to solve some practical problems that have been beyond the reach of conventional control techniques. Fuzzy logic control is a knowledge-based control that uses fuzzy set theory, fuzzy reasoning and fuzzy logic for knowledge representation and inference [11,12]. The apparent success of FLC can be attributed to its ability to incorporate expert information and generate control surfaces whose shape can be individually manipulated for different regions of the state space with virtually no effects on neighboring regions.

In this paper, a fuzzy system consisting of a fuzzifier, a knowledge base (rule base), a fuzzy inference engine and defuzzifier will be considered. The knowledge base of the fuzzy system is a collection of fuzzy IF-THEN rules. Fuzzy logic control is ideal for the AQM problem, since there is no complete mathematical model. However, human experience and experimental results can be used in the control system, design.

The controller has two inputs, the error (e) and its derivative (\dot{e}) and the control input (p). Five triangular membership functions are defined for speed error (Fig. 2), namely, Negative Large (NL), Negative Small (NS), Zero, Positive Small (PS), and Positive Large (PL). Similarly three triangular membership functions are defined for derivative of the error (Fig. 3) and there are as follows, Negative Small (NS), Zero, and Positive Small (PS). Also five triangular membership functions are defined for the

control input (Fig. 4) and there are Zero, Small, Medium, Large and Very Large. The complete fuzzy rules are shown in Fig. 5. The first rule is outlined below

Rule 1: If (e) is *PL* AND (\dot{x}) is *Zero*, THEN (p) is *Large*.

The rest of the rules are derived similarly. The label names used here give an intuitive sense of how the rules apply. Through experimentation and tuning of the membership functions it was determined that the number of rules was sufficient to encompass all realistic combinations of inputs and outputs. This fuzzy logic controller is implemented using product inference and a center-average defuzzifier.

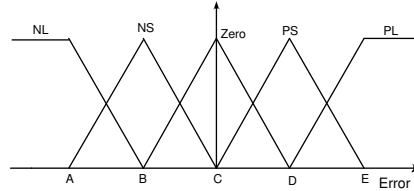


Fig. 3.2. Error membership function

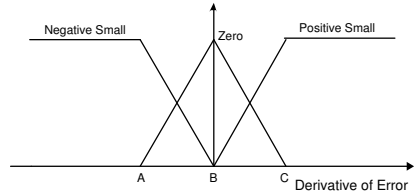


Fig. 3.3. Membership function for the derivative of Error

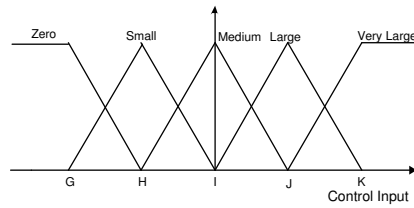


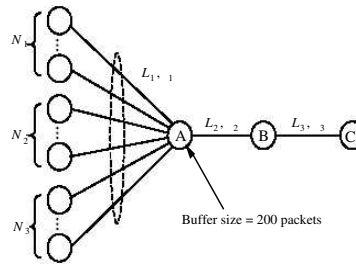
Fig. 3.4. Control input membership function

$e \backslash e$	NS	ZERO	PS
NL	ZERO	ZERO	ZERO
NS	SMALL	SMALL	SMALL
ZERO	ZERO	ZERO	ZERO
PS	SMALL	LARGE	MEDIUM
PL	MEDIUM	VERY LARGE	LARGE

Fig. 3.5. Fuzzy rules

4 Simulation results

The network topology used for simulation, is depicted in Fig. 6 [2,5]. The only bottleneck link lies between node A and node B. the buffer size of node A is 200 packets, and default size of the packet is 350 bytes. All sources are classed into three groups. The first one includes N_1 greedy sustained FTP application sources, the second one is composed of N_2 burst HTTP connections, each connection has 10 sessions, and the number of pages per session is 3. The thirds one has N_3 UDP sources, which follow the exponential service model, the idle and burst time are 10000msec and 1000msec, respectively, and the sending rate during "on" duration is 40kbps. We introduced short-lived HTTP flows and non-responsive UDP services into the router in order to generate a more realistic scenario, because it is very important for a perfect AQM scheme to achieve full bandwidth utilization in the presence of noise and disturbance introduced by these flows. The links between node A and all sources have the same capacity and propagation delay pair (L_1, τ_1) . The pair (L_2, τ_2) and (L_3, τ_3) define the parameter of links AB and BC, respectively.

**Fig. 4.6.** The simulation network topology

In the first study, we will use the most general network configuration to testify whether the proposed Adaptive Fuzzy Logic Controller (FLC) can reach the goals of

AQM, and freely control the queue length to stabilize at the arbitrary expected value. Therefore, given that $(L_1, \tau_1) = (10Mbps, 15ms)$, $(L_2, \tau_2) = (15Mbps, 15ms)$, $(L_3, \tau_3) = (45Mbps, 15ms)$, $N_1 = 270$, and $N_2 = N_3 = 0$. Let the expected queue length equal to 75 packets. The instantaneous queue length using the proposed FLC is depicted in Fig. 7. After a very short regulating process, the queue settles down its stable operating point. RED algorithm is unable to accurately control the queue length to the desired value [7,9]. The queue length varies with network loads. The load is heavier the queue length is longer. Attempting to control queue length through decreasing the interval between high and low thresholds, then it is likely to lead queue oscillation. To investigate the performance of the proposed FLC, we will compare the results with that of PI controller designed in [13]. The queue evaluation using PI controller is shown in Fig. 8. As it can be seen FLC acts much better than PI one.

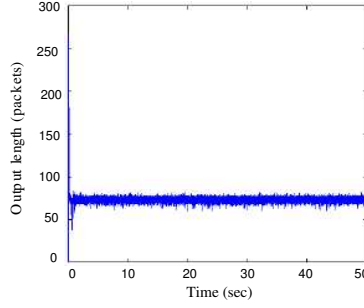


Fig. 4.7. Queue evaluation (FLC)

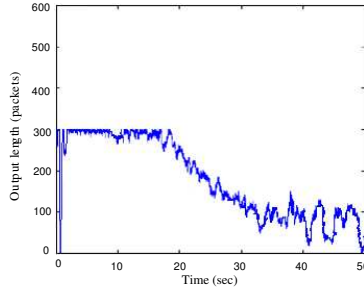


Fig. 4.8. Queue evaluation (PI)

Finally, we evaluate the integrated performance of the the proposed controller using one relatively real scenario, i.e., the number of active flows is changeable, which has 270 FTP flows, 400 HTTP connections and 30 UDP flows. Figs. 9 and 10 show the

evaluation of queue controlled by FLC and PI controllers, respectively. It is clear that the integrated performance of FLC controller, namely transient and steady state responses is superior to that of PI controller. The FLC controller is always keeping the queue length at the reference value, even if the network loads abruptly change, but PI controller has the inferior adaptability. In other words, the former is more powerful, robust and adaptive than the later one, which is in the favor of achievement to the objectives of the AQM policy.

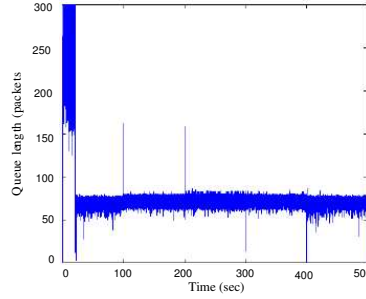


Fig. 4.9. Queue evaluation (FLC) for (FTP+UDP+HTTP) queue

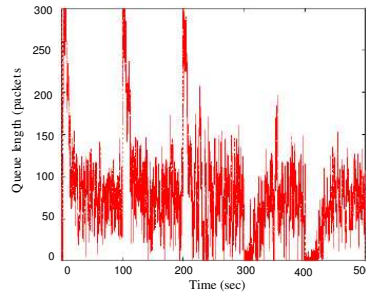


Fig. 4.10. Queue evaluation (PI) for (FTP+UDP+HTTP) queue

5 Conclusion

In this paper, an adaptive fuzzy logic based controller was applied to TCP/AQM networks for the objective of queue management and congestion avoidance. For this purpose, a linearized model of the TCP flow was considered. We took a complete comparison between performance of the proposed FLC and classical PI controller under various scenarios. The conclusion was that the integrated performance of FLC was superior to that of PI one.

References

1. Barden, B. et al., Recommendation on queue management and congestion avoidance in the internet, REC2309, 1998.
2. Jalili-Kharaajoo, M., Application of robust fuzzy adaptive second-order sliding-mode control to active queue management, LNCS, 2957, 109-119, 2004.
3. S. Ryu, C. Rump and C. Qiao, A Predictive and Robust Active Queue Management for Internet Congestion Control, in Proc. ISCC'03, 2003.
4. S. Ryu, C. Rump, and C. Qiao. Advances in Internet congestion control. IEEE Communication Survey and Tutorial, 2002.
5. R. Fengyuan, et al., A Robust AQM algorithm based on Sliding Mode Variable Structure Control. in Proc. INFOCOM'2002, 21, 13-20, 2002.
6. Floyd, S. and Jacobson, V., Random early detection gateway for congestion avoidance, IEEE/ACM Trans. Networking, 1993.
7. C. V. Hollot, V. Misra, D. Towsley, and W. Gong. A control theoretic analysis of RED. in Proc. of INFOCOM'2001, 1510-1519, 2001.
8. Firoiu, V. and Borden, M., A study of active queue management for congestion control, in Proc. INFOCOM, 2000.
9. May, M., Bonald, T. and Bolot, T., Analytic evaluation of RED performance, in Proc. INFOCOM, 2000.
10. S. Floyd and V. Paxson. Difficulties in simulating the Internet, IEEE/ACM Transactions on Networking, 9(4), 392-403, 2001.
11. Zadeh, L.A., Fuzzy sets, Inf. Control (1965), 338-353.
12. Jalili-Kharaajoo, M., Improvement of second order sliding mode control applied to position control of induction motors using fuzzy logic, LNAI, 2715, 2003.
13. Misra, V., Gong, W.B. and Towsley, D., Fluid-based analysis of network of AQM routers supporting TCP flows with an application to RED, in Proc. ACM/SIGCOMM, 2000.
14. Hollot, C., Misra, V., Towsley, D. and Gong, W.B., On designing improved controllers for AQM routers supporting TCP flows, in Proc. INFOCOM, 2001.
15. Misra, V., Gong, W.B. and Towsley, D., Analysis and design an adaptive virtual queue (AVQ) algorithm for active queue management, in Proc. ACM/SIGCOMM, 2001.
16. Kelly, F.P., Maulloo, A. and Tan, D., Rate control in communication networks, Journal of the Operation Research Society, 49, 237-252, 1998.
17. Athuraliya, S., Lapsley, D.E. and Low, S.H., Random early marking for internet congestion control, in Proc. Globecom, 1999.

Inversion and Division Architecture in Elliptic Curve Cryptography over $GF(2^n)$ (Not accepted yet)

Jun-Cheol Jeon¹, Kyo-Min Ku², Kee-Young Yoo¹

¹ Department of Computer Engineering at Kyungpook National University
Daegu, Korea, 702-701
jcjeon33@infosec.knu.ac.kr, yook@knu.ac.kr

² Mobilab.Co., Ltd
Plus B/D 4F 952-3, Dongchun-dong, Buk-gu, Daegu, Korea, 702-250
kmku@mobilab.co.kr

Abstract. The two elliptic curve operations are the Add and Double, which are computed by field arithmetic operations, such as additions, modular multiplications, modular squarings and divisions. The addition operation for field elements is trivial and squaring is so much faster than regular multiplication that it can be ignored in rough comparisons of the timings. The important contributors to the run time are divisions. Thus we propose efficient division architecture by recursive AB^2 multiplication algorithm based on Cellular Automata (CA) in Elliptic curve cryptosystems (ECC) over $GF(2^n)$. The proposed architectures can be used in the effectual hardware design of coprocessor for ECC since they have high regularity and a reduced latency.

1 Introduction

Elliptic Curve Cryptosystem was introduced by Victor Miller and Neal Koblitz in 1985. ECC proposed as an alternative to established public-key cryptosystem such as RSA and ElGamal, have recently gained a lot attention in industry and academia [1]Miller86. The main reason for attractiveness of ECC is the fact that there is no sub-exponential algorithm known to solve the discrete logarithm problem on a properly chosen elliptic curve.

The two elliptic curve operations that are most relevant to the complexity of multiplying a group element by a constant are the Add and Double operations, which are composed of field arithmetic operations such as additions, modular multiplications, modular squarings and divisions. The mostly cost field arithmetic operation is division [3]. Moreover the division can be computed by applying AB^2 multiplication repeatedly. Fast computation of a division operation can generally be classified into two approaches: a faster or smaller architecture design or noble algorithm generation, and this paper focused on the former approach.

Finite field $GF(2^n)$ arithmetic operations have recently been applied in a variety of fields, including cryptography and error-correcting codes [4]. A number of modern public key cryptography systems and schemes, for example, Diffie-Hellman key pre-distribution, the Elgamal cryptosystem, and ECC, require the operations of division,

exponentiation, and inversion, which are normally implemented using AB or AB^2 multiplier [5]. Wei designed a cellular power-sum circuit with a latency of $n(n-1)$ critical path of $n(T_{AND} + T_{3XOR})$ over $GF(2^n)$ [12]. Wang proposed parallel-in parallel-out $AB^2 + C$ architecture [13].

Cellular automata, which is introduced by Von Neumann in [6] has been accepted as a good computational model for the simulation of complex physical systems. It has been used for various applications, such as parallel processing computations and number theory etc. Zhang in [7] proposed architecture with programmable cellular automata and a cell complexity of $3SWITCH + 2XOR$, while Choudhury in [8] designed an LSB multiplier based on a CA with a cell complexity of $2AND + 2XOR$.

This paper proposes architectures for modular AB^2 multiplications and divisions based on CA architecture. We focused on the architectures in ECC, which uses restricted irreducible polynomials, specially, trinomials. The division structure has a time complexity of $n(n-1)(T_{AND} + T_{XOR})$ and hardware complexity of $n(AND + XOR + MUX + 3REGISTER) + 2XOR$. Our architectures offer a fair area/time performance trade-off.

The remainder of this paper is organized as follows. The conceptional background, including finite fields, ECC, and CA are described in section 2. Section 3 presents the proposed division architecture by recursive AB^2 multiplication architecture based on CA. In section 4, we presents discussion and performance analysis. Finally, section 5 gives concluding remarks.

2 Preliminaries

In this section, we present mathematical background in the finite field and ECC, and the characteristics and properties of CA.

2.1 Finite Fields

A finite field or Galois Field(GF), which is a set of finite elements, can be defined by commutative law, associative law, and distributive law and facilitates addition, subtraction, multiplication, and division. Numbers of architectures have already been developed to construct low complexity bit-serial and bit-parallel multiplications using various irreducible polynomials to reduce the complexity of the modular multiplication. Since a polynomial basis operation does not require a basis conversion, it can be readily matched to any input or output system. Also, due to its regularity and simplicity, the ability to design and expand into high-order finite fields with polynomial basis is easier to realize than with other basis operations [9].

The finite field $GF(2^n)$ can be viewed as a vector space of dimension n over $GF(2)$. That is, there exists a set of n elements $\{1, \alpha, \dots, \alpha^{n-2}, \alpha^{n-1}\}$ in $GF(2^n)$ such that each $A \in GF(2^n)$ can be written uniquely in the form $A = \sum A_i \alpha^i$, where $A_i \in \{0, 1\}$. This section provides one of the most common based of $GF(2^n)$ over $GF(2)$ [9], polynomial bases. Let $f(x) = x^n + \sum_{i=0}^{n-1} f_i x^i$, where $f_i \in \{0, 1\}$, for $i = 0, 1, \dots, n-1$, be an irreducible polynomial of degree n over $GF(2)$. For each

irreducible polynomial, there exists a polynomial basis representation. In such a representation, each element of $GF(2^n)$ corresponds to a binary polynomial of degree less than n . This is, for $A \in GF(2^n)$ there exist n numbers $A_i \in \{0, 1\}$ such that $A = A_{n-1}\alpha^{n-1} + A_{n-2}\alpha^{n-2} + \dots + A_1\alpha + A_0$.

The field element $A \in GF(2^n)$ is usually denoted by the bit string $(A_{n-1} \dots A_1 A_0)$ of length n . The following operations are defined on the elements of $GF(2^n)$ when using a polynomial representation with irreducible polynomial $f(x)$. Assume that $A = (A_{n-1} \dots A_1 A_0)$ and $B = (B_{n-1} \dots B_1 B_0)$. 1) Addition: $A + B = C = (C_{n-1} \dots C_1 C_0)$, where $C_i = (A_i + B_i) \bmod 2$. That is, addition corresponds to bitwise exclusive-or. 2) Multiplication: $A \cdot B = C = (C_{n-1} \dots C_1 C_0)$, where $C(x) = \sum_{i=0}^{n-1} C_i x^i$ is the remainder of the division of the polynomial $(\sum_{i=0}^{n-1} A_i x^i)(\sum_{i=0}^{n-1} B_i x^i)$ by $f(x)$. In many applications, such as cryptography and digital communication applications, the polynomial basis is still the most popularly employed basis [9]. In the following, we confine our attention to the computations that use the polynomial basis.

2.2 Elliptic Curve Cryptosystems

In ECC, computing kP is the most important operation, where k is an integer and P is a point on the elliptic curve. This operation can be computed using the addition of two points k times. ECC can be done with at least two types of arithmetic, each of which gives different definitions of multiplication [10]. The types of arithmetic are 1) \mathbb{Z}_p arithmetic(modular arithmetic with a large prime p as the modulus) 2) $GF(2^n)$ arithmetic, which can be done with shifts and exclusive-ors. This can be thought of as modular arithmetic of polynomials with coefficients $\bmod 2$.

We focused on $GF(2^n)$ arithmetic operation. Let $GF(2^n)$ be a finite field of characteristic. Then the set of all solution to the equation $E : y^2 + xy = x^3 + a_2x^2 + a_6$, where $a_2, a_6 \in GF(2^n)$, $a_6 \neq 0$, together with special point called the point at infinity O is a non-supersingular curve over $GF(2^n)$. Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points in $E(GF(2^n))$ given in affine coordinates [12]. Assume $P_1, P_2 \neq O$, and $P_1 \neq -P_2$. The sum $P_3 = (x_3, y_3) = P_1 + P_2$ is computed as follows: If $P_1 \neq P_2$ (called point addition) Then $\lambda = (y_1 + y_2)/(x_1 + x_2)$, $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a_2$, $y_3 = (x_1 + x_3)\lambda + x_3 + y_1$. If $P_1 = P_2$ (called point doubling) Then $\lambda = y_1/x_1 + x_1$, $x_3 = \lambda^2 + \lambda + a_2$, $y_3 = (x_1 + x_3)\lambda + x_3 + y_1$.

>From these formulas, we can determine the number of field operations required for each kind of elliptic curve operation. The addition algorithm for field elements is trivial: the two blocks of bits are simply combined with the bitwise *xor* operation. Because our field has characteristic 2, subtraction is the same as addition. The squaring can be substituted by multiplication. Multiplication of field elements uses the same shift-and-add algorithm as is used for multiplication of integers, except that the "add" is replaced with "xor". The important contributors to the run time are divisions.

2.3 Cellular Automata

CA is a collection of simple cells arranged in a regular fashion. CAs can be characterized based on four properties: the cellular geometry, neighborhood specification, number of states per cell, and rule to compute to successor state. The next state of a CA

depends on the current state and rules [6]. CA can also be classified as linear or non-linear. If the neighborhood is only dependent on an *XOR* operation, the CA is linear, whereas if it is dependent on another operation, the CA is non-linear. If the neighborhood is only dependent on an *EXOR* or *EXNOR* operation, then the CA can also be referred to as an additive CA.

Among additive CAs, CA of which dependency on neighbors is shown only in terms of *XOR* is called a non-complemented CA, and the corresponding rule is called the non-complemented rule. If the dependency on neighbors is shown only in terms of *XNOR*, the CA is called a complemented CA, and the corresponding rule is called the complemented rule. A hybrid CA can be subject to either the complemented or non-complemented rule. Also, there are the 1-dimensional, 2-dimensional, and 3-dimensional CAs according to the structure of arrangement of cells [6].

Furthermore, if the same rule applies to all the cells in a CA, the CA is called a uniform or regular CA, whereas if different rules apply to different cells, it is called a hybrid CA. And in the structure of CAs, the boundary conditions should be taken into consideration, where the boundary conditions incur since there exist no left neighbor of the leftmost cell and right neighbor of the rightmost cell among the cells composing CA. According to the conditions, they are divided into three types.

1) Null Boundary CA (NBCA): CA of which left neighbor of the leftmost cell and right neighbor of the rightmost cell are regarded to be '0'.

2) Periodic Boundary CA (PBCA): CA of which leftmost cell and rightmost cell are regarded to be adjacent to each other, i.e., the left neighbor of the leftmost cell becomes the rightmost cell, and the right neighbor of the rightmost cell becomes the leftmost cell.

3) Intermediate Boundary CA (IBCA): The left neighbor of the leftmost cell is regarded to be the second right neighbor, and right neighbor of the rightmost cell is regarded to be the second left neighbor.

The next state transition for the i th cell can be represented as a function of the present states. If next state determine by 2 bits shifting to the left, then it can be expressed as $Q_i(t+1) = Q_{i-2}(t)$, ($0 \leq i \leq n-1$), where $Q(t+1)$ denotes the next state for cell $Q(t)$. The proposed structure carries out efficient modular reduction based on using irreducible trinomials.

3 Proposed Architecture based on PBCA

In this section, we propose efficient AB^2 multiplication architecture and division architecture by applying recursive AB^2 multiplication architecture.

3.1 Proposed AB^2 Multiplication Architecture

This subsection presents efficient AB^2 multiplication algorithm using irreducible trinomials based on PBCA by the MSB-first method on $GF(2^n)$. Let us suppose that $A(x)$ and $B(x)$ are the elements on $GF(2^n)$. Then the two polynomials $A(x), B(x)$ are as follows:

$$A(x) = A_{n-1}x^{n-1} + \dots + A_1x^1 + A_0, B(x) = B_{n-1}x^{n-1} + \dots + B_1x^1 + B_0 \quad (3.1)$$

By equation (1), we have

$$B(x)^2 = B_{n-1}x^{2n-2} + B_{n-2}x^{2n-4} + \dots + B_1x^2 + B_0. \quad (3.2)$$

$A(x)B(x)^2 \bmod P(x)$ can be induced from equations (1) and (2) as shown in equation(3).

$$\{\dots [A(x)B_{n-1}x^2 \bmod P(x) + A(x)B_{n-2}]x^2 \bmod P(x) + \dots + A(x)B_1\}x^2 \bmod P(x) + A(x)B_0 \quad (3.3)$$

A definite algorithm for implementing Equation (3) in the above is as follows:

[Algorithm 1] AB^2 Multiplication Algorithm using general irreducible polynomials

Input : $A(x), B(x), P(x)$

Output : $A(x)B(x)^2 \bmod P(x)$

Step 1 : $M(x) = 0$

Step 2 : for $i = n - 1$ to 0

Step 3 : $M(x) = M(x) \cdot x^2 \bmod P(x) + A(x)B_i$

The $M(x) \cdot x^2 \bmod P(x)$ operation and $A(x)B_i (0 \leq i \leq n - 1)$ operation can be performed simultaneously in Step 3 of Algorithm 1, where the basic computations for implementing the above are as follows:

$C1$: 2-bit left shift: $M(x) \cdot x^2$

$C2$: Modular reduction: $M(x) \cdot x^2 \bmod P(x)$

$C3$: $A(x)B_i (0 \leq i \leq n - 1)$

First, in order to perform $C1$, which requires a two-bit left-shift to implement x^2 , a cellular automata with an initial value of 0 and n registers is used. The next state of each register is defined as the state of the second right neighbor in cellular automata with n registers. Here, the leftmost register and rightmost register of the cellular automata are adjacent.

In order to perform $C2$, which is the modular reduction, two modular reduction operations are required due to the two-bit left-shift resulting from $C1$. Since the resultant value obtained from the cellular automata has been shifted to the left by two bits as a result of $C1$, two modular reductions are implemented. The following equations yield $C2$. Let $M(x) \cdot x \bmod P(x)$ be $M'_{n-1}x^{n-1} + M'_{n-1}x^{n-2} + \dots + M'_kx^k + \dots + M'_2x^2 +$

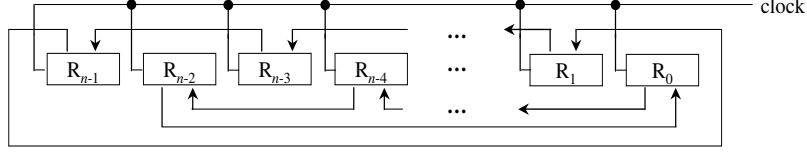


Fig. 3.1. Periodic boundary cellular automata structure reflecting $C1$.

$M'_1x^1 + M'_0$, where $P(x) = x^n + x^k + 1$. Then following equation (4) holds.

$$\begin{aligned} & ((M_{n-1} \wedge P_{n-1}) \oplus M_{n-2})x^{n-1} + ((M_{n-1} \wedge P_{n-2}) \oplus M_{n-3})x^{n-2} \\ & + \dots + ((M_{n-1} \wedge P_k) \oplus M_{k-1})x^k + \dots + ((M_{n-1} \wedge P_2) \oplus M_1)x^2 \\ & + ((M_{n-1} \wedge P_1) \oplus M_0)x^1 + ((M_{n-1} \wedge P_0) \oplus 0) \end{aligned} \quad (3.4)$$

In equation (4), $P_i(0 \leq i \leq n-1)$ has zero values but P_k and P_0 have always '1' since we only consider trinomial as irreducible polynomial introduced in section 2.2. The equation is rewritten as follows.

$$\begin{aligned} & M_{n-2} \cdot x^{n-1} + M_{n-3} \cdot x^{n-2} + \dots + (M_{n-1} \oplus M_{k-1})x^k \\ & + \dots + M_1 \cdot x^2 + M_0 \cdot x^1 + M_{n-1} \end{aligned} \quad (3.5)$$

Equation (5) needs one more computation shown in equation (4) and (5) for shift operation and modular reduction. The resultant equation by operating $C1$ and $C2$ is shown as follows.

$$\begin{aligned} & M(x) \cdot x^2 \bmod P(x) = M_{n-3}x^{n-1} + M_{n-4}x^{n-2} \\ & + \dots + (M_{n-1} \oplus M_{k-1})x^{k+1} + (M_{n-2} \oplus M_{k-2})x^k \\ & + \dots + M_0 \cdot x^2 + M_{n-1} \cdot x^1 + M_{n-2} \end{aligned} \quad (3.6)$$

Fig. 2 shows periodic boundary cellular automata structure considering modular reduction.

$C3$ can be easily obtained using n AND gates since each element of $A(x)$ should be multiplied by the element B_{n-i-1} in the $i(0 \leq i \leq n-1)$ th clock in order to perform $C3$. A method is presented for obtaining AB^2 based on Algorithm 1 using described $C1$, $C2$, and $C3$. The proposed AB^2 multiplication architecture is shown in Fig.3 as $C1$ and $C2$ are performed simultaneously. The proposed algorithm based on irreducible trinomials, $T(x)$, is shown as follows.

To perform our scheme based on Algorithm 2, cellular automata shown in Fig.2 is initialized as zero values.

The proposed AB^2 can be divided into two parts. The upper part performs $A(x)B_i(0 \leq i \leq n-1)$ and the lower part executes 2bit-circularly-left-shift and modular reductions. It is possible to perform AB^2 multiplication in n clock cycles using n AND gates, $(n+2)$ XOR gates and only n -bit register.

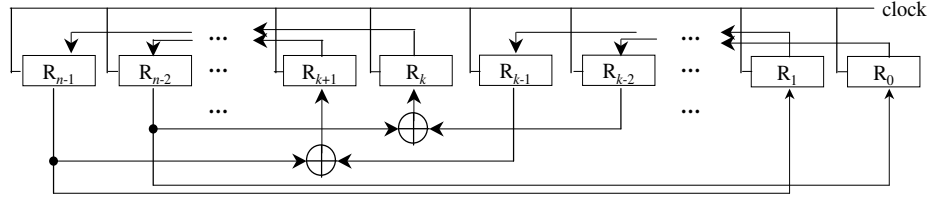


Fig. 3.2. Periodic boundary cellular automata structure reflecting C2.

[Algorithm 2] Proposed AB^2 Multiplication Algorithm using trinomial for ECC

Input : $A(x), B(x), T(x)$

Output : $A(x)B(x)^2 \bmod T(x)$

Step 1 : $M(x) = 0$

Step 2 : *for* $i = n - 1$ *to* 0

Step 3 :
$$M(x) = M_{n-3}x^{n-1} + M_{n-4}x^{n-2} + \dots + (M_{n-1} \oplus M_{k-1})x^{k+1} + (M_{n-2} \oplus M_{k-2})x^k + \dots + M_0x^2 + M_{n-1}x^1 + M_{n-2} + A(x)B_i$$

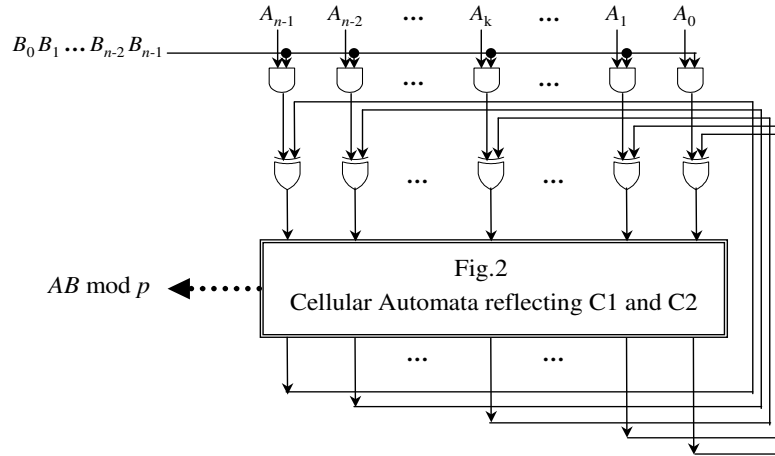


Fig. 3.3. Architecture of AB^2 multiplication using irreducible trinomials.

3.2 Proposed inversion/division architecture

Finite field division operation in $GF(2^n)$ can be performed using a multiplication and a inversion, that is, $A/B = A \cdot B^{-1}$, where the A and B are the elements of $GF(2^n)$. Here, the multiplicative inverse of the field elements B can be obtained by recursive squarings and multiplications, since the field element B can be expressed as

$$B^{-1} = B^{2^n-2} = (B(B(B \dots B(B(B)^2) \dots)^2)^2)^2 \quad (3.7)$$

Division also can be easily induced by equation (7).

$$D = A \cdot B^{-1} = A(B(B(B \dots B(B(B)^2) \dots)^2)^2)^2 \quad (3.8)$$

Here, AB^2 operations can be used as an efficient method. The equation can be computed as [9]:

[Algorithm 3] A/B Division Algorithm

Input : $A(x), B(x), P(x)$

Output : $D(x) = A(x)/B(x)$

Step 1 : $D(x) = B(x)$

Step 2 : for $i = n - 2$ to 1

Step 3 : $D(x) = B(x) \cdot D(x)^2$

Step 4 : $D(x) = A(x) \cdot D(x)^2$

The results is $D = A \cdot B^{-1}$ and when $A = 1$, the algorithm realizes the inverse operation B^{-1} . In this case, AB^2 operation can be used to compute the operations in step3 and 4. Each initial value is as follows: Cellular automata: all zeros, B register : $B(x) = B_{n-1} \dots B_2 B_1 B_0$, Shift register : $B(x) = B_{n-1} \dots B_2 B_1 B_0$

After AB^2 computation by Fig.3, the computed values transfer to Shift register. B^{-1} is computed after mentioned process $n-2$ times. After the whole previous process, system chooses $A(x)$ instead of $B(x)$ in upper registers by *Muxes* for final resultant values. It is possible to perform A/B division in $n(n-1)$ clock cycles using n *AND* gates, $(n+2)$ *XOR* gates, n *Muxes* and three n -bit registers.

4 DISCUSSION AND ANALYSIS

As usually, parallel fashion architectures need much more hardware equipments than serial fashion architectures, and latency is reverse. However the proposed architecture has better complexities than serial or parallel fashion architectures on the fields of the both sides, area and time. Our I/O format, the multiplicand input parallel while the

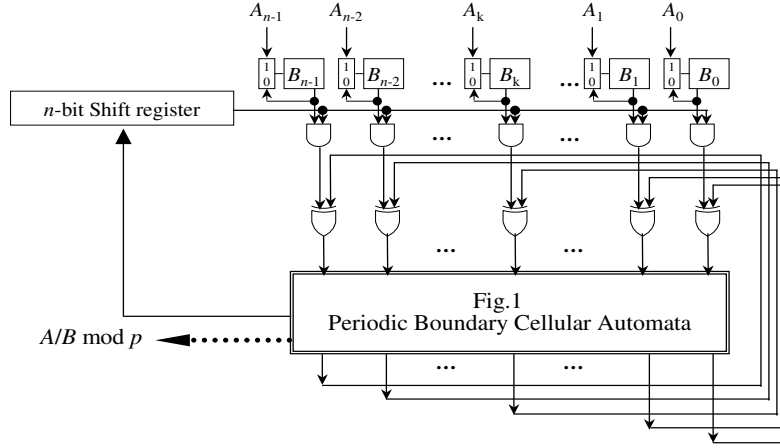


Fig. 3.4. division architecture using irreducible trinomial.

multiplier input serial fashion, differs from typical ways. Thus there are advantages compared to typical architectures.

1) Faster implementation: Bit-serial architectures, for small silicon area, usually takes much more time to perform the operations, and it cannot reduce both time and area complexity though irreducible trinomials are used because of the computation nature.

2) Smaller silicon area: Bit-parallel architectures, such as systolic architectures, usually demand wide silicon area though they are faster. Our architecture is not only much smaller but also faster as much as parallel fashion architectures.

Another advantage is that our architectures can be expanded for other public cryptosystems using general irreducible polynomials, but existing systolic architectures including Wang's and Wei's can be hardly reduced their complexities though they use the restricted irreducible polynomials for ECC, because the binary value of irreducible polynomial in systolic array should be computed with other inputted values whenever it passes through every register. Thus though some of the binary values in irreducible polynomial have zero values, the architecture should input zeros.

5 CONCLUSION

This paper has presented a division architecture based on a modular AB^2 multiplier using irreducible trinomials, which are restricted in ECC. Our scheme has been designed by characteristics of irreducible trinomials and periodic boundary CA. The proposed architecture has been minimized the both time complexity and area complexity such that it has only time complexity of $n(n-1)(T_{AND}+T_{XOR})$ and area complexity of $n(AND+XOR+MUX+3REG.)+2XOR$. Therefore, They have shown outstanding advantages in both area and time compared to typical structures. Our architecture

has a regularity and modularity. Accordingly, it can be used as a efficient basic arithmetic architecture in ECC.

References

1. N.Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.
2. V.Miller. Use of Elliptic Curves in Cryptography. *Advances in Cryptology-CRYPTO'85*, Springer-Verlog Lecture Notes in Computer Science, 218:417–448, Berlin 1986.
3. A.J.Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.
4. T. R. N. Rao and E. Fujiwara. *Error-Control Coding for Computer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
5. W. Drescher, K. Bachmann, and G. Fettweis. VLSI Architecture for Non Sequential Inversion over $GF(2^m)$ using the Euclidean Algorithm. *The International Conference on Signal Processing Applications and Technology*, 2:1815–1819, 1997.
6. J. Von Neumann. *The theory of self-reproducing automata*. University of Illinois Press, Urbana and London, 1966.
7. C. N. Zhang, M. Y. Deng and R. Mason. A VLSI Programmable Cellular Automata Array for Multiplication in $GF(2^n)$. *PDPTA '99 International Conference*, 1118–1123, 1999.
8. P. Pal, Choudhury and R. Barua. Cellular Automata Based VLSI Architecture for Computing Multiplication And Inverses in $GF(2^m)$. *IEEE 7th International Conference on VLSI Design*, 279–282, 1994.
9. A. J. Menezs. *Applications of Finite Fields*. Kluwer Academic Publishers, Boston, MA, 1993.
10. C. Kaufman, R. Perlman and M. Speciner. *Network Security private communication in a public world*. Prentice Hall, New Jersey, 2002.
11. *SEC 1: Elliptic Curve Cryptography version 1.0*. Certicom Reserch, 2000.
12. S. W. Wei. VLSI architecture of divider for finite field $GF(2^m)$. *IEEE International Symposium on Circuit and Systems*, 2:482–485, 1998.
13. C. L. Wang and J. H. Guo. New Systolic Arrays for $C + AB^2$, inversion, and division in $GF(2^m)$. *IEEE Transaction on Computer*, 49:1120–1125, 2000.

² *Corresponding author*. Department of Computer Science, and Research Institute of Computer and Information Communication, Gyeongsang National University.

Efficient On-the-fly Detection of First Races in Nested Parallel Programs^{*}

Keum-Sook Ha,¹ Yong-Kee Jun,² and Kee-Young Yoo³

¹ Kumi College, Kumi; ksha@kumi.ac.kr

² Gyeongsang National Univ., Jinju; jun@gsnu.ac.kr

³ Kyungpook National Univ., Daegu; yook@knu.ac.kr
South Korea

Abstract. To debug explicitly shared-memory programs with nested parallelism effectively, it is important to detect efficiently the first data races to occur in such programs because they incur non-deterministic executions and then may make other affected races hidden or appeared. The previous on-the-fly techniques used to detect the first races in such kind of programs are inefficient, as the number of the accesses stored for each shared variable during the execution depends on the maximum parallelism of the program. This paper presents a novel on-the-fly technique to detect the first races efficiently, in which a small constant is the number of accesses stored for each shared variable during the execution.

1 Introduction

The *data race* [10] is an access error which can arise in an execution of explicitly shared-memory program [1, 11] in which the parallel threads use shared variables and include at least one write access without appropriate synchronization. Since such the races result in unintended non-deterministic executions of programs, it is important to detect the races for the effective debugging of such programs, especially to detect efficiently the first races [2, 4, 10] to occur as they are unaffected by other races and may lead other races to appeared or hidden.

Among the previous on-the-fly detection techniques [2, 5, 12, 14] to detect the first races, the most practical one [12] for programs with nested parallelism employs a two-pass monitoring algorithm. The first pass examines the logical concurrency between each current access to a shared variable and the previous conflicting accesses maintained in a shared data structure for the shared variable in order to collect a subset of candidate accesses [4–6, 12] for being involved in a first race. The second pass also examines the logical concurrency between each current access and the conflicting candidate accesses already collected in the first pass in order to supplement the set of candidate accesses if they are concurrent with each other, and then reports the set as a representation of the detected first races in the end of the pass. This technique is however inefficient with regard to the execution time and memory space, as the number of accesses stored for each shared variable to discriminate the set of candidate accesses depends on the maximum parallelism of the program.

^{*} This work is supported in part by Ministry of Information and Communication, South Korea.

This paper presents a novel two-pass monitoring algorithm for detecting the first races to occur in programs with nested parallelism. In its first pass, the number of accesses stored is kept constant based on examining the *happened-before* relation [7] and the *left-of* relation [8] between every two accesses to the shared variable in order to collect a constant number of candidate accesses for being involved in first races. In the second pass, the candidates collected in the first pass are examined also based on the happened-before relation and the left-of relation with each current conflicting access in order to complete the set of candidate accesses. This technique is still more efficient with regard to the execution time and memory space, because a small constant is the number of accesses stored for each shared variable during the execution.

After describing the background to this work in the next section, we present our technique in Section 3 which include subsections to introduce the constant number of candidate accesses collected in the first pass and the two-pass monitoring algorithm. Section 4 analyzes the efficiency of the technique in terms of its time and space complexities, and compares it with the previous technique. The final section derives some conclusions and future work.

2 Background

We consider OpenMP programs [1, 11] as a representative example of explicitly shared-memory programs with nested parallelism. In an execution of OpenMP program, more than one thread can be forked to share the work based on a ‘`PARALLEL DO`’ directive, and joined based on the corresponding ‘`END PARALLEL DO`’ directive. The *nesting level* of such parallel construct is equal to one plus the number of enclosing outer construct. Figure 2.1 shows an OpenMP Fortran program where three parallel loops are specified with corresponding pairs of directives. The variable `X` is declared as a shared variable by specifying ‘`SHARED (X)`’ and the three variables `{I, J1, J2}` are declared as private variables of each thread by specifying ‘`PRIVATE (I, J1, J2)`’.

The concurrency relation among threads in an execution can be represented by a directed acyclic graph called *Partial Order Execution Graph* (POEG) [3] as shown in Figure 2.2. In a POEG, a vertex indicates a fork or join operation, and an arc starting from a vertex represents a forked thread. For indicating a read or write access executed in a thread, we usually put a small filled circle on the thread which is accompanied with a letter *r* or *w* respectively and a number indicating the order in which those accesses are observed.

Concurrency determination is not dependent on the number or relative speeds of processors executing the program. Because the graph captures the *happened-before* relationship [7], it represents a partial order over the set of events executed by the program and may be denoted as $Ordered(e_i, e_j)$. An event e_i *happened before* another event e_j if there exists a path from e_i to e_j in the POEG; and e_i is *concurrent* with e_j if neither one happened before the other. For example, consider the accesses in Figure 2.2, where $r0$ happened before $w7$ because there exists a path from $r0$ to $w7$, and $r0$ is concurrent with $w11$, because there is no path between them. The *maximum parallelism* of a POEG is the maximum number of mutually-concurrent events in the graph.

```

C$OMP PARALLEL DO SHARED(X)
C$OMP PRIVATE(I, J1, J2)
DO I = 1, 2
    ... = X {r0, r8}
    IF (I.EQ.2) THEN
C$OMP PARALLEL DO
DO J1 = 1, 2
    ... = X {r1, r2}
    ... = X {r3, r4}
C$OMP END PARALLEL DO
END IF
    ... = X {r5, r9}
    IF (I.EQ.2) THEN
C$OMP PARALLEL DO
DO J2 = 1, 2
    IF ... THEN ... = X {r6}
    IF ... THEN X = ... {w10}
    X = ... {w7, w13}
C$OMP END PARALLEL DO
END IF
    X = ... {w11, w14}
    IF (I.EQ.1) THEN ... = X {r12}
C$OMP END PARALLEL DO

```

Fig. 2.1. An OpenMP Program

Two accesses to a shared variable are *conflicting* if at least one of them is a write. If two accesses $\{a_i, a_j\}$ are conflicting and concurrent with each other then the two accesses are involved in a *race* denoted $a_i\text{-}a_j$. An access a_j is *affected* by another access a_i , if a_i happened before a_j and a_i is involved in a race. A race $a_i\text{-}a_j$ is *unaffected*, if neither a_i nor a_j are affected by any other accesses. The race is *partially affected*, if only one of a_i and a_j is affected by another access. A *tangle* is a set of partially affected races such that if $a_i\text{-}a_j$ is a race in the tangle then exactly one of a_i or a_j is affected by a_k such that $a_k\text{-}a_l$ is also in the same tangle. A *tangled race* is a partially affected race that is in a tangle. A *first race* is either an unaffected race or a tangled race.

There are twenty-seven races in the POEG shown in Figure 2.2; all of the accesses in the POEG are involved in races. Among them, only three races $\{r0\text{-}w11, w7\text{-}r8, r8\text{-}w10\}$ are first races which are tangled races. Eliminating the three tangled races may make the other twenty-four affected races disappear. The term tangled race was introduced by Netzer and Miller [9] describing the situation when no single race from a special set of races is unaffected by the others. Note that there can never be exactly one tangled race in an execution. They also introduce a tighter notion of first race called *non-artifact race* which uses the event-control dependences to define how accesses affect each other.

The previous on-the-fly techniques to detect the first races in programs with nested parallelism can be classified into three classes with respect to the number of re-executions required for monitoring the debugged program: Race Frontier [2] with an indefinite number of re-executions, RecPlay [14] with at least three re-executions, and another one [12] with at most two re-executions. The most practical technique [12] employs a two-pass monitoring algorithm. Its first pass examines the logical concurrency between

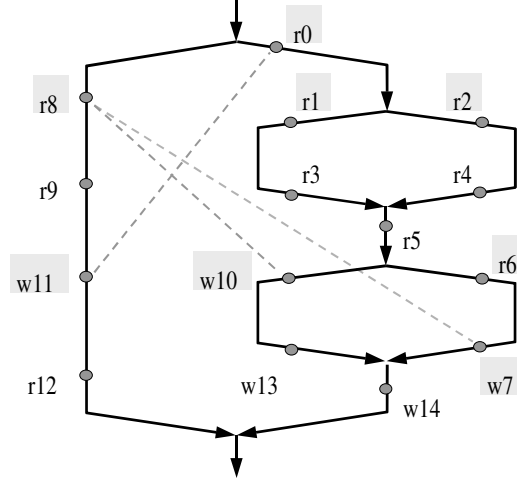


Fig. 2.2. A Partial Order Execution Graph (POEG)

each current access to a shared variable and the previous conflicting accesses maintained in a shared data structure called *access history* for the shared variable in order to collect a subset of candidate accesses [4–6, 12] for being involved in a first race. Its second pass also examines the logical concurrency between each current access and the conflicting candidate accesses already collected in the first pass in order to supplement the set of candidate accesses if they are concurrent with each other, and then reports the set as a representation of the detected first races in the end of the pass. This technique is however inefficient with regard to the execution time and memory space, as the number of accesses stored for each shared variable to discriminate the set of candidate accesses depends on the maximum parallelism of the program.

3 The Monitoring Algorithms

We present a novel two-pass monitoring algorithm for detecting the first races to occur in programs with nested parallelism based on the notion of *frontier candidates* which are a small subset of candidate accesses for being involved in first races and determined using *happened-before* [7] and *left-of* [8] relations of any pair of two accesses. The first pass collects the frontier candidates; the second pass completes the set of candidate accesses for being involved in first races including the frontier candidates collected in the first pass.

3.1 The Frontier Candidates

Consider that a fork event e_f happened before two mutually-concurrent accesses $\{e_i, e_j\}$ which are executed in two different threads preceded by their ancestor threads

forked by e_f with the index i and j ($i < j$) respectively. In this case, we say e_i is *left of* e_j and denote the relation as $Leftof(e_i, e_j)$. Given two accesses $\{e_i, e_j\}$ of the same access type, if there does not exist any pair of two accesses $\{e_h, e_k\}$ such that $Leftof(e_h, e_i) \wedge Leftof(e_j, e_k)$ is satisfied, then we say that $\{e_i, e_j\}$ are executed *outside*. Otherwise, we say that e_i is more *inside* than e_h , or e_j is more *inside* than e_k . For instance, the two accesses $\{r2, r8\}$ in Figure 2.2 are outside, because there does not exist any two accesses $\{r_x, r_y\}$ which satisfies $Leftof(r_x, r8) \wedge Leftof(r2, r_y)$. On the other hand, an access $r1$ is more inside than the two accesses, because it satisfies $Leftof(r8, r1) \wedge Leftof(r1, r2)$.

In this work, an access history AH_X for a shared variable X is composed of two subsets of most recent accesses: $AH_X[R]$ and $AH_X[W]$. $AH_X[R]$ stores two read accesses which were executed outside: $AH_X[R_L]$ for the leftmost read, and $AH_X[R_R]$ for the rightmost read. $AH_X[W]$ stores only one write access. Thus, the total number of entries in an access history is always three.

We say that an access is a *frontier access*, if it is executed outside at that time and involved in a race. For example, $r8$ shown in Figure 2.2 is a frontier access, because it is outside at that time and involved in a race $w7$ - $r8$. A frontier access can be a candidate access for being involved in first races. A read or write access is a *read* (or *write*) *candidate*, if there does not exist any accesses that happened before the access. A write access is a *read-write candidate* or *r-write candidate*, if there does not exist any other write access but a read access that happened before the access. A read or write candidate is *effective* by itself, yet an r-write candidate is only effective when there is no write candidate in the execution. If there exists an access involved in a first race, it must be an effective candidate access. Figure 2.2 shows two read candidates $\{r0, r8\}$, no write candidate, and three r-write candidates $\{w7, w10, w11\}$.

A *candidate history* CH_X for a shared variable X is composed of two subsets of the effective frontier candidates: $CH_X[R]$ and $CH_X[W]$. The number of frontier candidates is therefore at most four, which is composed of two frontiers for each one of $CH_X[R]$ and $CH_X[W]$. $CH_X[R]$ stores two read frontier candidates: $CH_X[R_L]$ for the leftmost and $CH_X[R_R]$ at the rightmost. Like $CH_X[R]$, $CH_X[W]$ also stores two write frontier candidates: $CH_X[W_L]$ for the leftmost and $CH_X[W_R]$ for the rightmost. A *candidate set* CS_X of a shared variables X is the set of all the effective candidates in the corresponding execution instance of monitored program. CS_X is composed of two subsets of effective candidates: $CS_X[R]$ and $CS_X[W]$. Note that these two notions, the candidate history and the candidate set, do not discriminate the r-write candidate from the write candidate, because the two candidate types of write accesses are not effective together in an execution instance.

3.2 The First-Pass Protocol

The first monitoring pass keeps a constant number of accesses to be stored in an access history based on examining the happened-before relation and the left-of relation between every pair of two accesses to the shared variable in order to collect a constant number of frontier candidates for being involved in first races. Figure 3.3 shows two protocol procedures for the two types of accesses in the first monitoring pass: **check-**

<pre> 0 checkread_1st($X, current$) 1 if \neg Ordered($AH_X[W], current$) then 2 $racing_current := true$; 3 if \neg Leftof($AH_X[R_L], current$) then 4 $AH_X[R_L] := current$; 5 if \neg Leftof($current, AH_X[R_R]$) then 6 $AH_X[R_R] := current$; 7 if \neg $racing_current$ then 8 return; 9 if $CH_X[R_L] = \emptyset$ or 10 Leftof($current, CH_X[R_L]$) then 11 $CH_X[R_L] := current$; 12 if $CH_X[R_R] = \emptyset$ or 13 Leftof($CH_X[R_R], current$) then 14 $CH_X[R_R] := current$; 15 end checkread_1st </pre>	<pre> 0 checkwrite_1st($X, current$) 1 for all c in AH_X do 2 if \neg Ordered($c, current$) then 3 $racing_current := true$; 4 $AH_X[W] := current$; 5 if \neg $racing_current$ then 6 return; 7 if $CH_X[W_L] = \emptyset$ or 8 Leftof($current, CH_X[W_L]$) then 9 $CH_X[W_L] := current$; 10 if $CH_X[W_R] = \emptyset$ or 11 Leftof($CH_X[W_R], current$) then 12 $CH_X[W_R] := current$; 13 add $current$ to $CS_X[Backup]$; 14 halt; 15 end checkwrite_1st </pre>
--	---

Fig. 3.3. The First-Pass Protocol

read_1st() and **checkwrite_1st**(), where *current* represents the current access to a shared variable X .

In **checkread_1st**(), line 1-2 determines if the *current* is involved in a race by checking the happened-before relation with one write access stored previously in $AH_X[W]$. Line 3-6 checks the left-of relation between the *current* and every read stored in $AH_X[R]$ in order to maintain $AH_X[R]$ to store the only outside accesses. If the *current* was not involved in a race by line 1, this procedure returns to exit in line 7-8, as the *current* is not a frontier. Otherwise, it checks in line 9-14 the left-of relations between the *current* and the previous frontier candidates stored in $CH_X[R]$ to determine if it becomes a new frontier candidate to replace the previous access there.

In **checkwrite_1st**(), line 1-3 determines if the *current* is involved in a race by checking the happened-before relations with three previous accesses stored in AH_X . In line 4, the *current* is stored into $AH_X[W]$ unconditionally, because a future access that is involved in a race with the access stored previously in $AH_X[W]$ must be also involved in another race with the *current*. If the *current* was not involved in a race by line 1-3, this procedure returns to exit in line 5-6 as the *current* is not guaranteed to be involved into a first race. Otherwise, it checks in line 7-12 the left-of relations between the *current* and the previous candidates stored in $CH_X[W]$ to determine if it becomes a new frontier candidate to replace the previous access there. And, the racing *current* is stored into a part of the candidate set $CS_X[Backup]$ to be reported optionally, because it becomes an effective r-write candidate if $CS_X[W]$ becomes empty in the second pass in which the thread is halted for efficiency immediately after any effective candidate is determined. Finally, this current thread is also halted to eliminate unnecessary monitoring time thereafter, because all accesses which are preceded by a racing write must not be a candidate.

Table 1. The Final States of Our Technique for Figure 2.2

	The First Pass	The Second Pass
$AH_X[R]$	$r6, r9$	–
$AH_X[W]$	$w11$	–
$CH_X[R]$	$r8, r8$	$r0, r8$
$CH_X[W]$	$w10, w11$	$w10, w11$
$CS_X[R]$	–	$r0, r8$
$CS_X[W]$	–	–
$CS_X[Backup]$	$w10, w11$	–

Table 1 shows an example of the final state which is resulted by applying these two procedures of the first-pass monitoring protocol to the execution instance shown in Figure 2.2.

3.3 The Second-Pass Protocol

In the second pass, the frontier candidates collected in the first pass are examined also based on the happened-before relation and the left-of relation with each current conflicting access in order to complete the set of candidate accesses. Figure 3.4 shows two protocol procedures for the two types of accesses in the second monitoring pass: **checkread_2nd()** and **checkwrite_2nd()**.

In **checkread_2nd()**, line 1-3 determines if the *current* is involved in a race by checking the happened-before relation with the two frontier candidates of $CH_X[W]$ collected in the first pass. If the *current* was not involved in a race by being checked in line 1-3, this procedure returns to exit in line 4-5 because this case implies that $CH_X[W]$ is empty and then the *current* is not a frontier. Otherwise, it checks in line 6-11 the left-of relations between the *current* and the previous frontier candidates stored in $CH_X[R]$ to determine if it becomes a new frontier candidate to replace the previous there. And, the racing *current* is added into $CS_X[R]$ to be reported, because it is an effective read candidate. Finally, this current thread is halted to eliminate unnecessary monitoring time thereafter, because an effective r-write candidate is the only candidate that can be preceded by a racing read candidate and might have been already reported in $CS_X[Backup]$ in **checkwrite_1st()**.

In **checkwrite_2nd()**, line 1-3 determines if the *current* is involved in a race by checking the happened-before relations with all of the four frontier candidates stored in CH_X . And then, the remaining part of the procedure is similar to that in **checkread_2nd()**.

Table 1 shows an example of the final states which are resulted by applying these procedures of the two-pass monitoring protocols shown in Figure 3.3-3.4 to the execution instance shown in Figure 2.2.

<pre> 0 checkread_2nd($X, current$) 1 for all c in $CH_X[W]$ do 2 if $\neg Ordered(c, current)$ then 3 $racing_current := true$; 4 if $\neg racing_current$ then 5 return; 6 if $CH_X[R_L] = \emptyset$ or 7 $Leftof(current, CH_X[R_L])$ then 8 $CH_X[R_L] := current$; 9 if $CH_X[R_R] = \emptyset$ or 10 $Leftof(CH_X[R_R], current)$ then 11 $CH_X[R_R] := current$; 12 add $current$ to $CS_X[R]$; 13 halt; 14 end checkread_2nd </pre>	<pre> 0 checkwrite_2nd($X, current$) 1 for all c in CH_X do 2 if $\neg Ordered(c, current)$ then 3 $racing_current := true$; 4 if $\neg racing_current$ then 5 return; 6 if $CH_X[W_L] = \emptyset$ or 7 $Leftof(current, CH_X[W_L])$ then 8 $CH_X[W_L] := current$; 9 if $CH_X[W_R] = \emptyset$ or 10 $Leftof(CH_X[W_R], current)$ then 11 $CH_X[W_R] := current$; 12 add $current$ to $CS_X[W]$; 13 halt; 14 end checkwrite_2nd </pre>
--	---

Fig. 3.4. The Second-Pass Protocol

4 Analysis

This two-pass monitoring technique stores *at most* seven entries of access information for each shared variable into the two kinds of *monitoring histories*: three accesses for access history and four accesses for candidate history. This makes the efficiency of this technique depend on the worst-case complexities of two factors: (1) the memory space required for each entry of the monitoring histories, and (2) the time required for comparing an access with another accesses to determine the happened-before relation and the left-of relation. Since these two factors are dependent on the labeling schemes like NR-Labeling [13] which generate constant-sized access information, the complexities of this technique are constant for both the space required for each monitoring history and the number of event comparisons for each access to a shared variable.

Let V be the number of shared variables, N be the nesting depth, and T be the maximum parallelism of the monitored parallel program. The worst-case complexities of the technique includes therefore $O(V)$ space for seven constant-sized entries of monitoring histories except the space to generate access information which is $O(NT)$ by NR-Labeling for all of the simultaneously active threads each of which has $O(N)$; and $O(\log_2 N)$ time by NR-Labeling on every access for comparing with at most seven previous accesses in monitoring histories except the time to generate access information which is $O(N)$ by NR-Labeling at every fork or join operation. On the other hand, the worst-case complexities of the most practical previous technique [12] includes $O(VT)$ space for the monitoring histories with constant-sized entries except the space to generate access information which is $O(NT)$ by NR-Labeling; and $O(T \log_2 N)$ time by NR-Labeling on every access for comparing with the previous accesses in monitoring histories except the time to generate access information which is $O(N)$ by NR-Labeling, as the number of the accesses stored for each shared variable during the execution depends on the maximum parallelism of the program.

Table 2. The Final States of the Previous Technique [12] for Figure 2.2

	The First Pass	The Second Pass
$AH_X[R]$	$r6, r8$	–
$AH_X[W]$	$w7, w10, w11$	–
$CS_X[R]$	$r8$	$r0, r8$
$CS_X[W]$	$w10, w11$	–
$CS_X[RW]$	–	$w7, w10, w11$

For example, Table 1 and 2 show the final states resulted by applying to Figure 2.2 the previous technique [12] and our technique, respectively. Note that Table 2 uses CS_X to share the usages of both CH_X and CS_X in Table 1. After the first pass, Table 1 reports two candidate accesses optionally in $CS_X[Backup]$ using the two monitoring histories which are full in its seven entries without regard to the maximum parallelism of the graph, but Table 2 reports three candidate accesses in CS_X using the two monitoring histories in which the number of write accesses in $AH_X[W]$ is the maximum parallelism of the graph. After the second pass, Table 1 reports additional two candidate accesses in CS_X using the two monitoring histories which are full only in its four entries of candidate history without regard to the maximum parallelism of the graph, but Table 2 reports additional two candidate accesses in CS_X shared for the candidate history in which the number of write accesses is the maximum parallelism of the graph.

5 Conclusions

The previous on-the-fly techniques used to detect the first races in the programs with nested parallelism are inefficient, as the number of the accesses stored in monitoring histories for each shared variable during the execution depends on the maximum parallelism of the program. This paper presents a novel two-pass monitoring algorithm for detecting the first races to occur in such kind of programs. In its first pass, the number of accesses stored in monitoring histories is kept constant based on examining the happened-before relation and the left-of relation between every two accesses to the shared variable in order to collect a constant number of frontier candidates for being involved in first races. In the second pass, the frontier candidates collected in the first pass are examined also based on the happened-before relation and the left-of relation with each current conflicting access in order to complete the set of candidate accesses. This technique is still more efficient with regard to the execution time and memory space, because a small constant is the number of accesses stored in monitoring histories for each shared variable during the execution. We have been extending our technique for a broader class of programs with inter-thread coordination, and developing a visualization scheme that is easy for programmers to debug programs in accordance with the power of this technique.

References

1. Chandra, R., L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon: Parallel Programming in OpenMP, Academic Press (2001)
2. Choi, J., and S. Min: Race Frontier - Reproducing Data Races in Parallel Program Debugging, 3rd ACM symp. on Principles and Practice of Parallel Programming (1991) 145-154
3. Dinning, A., and E. Schonberg: An Empirical Comparison of Monitoring Algorithms for Access Anomaly Detection, 2nd ACM Symp. on Principles and Practice of Parallel Programming (1990) 1-10
4. Jun, Y., and C. E. McDowell: On the fly Detection of the First Races in Programs with Nested Parallelism, 2nd Int'l Conf. on Parallel and Distributed Processing Techniques and Application, CSREA (1996) 1549-1560
5. Kim, J., and Y. Jun: Scalable On the fly Detection of the First Races in Parallel Programs, 12nd ACM Int'l. Conf. on Supercomputing (1998) 345-352
6. Kim, J., D. Kim, and Y. Jun: Scalable Visualization for Debugging Races in OpenMP Programs, 3rd Int'l. Conf. on Communications in Computing, World Academy of Sciences (2002) 259-265
7. Lamport, L.: Time, Clocks, and the Ordering of Events in Distributed System, Communication of the ACM, Vol. 21(7) (1978) 558-565
8. Mellor-Crummey, J.: On-the-fly Detection of Data Races for Programs with Nested Fork-Join Parallelism, ACM/IEEE Supercomputing (1991) 24-33
9. Netzer, R. H. B., and B. P. Miller: Improving the Accuracy of Data Race Detection, 3rd ACM Symp. on Principles and Practice of Parallel Progr. (1991) 133-144
10. Netzer, R. H. B., and B. P. Miller: What are Race Conditions? Some Issues and Formalizations, ACM Letters on Programming Language and Systems, Vol. 1(1) (1992) 74-88
11. OpenMP Architecture Review Board: OpenMP Fortran Application Program Interface, Ver. 2.0 (2000)
12. Park, H. and Y. Jun: Detecting the First Races in Parallel Programs with Ordered Synchronization, 6th Int'l Conf. on Parallel and Distributed Systems, IEEE (1998) 201-208
13. Park, S., M. Park, and Y. Jun: A Comparison of Scalable Labeling Schemes for Detecting Races in OpenMP Programs, Int'l Workshop on OpenMP Applications and Tools (2001) 68-80
14. Ronsse, M., and K. De Bosschere: RecPlay - A Fully Integrated Practical Record/Replay System, ACM Tr. on Computer Systems, Vol. 17(2) (1999) 133-152

New Architecture for Inversion and Division over $GF(2^m)$

Kyeoung Ju Ha¹, Kyo Min Ku², and Kee Young Yoo³

¹ Daegu Hanny University, 290 Yugok-dong, Gyeongsan-si, Gyeongsangbuk-do, Korea
712-715

kjha@dhu.ac.kr

² Mobilab Co. Ltd, 952-3 Dongchun-dong, Buk-gu, Daegu, Korea 702-250
kmku@mobilab.co.kr

³ Kyungpook National University, 1370 Sankyuk-dong, Buk-gu, Daegu, Korea 702-701
yook@knu.ac.kr

Abstract. This paper presents a new architecture for inversion and division based on a Cellular Automata over $GF(2^m)$. The Proposed architecture uses the characteristics of AB^2 operation in finite field. The architecture proposed in this paper is more efficiently in terms of the space and time.

1 Introduction

In this paper, we propose the new architecture for inversion and division over $GF(2^m)$. In finite field, the division can be computed using the multiplication and the inversion of the multiplication since $A(x)/B(x)$ is equal to $A(x)B(x)^{-1}$. For an element over finite fields, the methods that compute the inversion for multiplication are being way using logical function [1], Euclid's algorithm [2] or a lot of multiplication's operation [3]. In this paper we use the AB^2 architecture[4] to design the inversion and division architecture.

2 Inversion and Division

2.1 Inversion

The inversion of the multiplication is a special case of the exponentiation since $B(x)^{-1}$ is equal to $B(x)^{2^m-2}$ in finite field. The $B(x)^{2^m-2}$ can be represented as $B(x)^{2^m-2} = (B(x)^2)(B(x)^{2^2}) \dots (B(x)^{2^{m-1}})$.

Because the inversion of the multiplication is the case that the exponentiation is equal to $(2m-2)$, where $(2m-2)$ is represented by m -tuple vector $[1 \ 1 \ 1 \ \dots \ 0]$ ($e_0=0$, $e_i=1 (1 \leq i \leq m-1)$). So, $B(x)^{2^m-2}$ can be represented as follows:

$$\begin{aligned} B(x)^{2^m-2} &= (\dots((B(x)^{e_{m-1}})^2 B(x)^{e_{m-2}})^2 \dots B(x)^{e_1})^2 B(x)^{e_0} \\ &= B(x)^{e_0} (B(x)^{e_1} \dots (B(x)^{e_{m-2}} (B(x)^{e_{m-1}})^2)^2 \dots)^2 \\ &= B(x)^0 (B(x) \dots (B(x) (B(x))^2 \dots)^2)^2 \end{aligned}$$

$$= (B(x) \dots (B(x) (B(x))^2 \dots)^2 \dots)^2 \quad (1)$$

In the equation (1), the regularity is discovered as follows:

$$\begin{aligned} C^{(0)}(x) &= B(x) \\ C^{(1)}(x) &= B(x)C^{(0)}(x)^2 \\ C^{(2)}(x) &= B(x)C^{(1)}(x)^2 \\ C^{(3)}(x) &= B(x)C^{(2)}(x)^2 \\ &\dots \\ C^{(m-2)}(x) &= B(x)C^{(m-3)}(x)^2 \\ C^{(m-1)}(x) &= B(x)C^{(m-2)}(x)^2 = C(x) \end{aligned}$$

Accordingly, $C^{(i)}(x)$ is represented by $C^{(i)}(x) = B(x)C^{(i-1)}(x)^2$ ($1 \leq i \leq m-2$, $C^{(0)}(x) = B(x)$) and $C^{(i)}(x) = C^{(i-1)}(x)^2$ ($i = m-1$). Therefore, the equation (1) can be represented by as follows:

$$\begin{aligned} C^{(i)}(x) &= B(x)C^{(i-1)}(x)^2, \text{ for } 1 \leq i \leq m-1 \text{ (if } i = m-1 \text{ then } B(x)=1) \quad (2) \\ \text{, where } C^{(0)}(x) &= B(x) \text{ and } C^{(m-1)}(x) = B(x)^{-1} \text{ if } i = m-1. \end{aligned}$$

As in the equation (2), the AB^2 architecture is used in each step. The algorithm 1 which computes the inversion for multiplication using AB^2 architecture is as follows:

Algorithm 1 : Inversion($B(x)$, $P(x)$)
 Input : $B(x)$, $P(x)$
 Output : $C(x) = B(x)^{-1} \bmod P(x)$
 1 : $C^{(0)}(x) = B(x)$
 2 : for $i = 1$ to $m-2$
 3 : $C^{(i)}(x) = B(x)C^{(i-1)}(x)^2 \bmod P(x)$
 4 : end for
 5 : $C(x) = C^{(m-2)}(x)^2 \bmod P(x)$

The algorithm 1 can be implemented such as in Fig. 3.1 using the AB^2 architecture which is proposed in [4]. In Fig. 3.1, an AB^2 architecture is used for operation $B(x)^{-1} \bmod P(x)$ over finite fields $\text{GF}(2^m)$.

2.2 Division

In the 5^{th} phase of algorithm 1, If $C(x) = C(x)^2 \bmod P(x)$ is converted by $C(x) = A(x)C(x)^2 \bmod P(x)$, then we can obtain the result of A/B operation. As mentioned above, because $A/B = AB^{-1}$. The Fig. 3.2 shows the A/B architecture using this property.

3 Results

The MUX is unnecessary and the number of latch is $(m^2 - m - 3)$ in the proposed inversion for multiplication and division.

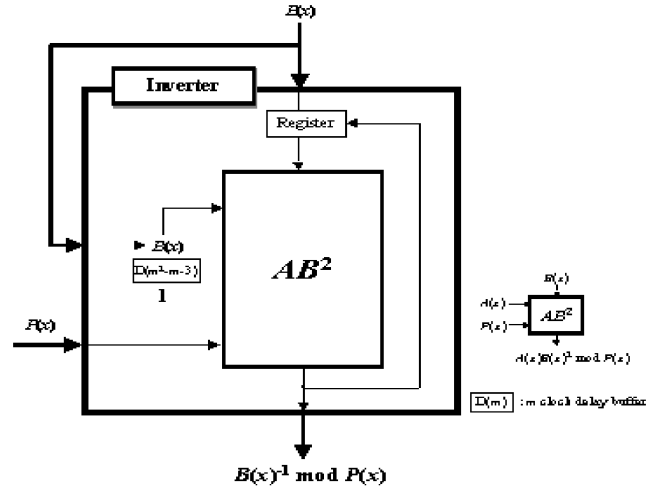


Fig. 2.1. A structure for performing an $B(x)^{-1} \bmod P(x)$ operation.

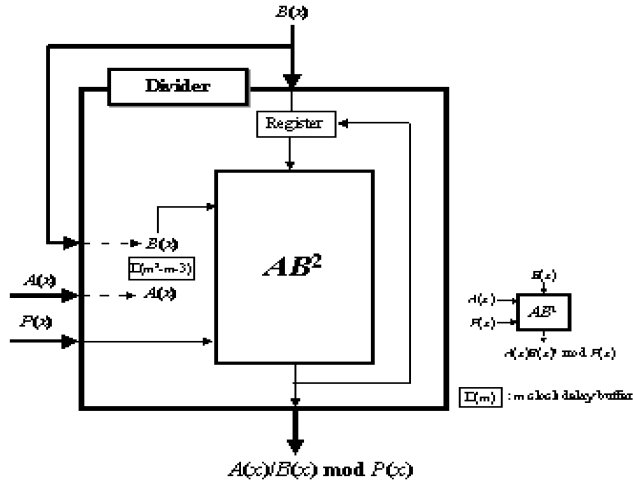


Fig. 2.2. Structure for performing an $A(x)/B(x) \bmod P(x)$ operation.

The proposed architecture is much more efficient in terms of the space and time than that of the systolic array and can be utilized more efficiently in crypto systems.

References

1. G.I.Davida. Inverse of Elements of a Galois Field. *IEE Electronics Letters*, 19th Oct. Vol. 8 1972
2. W.Diffie, M.E.Hellman. New Directions in Cryptography. *IEEE Trans. On Info. Theory*, Vol.22 1976
3. C.C.Wang, T.K.Trung, H.M.Shao, L.J.Deutsch, J.K.Omura, I.S.Reed. *VLSI for Computing Multiplications and Inversses in $GF(2^m)$* . IEEE Trans. On Comp., Vol. C-34 1985
4. K.M.Ku, K.J.Ha, K.Y.Yoo. Design of New AB^2 Multiplier over $GF(2^m)$ using Cellular Automata. *IEE Proceedings-Circuits, Devices and Systems* Vol. 151, No 2 2004

Solving Linear Systems on Cluster Computers with High Accuracy

Carlos Amaral Hölbíg¹,
Paulo Sérgio Morandi Jr.², Bernardo Frederes Krämer Alcalde²,
Tiarajú Asmuz Diverio², and Dalcídio Moraes Claudio³

¹ Universidade de Passo Fundo and PPGC at UFRGS - Passo Fundo - Brazil
holbig@upf.br

² Instituto de Informática and PPGC at UFRGS - Porto Alegre - Brazil
{diverio, sergio, bfcalkalde}@inf.ufrgs.br

³ Faculdade de Informática and PPGCC at PUCRS - Porto Alegre - Brazil
dalcidio@pucrs.br

Abstract. Many different numerical algorithms contain the solution of linear equation systems as a subproblem. Because of these aspects, this work aims the development of solvers to linear systems (for dense and sparse matrices) with high accuracy on cluster computers using C-XSC library (a C++ class library for extended scientific computing). We want to combine the high accuracy given by C-XSC with the computational gain provided by parallelization.

1 Introduction

One of the most frequent tasks in numerical analysis is the solution of linear systems of equations

$$Ax = b \tag{1.1}$$

with an $m \times n$ matrix A and a right hand side $b \in \mathbb{R}^n$. Many different numerical algorithms contain this task as a subproblem. Because of these aspects, this work aims the development of solvers with high accuracy for linear systems of equations and the adaptation of the algorithms implemented to cluster computers using C-XSC library (see details about this library in [2] and [3]). This library is available for download in www.math.uni-wuppertal.de/wrswt/index_en.html. Our solvers work with dense and sparse (in special banded matrices) linear systems of equations. Nowadays, the solver for dense matrices works with all four basic numerical C-XSC data types: *real*, *interval*, *complex*, and *complex interval* and the solver for sparse matrices works with *real* and *interval* data types. All our programs are freeware (C++ templates and the C++ exception handling are not used in the actual implementations, these characteristics will be used in future versions of our solvers).

2 The Algorithms

The algorithms implemented in our work were described in [4] and can be applied to any system of linear equations which can be stored in the floating point system on

the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular. In the implementation in C-XSC, there is a chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C-XSC offers for IEEE floating point arithmetic [5].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. We present some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms presented here can be found in [6].

3 Solvers for Dense and Sparse Linear Systems

The C-XSC programs implemented in our solver for dense linear systems were written for the case of *real* input data (i.e. A is of type *rmatrix* and b is of type *rvector*) and for the case of the data types *interval*, *complex*, and *complex interval*. The changes made for the use of these other types are mainly changes of the data type of certain variables and functions in the program. Our C-XSC program *verifies the existence* of a solution and *computes an enclosure* for each of the following types of problems:

- (s) compute an enclosure for the solution of system (1.1) for a *square* $n \times n$ matrix A .
- (o) compute an enclosure for the solution of system (1.1) in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.
- (u) compute an enclosure for the solution of system (1.1) in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.
- (S) compute an enclosure of the *inverse* A^{-1} of A .
- (O) compute an enclosure of the *pseudo inverse* A^+ of A in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.
- (U) compute an enclosure of the *pseudo inverse* A^+ of A in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.

This solver has two modules: the module `lss_aprx` contains the function `MINV` which computes an approximate inverse of the input matrix A of type *rmatrix* using the Gauss-Jordan algorithm (see i.e. [7]), when A is a square matrix. In the over- or under-determined case we use the Moore-Penrose pseudo inverse A^+ of A (if A has full rank). The second module `lss` contains the functions which solve the dense linear system. This system may be square and non square ($m \times n$). In the over-determined case

($m > n$) a vector $x \in \mathbb{R}^n$ is sought whose residuum $b - Ax$ has minimal Euclidian norm whereas in the under-determined case ($n < m$) a solution $x \in \mathbb{R}^n$ is sought which has minimal norm. Example solved with this solver is showed in Sect. 5.

For the solution of a sparse linear system we present an implementation of an algorithm to compute efficiently componentwise good enclosures. Our implementation works with point as well as *interval* data (data afflicted with tolerances). We assume linear systems whose coefficient matrix has a banded structure. In this case the well known general algorithm (using the Krawczyk operator) to solve systems with dense matrices is not efficient. Since the approximate inverse R of a banded matrix A is in general a full matrix, a lot of additional storage would be required, especially if the bandwidth of A is small compared with its dimension. So a special algorithm is used to reduce the amount of storage and runtime. This method is based on the fact that matrices with banded structure are closely related to difference equations. For the banded system, we apply a LU -decomposition without pivoting (to avoid fill in) to the coefficient matrix A and derive an interval iteration similar to the well known interval iteration used in case of dense matrices. Here, however, we do not use a full approximate inverse R , but rather the interval iteration will be performed by solving two systems with banded triangular matrices L and U . The banded triangular systems are solved with the special method for difference equations described in [4]. In case of point matrices the method is designed to give almost sharp enclosures for all components (large or small in modulus) of the solution vector. A different approach to compute an enclosure for the solution vector of a large linear systems with banded or arbitrary sparse coefficient matrix (which gives enclosures with respect to the infinity norm $\| \cdot \|_\infty$ only) is described in [6].

In addition to the implementation of the solution method in C-XSC, the program includes a small demonstration part (a driver) which can be used to solve some simple systems. First the program reads the number of lower and upper bands and then one value for each of the bands, i.e. initially a Toeplitz matrix is generated. In the next step, however, any number of elements of the matrix can be changed, such that arbitrary banded matrices can be entered. To change the element $a_{i,j}$, only i, j and the new value for this element must be entered. Changing of elements is finished by entering zeros for i and j . Next the right hand side must be entered. There are several choices of predefined solutions, such that the right hand side b will be determined from this given solution. Alternatively b can be set to a constant value in all components or all components can be entered successively. In any case, the values of the components of b may be changed again similarly as for the matrix. When no changes are done anymore, the solution algorithm starts. The banded solver is called and the solution and error statistics are printed. In this way it is quite easy to explore the our C-XSC solver. Examples solved with these solvers are showed in section 5.

4 Integration between C-XSC and MPI Libraries

As part of our research, we did the integration between C-XSC and MPI libraries on cluster computers. This step was necessary and essential for the future adaptation of our solvers to high performance environments. This integration was developed using, initially, algorithms for matrix multiplication in parallel environments of cluster com-

puters. We did some comparisons about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order, and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory. This integration has been developed on LabTeC Cluster at II-UFRGS (cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet; cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet). We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization [8]. This parallelization was developed with the tasks division among various nodes on the cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol. About the C-XSC programs executed on cluster, some changes were made in the programs for their correct use in this environment, mainly about how to manipulate *dotprecisions* variables (high accuracy variables of C-XSC).

5 Tests and Results

Measures and tests were made to compare the routines execution time in C/C++ language, C/C++ using MPI library, C/C++ using C-XSC library and C/C++ using C-XSC and MPI libraries. Our first tests were made to compare the routines execution time and your accuracy:

- scalar product: tests comparing the accuracy and execution time between programs in C/C++ language (scalar product) and C/C++ using C-XSC library (optimal scalar product). In the tables 1, 2 and 3 we showed the results of test with scalar product between two vectors a and b . The values of this vectors are:

$$a = [10^{50}, 1.25, 10^{50}, 1.1, \dots, 10^{50}, 1.25, 10^{50}, 1.1]$$

and

$$b = [1, 1, -1, -1, \dots, 1, 1, -1, -1]$$

with size of this vectors (n) equal to 30000, 90000 and 180000;

Order (n)	program in C/C++	program in C-XSC	Factor
30000	0.031205	0.041015	1.31
90000	0.096727	0.113847	1.18
180000	0.188670	0.225083	1.19

Table 1. Parallel Scalar Product (Optimal in C-XSC) $a \times b$ (using 4 nodes of cluster labtec – time in seconds)

- matrix multiplications: tests with sequential and parallel programs in C/C++ language, C/C++ using MPI library, C/C++ using C-XSC library and C/C++ using C-XSC and MPI libraries (see table 4);
- Methods to solve linear systems (Conjugate Gradient, Householder, Givens, Gauss-Seidel, Gauss Elimination, LU Decomposition, ...): tests with sequential version of these methods in C/C++ language and C/C++ using C-XSC library (only with high accuracy operations, not using interval arithmetic).
- solvers to linear systems with dense and banded matrices: tests with sequential versions using C-XSC library (interval versions);

A very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n . Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the least common multiple lcm_n of all denominators in H_n . Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For the system $(lcm_{10} H_{10})x = (lcm_{10} e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (5.2), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

Order (n)	$a \times b$	Exact Result
30000	−3.30000000000000002664535259100375697016716	1125
90000	−3.30000000000000002664535259100375697016716	3375
180000	−3.30000000000000002664535259100375697016716	6750

Table 2. Scalar Product $a \times b$ in C/C++ (results)

Order (n)	$a \times b$	Exact Result
30000	1124.999999999999317878973670303821563720703	1125
90000	3374.999999999998181010596454143524169921875	3375
180000	6749.99999999996362021192908287048339843750	6750

Table 3. Scalar Product $a \times b$ in C-XSC (results)

Program/Order	256×256	512×512
C/C++ with MPI	0.4172185	4.0385343
C-XSC with MPI	4.7586004	39.3353821

Table 4. Parallel Matrix Multiplication (using 8 nodes of cluster labtec – time in seconds)

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1.000000000000000E+002, & 1.000000000000000E+002 \\ -4.950000000000000E+003, & -4.950000000000000E+003 \\ 7.920000000000000E+004, & 7.920000000000000E+004 \\ -6.006000000000000E+005, & -6.006000000000000E+005 \\ 2.522520000000000E+006, & 2.522520000000000E+006 \\ -6.306300000000000E+006, & -6.306300000000000E+006 \\ 9.609600000000000E+006, & 9.609600000000000E+006 \\ -8.751600000000000E+006, & -8.751600000000000E+006 \\ 4.375800000000000E+006, & 4.375800000000000E+006 \\ -9.237800000000000E+005, & -9.237800000000000E+005 \end{pmatrix} \quad (5.2)$$

As other example, we compute an enclosure for a very large system. We take a symmetric Toeplitz matrix with five bands having the values 1, 2, 4, 2, 1 and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```

Dimension  n = 200000 Bandwidths 1,k : 2 2 A = 1 2 4 2 1
change elements ? (y/n) n b = 1 change elements ?
(y/n) n x =
  1: [ 1.860146067479180E-001, 1.860146067479181E-001 ]
  2: [ 9.037859550210300E-002, 9.037859550210302E-002 ]
  3: [ 7.518438200412189E-002, 7.518438200412191E-002 ]
  4: [ 1.160876404875081E-001, 1.160876404875082E-001 ]
  5: [ 1.003153932563721E-001, 1.003153932563722E-001 ]
  6: [ 9.427129202687645E-002, 9.427129202687647E-002 ]
  7: [ 1.028361799416204E-001, 1.028361799416205E-001 ]
  8: [ 1.005240450090008E-001, 1.005240450090009E-001 ]
  9: [ 9.874921290539136E-002, 9.874921290539138E-002 ]
 10: [ 1.004617422430963E-001, 1.004617422430964E-001 ]

199990: [ 1.001953939326196E-001, 1.001953939326197E-001 ]
199991: [ 1.004617422430963E-001, 1.004617422430964E-001 ]
199992: [ 9.874921290539136E-002, 9.874921290539138E-002 ]
199993: [ 1.005240450090008E-001, 1.005240450090009E-001 ]
199994: [ 1.028361799416204E-001, 1.028361799416205E-001 ]
199995: [ 9.427129202687645E-002, 9.427129202687647E-002 ]
199996: [ 1.003153932563721E-001, 1.003153932563722E-001 ]
199997: [ 1.160876404875081E-001, 1.160876404875082E-001 ]
199998: [ 7.518438200412189E-002, 7.518438200412191E-002 ]
199999: [ 9.037859550210300E-002, 9.037859550210302E-002 ]

```

```

200000: [ 1.860146067479180E-001, 1.860146067479181E-001 ]

max. rel. error = 1.845833860422451E-016 at i = 3 max.
abs. error = 2.775557561562891E-017 at i = 1 min. abs.
x[3] = [ 7.518438200412189E-002, 7.518438200412191E-002 ]
max. abs. x[1] =
[ 1.860146067479180E-001, 1.860146067479181E-001 ]

```

Our last example is about the matrix inversion (test about accuracy). In this example

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon^2 \end{pmatrix}, \text{ and } A^{-1} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} & 0 \\ 0 & \frac{1}{\epsilon} & 0 \\ 0 & 0 & \frac{1}{\epsilon^2} \end{pmatrix}.$$

With $\epsilon = 1.084202172485504E - 019$, our program in C-XSC obtained:

$$A = \begin{pmatrix} 1.0E + 000, 1.000000000000000E + 000, 0.000000000000000E + 000 \\ 0.0E + 000, 1.084202172485504E - 019, 0.000000000000000E + 000 \\ 0.0E + 000, 0.000000000000000E + 000, 1.175494350822288E - 038 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1.0E + 000, 1.000000000000000E + 000, 0.000000000000000E + 000 \\ 0.0E + 000, 7.136238463529799E + 044, 0.000000000000000E + 000 \\ 0.0E + 000, 0.000000000000000E + 000, 8.507059173023462E + 037 \end{pmatrix}.$$

In the results obtained until now, the execution time of the algorithms using C-XSC library are much larger than the execution time of the algorithms that do not use this library. Even in this initial tests, it is possible to conclude that the use of high accuracy operations make the program slower. It shows that the C-XSC library need to be optimized to have an efficient use on clusters, and make it possible to obtain high accuracy and high performance in this kind of environment [8].

6 Conclusions

In our research some programs were developed in C-XSC with the validated numeric paradigm, where the results are obtained with a good quality. The main contributions of our work are: integration between the libraries C-XSC and MPI; effective use of library C-XSC on cluster computers; and resolution of linear systems with high accuracy. In our work we provide the development of selfverifying solvers for linear systems of equations with dense and sparse matrices and the integration between C-XSC and MPI libraries on cluster computers. This integration was not trivial because was necessary to

send correctly the special high accuracy variables (*dotprecision*) of C-XSC to the cluster processors using the library MPI without loss of the high accuracy characteristic.

Nowadays, our software runs on the labtec cluster at UFRGS and the integration between C-XSC and MPI was done correctly. Our tests with matrix multiplication, scalar product and methods to solve linear systems of equations show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment [9]. Nowadays we are working in the implementation of parallel versions of methods to solve linear systems (without and with high accuracy). These methods (Conjugate Gradient, Householder, Givens, ...) are used in real life applications like hydrodynamic (parallel computational model with local refinement and dynamic load balancing for the simulation of substances transportation and hydrodynamic), agriculture (optimization of the air distribution in grain storehouse with aeration of the mass of grains) and in weather applications.

Acknowledgement: This work is supported in part by CAPES (Brazil) and DAAD (Germany). It is part of an international cooperation project between Brazilian universities (Universidade Federal do Rio Grande do Sul, Pontifícia Universidade Católica do Rio Grande do Sul and Universidade de Passo Fundo) and German universities (Universität Wuppertal and Universität Karlsruhe).

References

1. Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum 9, Springer-Verlag (1993).
2. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.
3. Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).
4. Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see <http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz>.
5. American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.
6. Rump, S. M.: *Validated Solution of Large Linear Systems*. In [1], pp 191–212 (1993).
7. Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.
8. Hölbis, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, 2002, Paris. Extended abstracts, pg. 54-55 (2002).
9. Hölbis, C.A., Diverio, T.A., Krämer, W.: An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix. In: International Parallel Computing, 2003, Dresden. Book of Resumes, pg. 58 (2003).

Finite Fields Multiplier based on Cellular Automata

Hyun-Sung Kim¹, and Il-Soo Jeon²

¹ Kyungil University, Computer Engineering,
712-701, Kyongsansi, Kyungpook Province, Korea
kim@kiu.ac.kr

² Kumho Nat'l Inst. of Tech., School of Electronic Eng.,
730-703, Gumi, Gyungbuk, Korea

Abstract. This paper proposes two new multipliers to compute AB^2 based on cellular automata over finite field. First, a multiplier with a generalized irreducible polynomial is implemented. Then, a new algorithm and a circuit are proposed to reduce the size of the first multiplier. They are suitable for VLSI implementation and could be used in IC cards because they have a particularly simple architecture.

1 Introduction

Finite field or Galois fields play an important role in error-control coding, digital signal processing and cryptography [1-2]. The finite field $GF(2^m)$ is suitable for computation to implement computer architecture. Since, finite extension field $GF(2^m)$ of finite field $GF(2)$ has 2^m elements and these elements are composed of bits string of zeroes and ones [3-4]. The finite field operation is necessary to implement an efficient and low-complexity cryptosystem.

In 1984, Yeh, et al. [5] developed a parallel systolic architecture for performing the operation $AB + C$ in a general $GF(2^m)$. Semi-systolic array architecture in [6] was represented with the standard basis. Architecture to compute multiplication and inverse were represented with the normal basis [7]. Systolic power-sum circuit was presented in [8]. Since then, many bit-parallel systolic multipliers have been proposed. However, these multipliers are not efficient for cryptography application due to the system complexity. To reduce the system complexity, Itoh and Tsujii[9] designed two low-complexity multipliers for the class of $GF(2^m)$, based on the irreducible AOP (All One Polynomial) of degree m and the irreducible equally spaced polynomial of degree m . Later, Kim in [10] developed linear feedback shift register based multipliers with a low complexity of hardware architecture using the property of AOP. CA(Cellular automata), first introduced by John Von Neumann in the 1950s, have been accepted as a good computational model for the simulation of complex physical systems [11]. Kim et al. proposed an LSB-first multiplier using CA with a low latency [12]. Ku et al. in [13] proposed an MSB-first AB^2 multiplier using PBCA. However, all such previously designed systems still have certain shortcomings.

Accordingly, the purpose of this paper is to propose two new modular multipliers to compute AB^2 based on CA over $GF(2^m)$. Two multipliers deploy the mixture advantages from the previous architectures in the perspective of area and time complexity. First, a multiplier is implemented with a generalized irreducible polynomial. Then, a

new algorithm and a circuit are proposed to reduce the hardware requirement of the first one. This uses the property of irreducible all one polynomial as a modulus.

2 Background

This section provides the necessary operations in the public-key cryptosystem and a brief description of finite fields. These properties will be used to derive new multipliers.

2.1 Public-key Cryptosystem

ElGamal proposed a public-key cryptosystem[14]; it gets its security from the difficulty of calculating discrete logarithms in a finite field. To generate a key pair, first choose a prime, p , and two random numbers, g and x , such that both g and x are less than p , then calculate $y = g^x \text{ mod } p$. The public key is y, g , and p . Both g and p can be shared among a group of users. The private key is x . The ElGamal scheme can be used for both digital signatures and data encryption. The modular exponentiation over finite fields is a very critical operation to implement the public-key cryptosystem. So, it is necessary to see the operation in detail. Let C and M be elements of $\text{GF}(2^m)$, the exponentiation of M is then defined as $C = M^E, 0 \leq E \leq n$.

The exponent E , which is an integer can be expressed by

$$E = e_{m-1}2^{m-1} + e_{m-2}2^{m-2} + \dots + e_12^1 + e_0 \quad .$$

The exponent can also be represented with a vector $[e_{m-1} e_{m-2} \dots e_1 e_0]$. A popular algorithm for computing the exponentiation is the binary method [10]. The exponentiation of M can be expressed as

$$M^E = M^{e_0} (M^{e_1} (\dots (M^{e_{m-2}} (M^{e_{m-1}})^2) \dots)^2) \quad .$$

An algorithm for computing exponentiation is presented as follows :

[Algorithm 1] MSB-first Exponentiation

Input : $M, E, f(x)$

Output : $C = M^E \text{ mod } f(x)$

- 1 : if $(e_{m-1} == 1) C = M$ else $C = \alpha^0$
- 2 : for $i = m - 2$ to 0
- 3 : if $(e_i == 1) C = MC^2 \text{ mod } f(x)$
- 4 : else $C = \alpha^0 C^2 \text{ mod } f(x)$

The exponentiation can be computed using power-sum operations or AB^2 multiplications. Next section presents two new architectures for the operation with a significantly low complexity of operations.

2.2 Finite Fields

A finite field $\text{GF}(2^m)$ contains 2^m elements that are generated by an irreducible polynomial of degree m over $\text{GF}(2)$. A polynomial $f(x)$ of degree m is said to be irreducible if the smallest positive integer n for which $f(x)$ divides $x^n + 1$ is $n = 2^m - 1$ [1,10]. Let $f(x) = x^m + f_{m-1}x^{m-1} + \dots + f_1x^1 + f_0$ be an irreducible polynomial over

$\text{GF}(2)$ and α be a root of $f(x)$. Any field element $\text{GF}(2^m)$ can be represented by a standard basis such as $a = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \dots + a_0$, where $a_i \in \text{GF}(2)$ for $0 \leq i \leq m-1$. $\{1, \alpha, \alpha^2, \dots, \alpha^{m-2}, \alpha^{m-1}\}$ is an ordinary standard basis of $\text{GF}(2^m)$.

It has been shown that an AOP (All One Polynomial) is irreducible if and only if $m+1$ is a prime and 2 is a generator of the field $\text{GF}(m+1)$ [9-10]. The values of m for which an AOP of degree m is irreducible are 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, and 100 for $m \leq 100$. Let $ff(x) = x^m + x^{m-1} + \dots + x + 1$ be an irreducible AOP over $\text{GF}(2)$ and α be the root of $ff(x)$ such that $ff(\alpha) = \alpha^m + \alpha^{m-1} + \dots + \alpha + 1 = 0$. Then we have $\alpha^m = \alpha^{m-1} + \alpha^{m-2} + \dots + \alpha + 1, \alpha^{m+1} = 1$.

The reduction is often performed using the polynomial $\alpha^{m+1} + 1$. This property of the irreducible polynomial is very adaptable for PBCA (Periodic Boundary Cellular Automata) architecture. If it is assumed that $\{1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^m\}$ is an extended standard basis, the field element A can also be represented as $A = A_m\alpha^m + A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + \dots + A_0$, where $A_m = 0$ and $A_i \in \text{GF}(2)$ for $0 \leq i \leq m$. Here, $a = A \pmod{ff(x)}$, where $ff(x)$ is an AOP of degree m , then the coefficients of a are given by $a_i = A_i + A_m \pmod{2}, 0 \leq i \leq m-1$.

3 AB^2 Multipliers

This section presents two new multipliers over $\text{GF}(2^m)$. These multipliers are based on cellular automata. First, we propose a new modular multiplier using a generalized irreducible polynomial with the ordinary AB^2 multiplication. Then, devise a new multiplication algorithm and a new multiplier, which uses the property of irreducible AOP as a modulus.

3.1 Generalized AB^2 Multiplier

Let a and b be the elements over $\text{GF}(2^m)$ and $f(x)$ be a modulus. Then each element over the ordinary standard basis is expressed as follows: $a = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \dots + a_0, b = b_{m-1}\alpha^{m-1} + b_{m-2}\alpha^{m-2} + \dots + b_0, f(x) = x^m + f_{m-1}x^{m-1} + f_{m-2}x^{m-2} + \dots + f_0$.

For step 3 of the Algorithm 1, the modular multiplication $p = ab^2 \pmod{f(x)}$ can be represented with a recurrence equation as follows: $p = ab^2 \pmod{f(x)} = a_0b + a_1[b\alpha^2 \pmod{f(x)}] + a_2[b\alpha^4 \pmod{f(x)}] + \dots + a_{m-1}[b\alpha^{2m-2} \pmod{f(x)}]$. Following shows the ordinary modular multiplication.

[Algorithm 2] Ordinary AB^2 Multiplication

Input : $a, b, f(x)$

Output : $p = ab^2 \pmod{f(x)}$

Initial value : $p^{(m)} = (p_{m-1}^{(m)}, p_{m-2}^{(m)}, \dots, p_0^{(m)}) = (0, 0, \dots, 0)$

1 : for $i = m-1$ to 0

2 : for $j = m-1$ to 0

$$\begin{aligned}
3 : \quad & b_j^{(i)} = b_{j-2}^{(i+1)} + b_{m-1}^{(i+1)} f_{j+1} + b_{m-2}^{(i+1)} f_j \\
4 : \quad & p_j^{(i)} = p_j^{(i+1)} + b_j^{(i+1)} a_i
\end{aligned}$$

The following are the basic operations for steps 3 and 4.

Step 3-1: m -tuple of b is circular shifting 2-bits to the left as follows:

$$(b_{m-3}', b_{m-4}', \dots, b_0', b_{m-1}', b_{m-2}') \leq (b_{m-1}, b_{m-2}, \dots, b_1, b_0)$$

Step 3-2: Apply modular operation with $b_{m-1}^{(i+1)} + b_{m-2}^{(i+1)} f_j$ as follows:

$$\begin{aligned}
(b_{m-1}, b_{m-2}, \dots, b_1, b_0) &\leq (b_{m-3}', b_{m-4}', \dots, b_0', b_{m-1}', b_{m-2}') + \\
&b_{m-2}' (f_{m-1}, f_{m-2}, \dots, f_1, f_0, 0) + b_{m-2}' (f_{m-1}, f_{m-2}, \dots, f_1, f_0)
\end{aligned}$$

Step 4: Multiply b to m -tuple of a_i , add it to m -tuple of p as follows:

$$(p_{m-1}, p_{m-2}, \dots, p_1, p_0) \leq (p_{m-1}, p_{m-2}, \dots, p_1, p_0) + a_i (b_{m-1}, b_{m-2}, \dots, b_1, b_0)$$

In order to perform step 3-1, an 1-dimensional PBCA, where extreme cells are adjacent, having m cells is used which is the upper part with the gray colored in Fig. 1. b is inputted into m cells of CA and CA has a characteristic matrix with the characteristic matrix T shown in Fig. 2 for the step 3-1. Fig. 1 shows the proposed generalized AB^2 multiplier using PBCA. It is possible to perform the multiplication in m clock cycles over $\text{GF}(2^m)$.

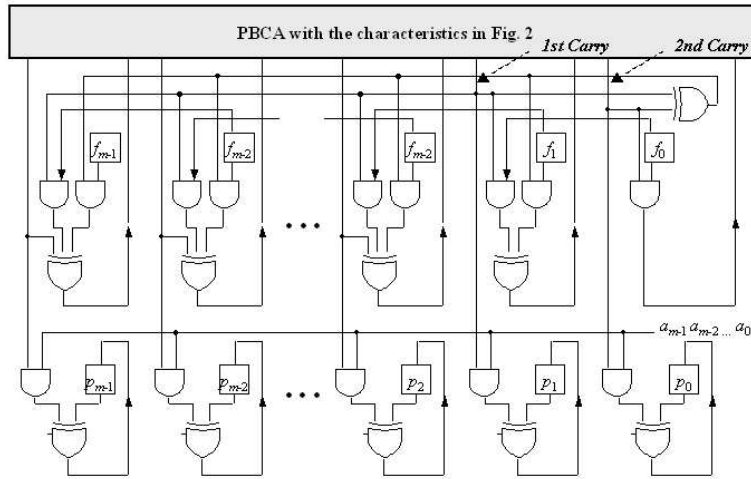


Fig. 1 Generalized AB^2 modular multiplier.

3.2 AOP AB^2 Multiplier

Let A and B be the elements over $\text{GF}(2^m)$ and $t(x)$ be a modulus which uses the property of an irreducible AOP. Then each element over an extended standard basis

$$T = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Fig.2 Characteristic matrices T .

is expressed as follows: $A = A_m\alpha^m + A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + \dots + A_0$, $B = B_m\alpha^m + B_{m-1}\alpha^{m-1} + B_{m-2}\alpha^{m-2} + \dots + B_0$, $t(x) = \alpha^{m+1} + 1$.

A new modular multiplication $P = AB^2 \bmod t(x)$ can be derived which applied the property of AOP as a modulus as follows : $P = AB^2 \bmod t(x) = A_0B + A_1[CLS_2(B)] + A_2[CLS_4(B)] + \dots + A_m[CLS_{2m}(B)]$, where circular shifting i -bits to the left is represented as $CLS_i()$. After the applying the property of AOP as a modulus, the modular reduction is efficiently performed with just $CLS_i()$ operation. Following shows the proposed AB^2 multiplication with AOP as an irreducible polynomial.

[Algorithm 3] Proposed AB^2 Multiplication

Input : A, B

Output : $P = AB^2 \bmod \alpha^{m+1} + 1$

Initial value : $P^{(m+1)} = (P_m^{(m+1)}, P_{m-1}^{(m+1)}, \dots, P_0^{(m+1)}) = (0, 0, \dots, 0)$

```

1 :   for  $i = m$  to 0
2 :        $B^{(i)} = CLS_2(B^{(i+1)})$ 
3 :       for  $j = m$  to 0
4 :            $P_j^{(i)} = P_j^{(i+1)} + B_j^{(i+1)} A_i$ 

```

The following are the basic operations for performing the modular multiplication from the above algorithm.

Step 2: $(m+1)$ -tuple of P is circular shifting 2-bits to the left as follows:

$$(B_{m-2}', B_{m-3}', \dots, B_0', B_m', B_{m-1}') \leq (B_m, B_{m-1}, B_{m-2}, \dots, B_0)$$

Step 4: Multiply A_i to $(m+1)$ -tuple of B and add it to $(m+1)$ -tuple of P as follows:

$$(P_m, P_{m-1}, P_{m-2}, \dots, P_0) \leq (P_m, P_{m-1}, P_{m-2}, \dots, P_0) + A_i(B_{m-2}', B_{m-3}', \dots, B_0', B_m', B_{m-1}')$$

In order to perform step 2, the 1-dimensional PBCA having $m+1$ cells is also used which is the upper part with the gray colored in Fig. 3. B is inputted into $m+1$ cells of CA and CA has the same characteristic matrix with the first architecture. Fig. 3 shows the proposed AOP AB^2 multiplier using PBCA. It is possible to perform multiplication

in $m + 1$ clock cycles over $GF(2^m)$. Each cell has a lower hardware complexity than the first one.

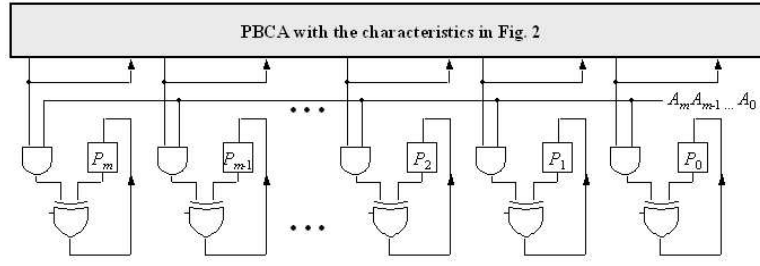


Fig. 3 AOP AB^2 modular multiplier.

4 Analysis

Proposed two AB^2 multipliers were simulated using ALTERA MAX+PLUSII simulation tool with FLEX10K device. Table. 1 shows a comparison between the proposed and the previous modular multipliers. For the comparison, it is assumed that n AND and n XOR represent n number of 2-input AND gate and XOR gate, respectively, and REG and Latch represents 1-bit register and latch, respectively. Comparison shows that the proposed architectures hybrid the advantages from previous architectures. Proposed two architectures have the good property in the perspective of hardware and time complexity compared with the architecture of Wei in [8]. Wei proposed a systolic multiplier with a latency of $3m$ and a critical path of 1-AND+2-XOR. Ku et al. in [13] proposed a multiplier based on cellular automata with the MSB-first fashion. The multiplier has similar property with our first multiplier but our multiplier has advantage in the critical path. Kim in [10] designed modular multipliers based on LFSR architecture. His architecture used the property of AOP as the irreducible polynomial.

5 Conclusion

This paper proposed two new AB^2 multipliers over $GF(2^m)$. They are based on cellular automata, especially PBCA. First, a multiplier with a generalized irreducible polynomial was implemented. Then, a new algorithm and its architecture were proposed to reduce the size of the first one. The new algorithm and architecture use the property of irreducible all one polynomial as a modulus. Proposed AB^2 multipliers hybrid the advantages from previous architectures.

Since the proposed multipliers have regularity, modularity and concurrency, they are suitable for VLSI implementation. The proposed multipliers can be used as a kernel circuit for public-key cryptosystems, which requires exponentiation, inversion, and division as their basic operation.

Table 1. Comparison for AB^2 multipliers.

Item Circuit	Function	Irreducible Polynomial	Number of cells	Latency	Hardware Complexity	Critical Path
Wei in [8]	AB^2+C	Generalized	m^2	$3m$	$3m$ AND $3m$ XOR $10m$ Latch	AND+2 XOR
Ku et al. in [13]	AB^2	Generalized	m	m	$3m-2$ AND $3m-2$ XOR $3m-1$ REG	AND+3 XOR
Kim in [10]	AB^2	AOP	m	$2m-1$	$2m$ AND $2m-1$ XOR $3m+4$ REG	AND +XOR(log ₂ m)
Gen. Mul.	AB^2	Generalized	m	m	$3m-2$ AND $3m-2$ XOR $3m$ REG	AND +2 XOR
AOP Mul.	AB^2	AOP	$m+1$	$m+1$	$m+1$ AND $m+1$ XOR $3(m+1)$ REG	AND+XOR

References

1. W. W. Peterson and E. J. Weldon, Error-Correcting Codes, Cambridge, MA: MIT Press, 1972.
2. D. E. R. Denning, Cryptography and data security Reading, MA: Addison-Wesley, 1983.
3. E. R. Berlekamp, Algebraic Coding Theory, New York: McGraw-Hill, 1986.
4. B. Benjauthrit and I. S. Reed, Galois switching function and their applications, IEEE Trans. Comput., vol. C-25, pp. 78-86, Jan. 1976.
5. C. S. Yeh, S. Reed, and T. K. Truong, Systolic multipliers for finite fields $GF(2^m)$, IEEE Trans. on Computers, vol. C-33, pp.357-360, Apr. 1984.
6. S. K. Jain and L. Song, Efficient Semisystolic Architectures for finite field Arithmetic, IEEE Trans. on VLSI Systems, vol. 6, no. 1, Mar. 1998.
7. J. L. Massey and J. K. Omura, Computational method and apparatus for finite field arithmetic, U. S. Patent application, 1981.
8. W. Wei, A systolic power-sum circuit for $GF(2^m)$, IEEE Trans. Comput., vol. 43, pp. 226-229, Feb. 1994.
9. T. Itoh and S. Tsujii, Structure of parallel multipliers for a class of finite fields $GF(2^m)$, Info. Comp., vol. 83, pp.21-40, 1989
10. H. S. Kim, Bit-Serial AOP Arithmetic Architecture for Modular Exponentiation, Ph. D. Thesis, Kyungpook National Univ., 2002.
11. V. Neumann, The theory of self-reproducing automata, Univ. of Illinois Press, Urbana London, 1966.
12. H. S. Kim and K. Y. Yoo, Multiplier for Public-key Cryptosystem based on Cellular Automata, LNCS 2776, pp. 442-445, Sept. 2003.
13. K. M. Ku, K. J. Ha, H. S. Kim, and K. Y. Yoo, New AB^2 Multiplier over $GF(2^m)$ using Cellular Automata, CATA 2003, pp. 283-286, March, 2003.
14. B. Schneier, Applied Cryptography - second edition, John Wiley and Sons Inc., 1996.

Efficient Systolic Architecture for Modular Multiplication over $GF(2^m)$

Hyun-Sung Kim¹ and Sung-Woon Lee²

¹ Kyungil University, Computer Engineering,
712-701, Kyungsansi, Kyungpook Province, Korea
kim@kiu.ac.kr

² Kyungpook Nat'l Univ., Computer Eng.,
702-701, Daegu, Korea

Abstract. This paper² presents a new modular multiplication algorithm and its systolic realizations in $GF(2^m)$. The proposed algorithm is based on the LSB-first scheme using a standard basis representation. From the proposed algorithm, a parallel systolic array is derived, and the architecture has a lower hardware complexity and smaller latency than the conventional approaches. Additionally, since the proposed architecture incorporates simplicity, regularity, and modularity, it is well suited to VLSI implementation and can be easily applied to the modular exponentiation architecture. Furthermore, the architecture will be utilized for the basic architecture of the crypto-processor.

1 Introduction

The arithmetic operations in the finite field have several applications in error-correcting codes[1], cryptography[2, 3], digital signal processing[4], and so on. Information processing in such areas usually requires performing multiplication, inverse/division, and exponentiation. Among these operations, the modular multiplication is known as the basic operation for public key cryptosystems over $GF(2^m)$ [2- 4]. Exponentiation is computed efficiently by the sequences of modular multiplications. And division and inverse can be regarded as a special case of an exponentiation because $B^{-1} = B^{2^m-2}$ [5, 6].

Recently, three types of multipliers over $GF(2^m)$ have been proposed that are easily realized using VLSI techniques. These are normal, dual, and standard basis multipliers, which have their own distinct features. The normal and dual basis multipliers need basis conversion, while the standard does not. In the following, we restrict our attention to the standard basis multiplier.

Numerous architectures for the modular multiplication in $GF(2^m)$ have been proposed in [7-10] over the standard basis. In 1984, Yeh et al. proposed two systolic array architectures with the LSB-first modular multiplication [7]. Wang et al. in [8] proposed two systolic architectures with the MSB-first fashion with less control problems as compared to [6]. Jain et al. proposed another multiplier architecture [9]. Its latency is smaller than those of other standard-basis multipliers, but there are broadcast lines in the circuit.

² This work was supported by the research fund of Kyungil University.

Wu et al. in [10] proposed bit-level systolic arrays with a simple hardware complexity with the MSB-first modular multiplication.

This paper proposes a new modular multiplication and its architecture with a parallel-in parallel-out systolic architecture. The proposed algorithm supports the LSB-first fashion. The proposed parallel architecture has a good time and area complexity compared to the previous multipliers.

2 Modular Multiplication

A finite field $GF(2^m)$ has 2^m elements and it is assumed that all the $2^m - 1$ non-zero elements of $GF(2^m)$ are represented using the standard basis. Let $A(x)$ and $B(x)$ be two elements in $GF(2^m)$ and $F(x)$ be the primitive polynomial, where $A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_1x + a_0$, and $B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \dots + b_1x + b_0$, where a_i and $b_i \in GF(2)$ ($0 \leq i \leq m-1$). A finite field of $GF(2^m)$ elements is generated by a primitive polynomial of degree m over $GF(2)$. Let $F(x)$ be an irreducible polynomial that generates the field and is expressed as $F(x) = x^m + f_{m-1}x^{m-1} + \dots + f_1x + f_0$. If α is the root of $F(x)$, then $F(\alpha) = 0$, and $F(\alpha) \equiv \alpha^m = f_{m-1}\alpha^{m-1} + \dots + f_1\alpha + f_0$, where $f_i \in GF(2)$ ($0 \leq i \leq m-1$).

To compute a modular multiplication, $AB \bmod F(x)$, the following equation is a common LSB-first algorithm

$$\begin{aligned} P &= AB \bmod F(x) \\ &= b_0[A\alpha^0 \bmod F(x)] + b_1[A\alpha^1 \bmod F(x)] + \dots + b_{m-1}[A\alpha^{m-1} \bmod F(x)] \end{aligned}$$

A new recursive multiplication is derived that is suitable for the systolic array implementation. The modular reduction is necessary because of the operation $[A\alpha^i \bmod F(x)]$ on the i -th step. We just try to concentrate on the modular reduction from the first term of the above equation, $b_0A \bmod F(x)$. The subsequent terms in the above equation are accumulated until reaching the end. The procedure of the new algorithm is as follows :

First of all,

$$P^{(1)} = b_0[A\alpha^0 \bmod F(x)] = [\sum_{k=0}^{m-1} a_k b_0 \alpha^k] \alpha^0 \bmod F(x).$$

However, in this step we can overlap the modular multiplication of $[A\alpha \bmod F(x)]$ for the next step operations. For the better understanding, we will use the symbol $A^{(i)}$ for the i -th snapshot for A , which has the value of $A\alpha^i \bmod F(x)$. Thereby, our algorithm computes two operations as follows

$$\begin{aligned} P^{(1)} &= b_0 A^{(0)} = [\sum_{k=0}^{m-1} p_k^{(1)} \alpha^k] \\ A^{(1)} &= A^{(0)} \alpha \bmod F(x) = [\sum_{k=0}^{m-1} a_k^{(0)} \alpha^k] \alpha \bmod F(x) = [\sum_{k=0}^{m-1} a_k^{(1)} \alpha^k] \end{aligned}$$

where

$$p_k^{(1)} = a_k^{(0)} b_0, (k = 0, 1, \dots, m-1)$$

$$a_k^{(1)} = a_{m-1}^{(0)} f_k + a_{k-1}^{(0)}, (k = 1, 2, \dots, m-1)$$

$$a_0^{(1)} = a_{m-1}^{(0)} f_0.$$

$$\begin{aligned} \text{In the general case, } P^{(i)} &= P^{(i-1)} + b_{i-1} A^{(i-1)} = [\sum_{k=0}^{m-1} (p_k^{(i-1)} + p_k^{(i)}) \alpha^k] \\ A^{(i)} &= A^{(i-1)} \alpha \bmod F(x) = [\sum_{k=0}^{m-1} a_k^{(i-1)} \alpha^k] \alpha \bmod F(x) = [\sum_{k=0}^{m-1} a_k^{(i)} \alpha^k] \end{aligned}$$

where

$$p_k^{(i)} = a_k^{(i-1)} b_{i-1}, (k = 0, 1, \dots, m-1)$$

$$a_k^{(i)} = a_{m-1}^{(i-1)} f_k + a_{k-1}^{(i-1)}, (k = 1, 2, \dots, m-1)$$

$$a_0^{(i)} = a_{m-1}^{(i-1)} f_0.$$

Finally,

$$P^{(m)} = P^{(m-1)} + b_{m-1} A^{(m-1)} = [\sum_{k=0}^{m-1} (p_k^{(m-1)} + p_k^{(m)}) \alpha^k]$$

where

$$p_k^{(m)} = a_k^{(m-1)} b_{m-1}, (k = 0, 1, \dots, m-1).$$

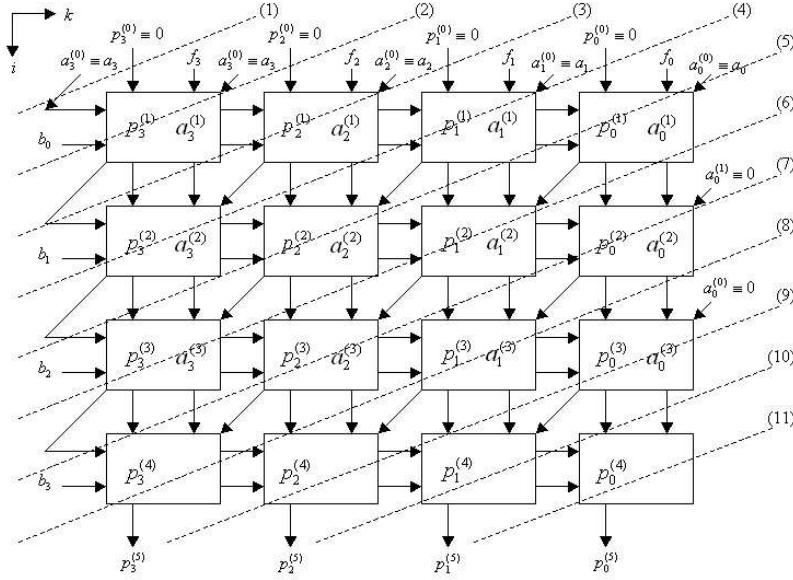


Fig.1 Dependency graph.

Thus, the multiplication of two elements A and B in $\text{GF}(2^m)$ can be computed using the above new recursive algorithm. From the algorithm, we can derive an efficient systolic multiplier by following the procedures in [8-9].

3 Systolic Modular Multiplier

This section proposes a parallel systolic multiplier. Fig.1 shows the dependency graph for our new multiplication over $\text{GF}(2^4)$. The inputs A and F enter the array in parallel from the top row, while B is from the leftmost column. The output P is transmitted from the bottom row of the array in parallel.

Fig. 3 (a) shows a basic cell in Fig. 1 for the general case where the circuit function is primarily governed by the following recurrence equation:

$$\begin{aligned} p_k^{(i)} &= a_k^{(i-1)} b_{i-1}, (k = 0, 1, \dots, m-1) \\ a_k^{(i)} &= a_{m-1}^{(i-1)} f_k + a_{k-1}^{(i-1)}, (k = 1, 2, \dots, m-1) \\ a_0^{(i)} &= a_{m-1}^{(i-1)} f_0 \end{aligned}$$

where $P^{(i)}$ is the i -th intermediate result of the product. Fig. 3 (b) shows a processing element for the case that the cell is located in the last row.

$$p_k^{(m)} = a_k^{(m-1)} b_{m-1}, (k = 0, 1, \dots, m-1).$$

By applying the cut-set systolization from Fig. 1, we obtain a parallel systolic multiplier as shown in Fig. 2 over $\text{GF}(2^4)$ with the processing elements in Fig. 3. Since the vertical path of each cell only requires two delay elements, except for the cells in the bottom row, the latency is $3m-1$.

Note that the processing element in the bottom row, Fig. 3 (b), is very simple and reduces the total cell complexity compared to the previous architectures.

4 Analysis

Our multiplier was simulated and verified using the ALTERA MAX+PLUSII simulator. Table 1 shows comparisons between the proposed architecture and the related circuits. We will give a comparison of systolic architectures with Yet et al.'s in [7] and Wang et al.'s in [8].

Before the comparison, it was assumed that AND and XOR_i denote a 2-input AND gate and i -input XOR, respectively. And T_{AND} and T_{XOR-i} are the propagation delay of a 2-input AND gate and i -input XOR gate, respectively.

As shown in Table 1, our multiplier has a good area and time complexity compared with the previous architectures.

5 Conclusion

This paper has explored a new algorithms for computing the modular multiplication into a low-complexity systolic architecture in $\text{GF}(2^m)$. A comparison between related sys-

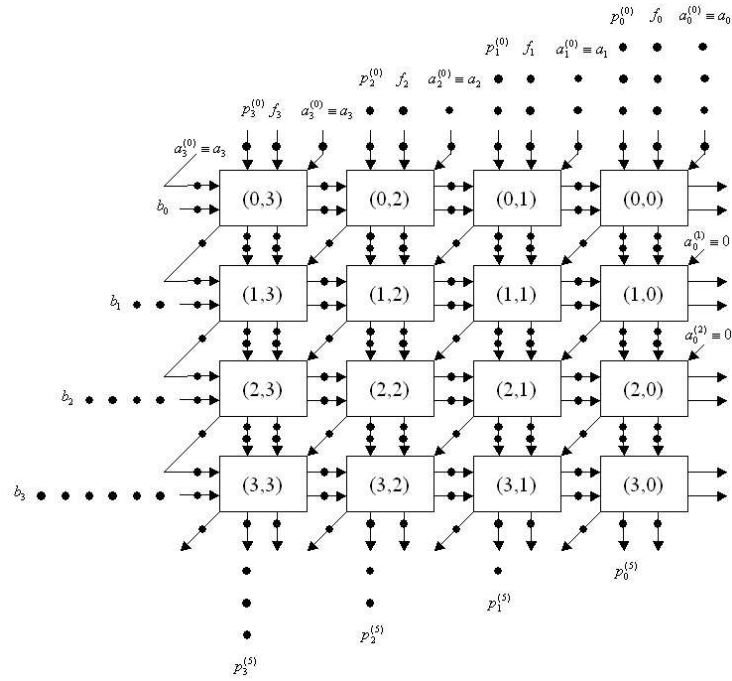


Fig.2 Proposed systolic multiplier.

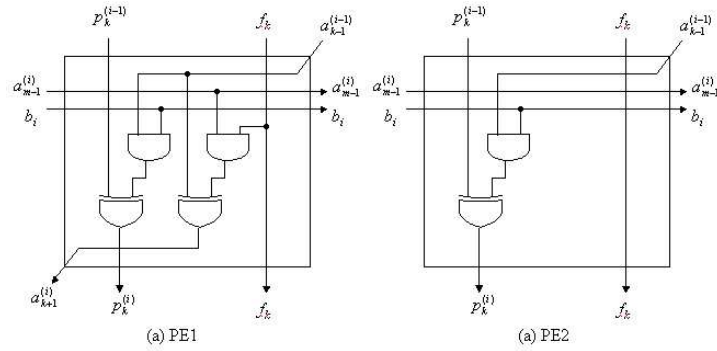


Fig.3 Processing elements.

Table 1. Comparison for AB^2 multipliers.

Circuit Item	Yeh et al. [7]	Wang et al. [8]	Proposed
No. of cells	m^2	m^2	m^2
Function	$AB+C$	$AB+C$	$AB+C$
Throughput	1	1	1
Latency	$3m$	$3m$	$3m-1$
Computation time per basic cell	$T_{AND} + T_{XOR-2}$	$T_{AND} + T_{XOR-3}$	$T_{AND} + T_{XOR-2}$
Cell complexity	2 2-input AND 2 2-input XOR 7 1-bit latches	2 2-input AND 1 3-input XOR 7 1-bit latches	PE 1 2 2-input AND 2 2-input XOR 7 1-bit latches
			PE 2 1 2-input AND 1 2-input XOR 2 1-bit latch
Algorithm fashion	LSB	MSB	LSB

tolic architectures reveals that the new systolic architecture has lower property than the conventional architectures for the hardware complexity and the latency. Furthermore, it can be used as the basic architecture for computing an inverse/division operation. Moreover, the architecture has a simplicity, regularity, and modularity. Thereby, it is well suited to VLSI implementation and it can be easily utilized for the crypto-processor chip design.

References

1. W.W.Peterson and E.J.Weldon, Error-correcting codes, MIT Press, MA, 1972.
2. D.E.R.Denning, Cryptography and data security, Addison-Wesley, MA, 1983.
3. A.Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, Boston, 1993.
4. I.S.Reed and T.K.Truong, The use of finite fields to compute convolutions, IEEE Trans. Inform. Theory, 21, pp.208-213, 1975.
5. H.S.Kim, Bit-Serial AOP Arithmetic Architecture for Modular Exponentiation, PhD. Thesis, Kyungpook National University, 2002.
6. S.W.Wei, VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF(2^m)$, IEEE Trans. Circuits and Systems, 44, pp.847-855, 1997.
7. C.S.Yeh, S.Reed, and T.K.Truong, Systolic multipliers for finite fields $GF(2^m)$, IEEE Trans. Comput., vol.C-33, pp.357-360, Apr. 1984.
8. C.L.Wang and J.L.Lin, Systolic Array Implementation of Multipliers for Finite Fields $GF(2^m)$, IEEE Trans. Circuits and Systems, vol.38, pp796-800, July 1991.
9. S. K. Jain and L. Song, Efficient Semisystolic Architectures for finite field Arithmetic, IEEE Trans. on VLSI Systems, vol. 6, no. 1, Mar. 1998.
10. C.W.Wu and M.K.Chang, Bit-Level Systolic Arrays for Finite-Field Multiplications, Journal of VLSI Signal Processing, vol. 10, pp.85-92, 1995.

11. S.Y.Kung VLSI Array Processors, Prentice-Hall, 1987.
12. K.Y.Yoo A Systolic Array Design Methodology for Sequential Loop Algorithms, PhD. thesis, Rensselaer Polytechnic Institute, New York, 1992.

Analyzing the Safety Problem in Security Systems using SPR Tool*

Il-Gon Kim¹, Jin-Young Choi¹, Peter D. Zegzhda², Maxim O. Kalinin², Dmitry P. Zegzhda², In-Hye Kang³, Pil-Yong Kang⁴ and Wan S. Yi⁴

¹ Dept. of Computer Science and Engineering, Korea University,
Seoul, Korea

{igkim, choi}@formal.korea.ac.kr

² St.-Petersburg State Polytechnical University,
St. Petersburg State, Russia

{zeg, max, dmitry}@ssl.stu.neva.ru

³ Dept. of Mechanical and Information Engineering, University of Seoul,
Seoul, Korea

inhye@uos.ac.kr

⁴ Korea Information Security Agency,
Seoul, Korea

{kangpy, wsyi}@kisa.or.kr

Abstract. In this paper, we address formal verification methodologies and develop SPR(Safety Problem Resolver) tool to verify security model's safety property. Using our technique it is possible to examine the protections of thousands of security-related objects on a multi-user system and identify security drawbacks. By acting on this information, security officers or system administrators can significantly reduce their system security exposure.

1 Introduction

Assurance that a system behavior will not result in the unauthorized access is fundamental to ensuring that the enforcing of the security policy will guarantee a system security. The greater the assurance, the greater the confidence that a security system will protect its assets against the threat with an acceptable risk. Most existing systems lack adequately secure interconnectability and interoperability. Each vendor has taken its own approach, relatively independent of the others. The revealing of all this vulnerable features comprises the goal of security evaluation process. There is very little understanding as to how security can be attained by integrating a collection of components, and even less understanding as to what assurance that security might provide.

A number of individual countries developed their own security evaluation standards[2]. In addition to, opening the way to universal standardization of security evaluation results, the new Common Criteria(CC)[3] has been developed. For example, CC defines seven level of assurance for security systems, rising from EAL1 to EAL7. To get a higher assurance over EAL5, developers of security systems require to specify security models and verify their safety properties using a formal and semi-formal approach.

* This work was supported by Korea Information Security Agency.

For high assurance systems, the difficulties of using formal methods add further complexity to both development and evaluation. However, given the lack of suitable mature, “industrial-strength tools” and the cost of a formal verification activity, informal approach generally represents a suitable compromise. This paper discusses a technique to evaluation of the security policy enforcement and a logical verification tool, SPR(Safety Problem Resolver). All of these allow specification of the system security-related elements and verification of the system safety.

This paper is structured as follows. Section 2 gives an overview of background of this paper. Section 3 introduces SEW (Safety Evaluation Workshop) structure to evaluate the safety of security model. Sections 4 illustrates SPR tool, the core of evaluation. In section 5, we explain the example of formal specification and verification for SACM(Simple Access Control Model) using SPR tool. Section 6 show the experimental results that SPR tool uses system resources. Finally, section 7 discusses conclusion and future directions.

2 Background

Since Harrison, Ruzzo, and Ullman showed that the safety problem was undecidable in common case[6], research has focused on determining whether safety could be decided for access control models with limited, but practical, expressive power. First, the take-grant model has a linear time safety algorithm, but there is still a significant difference in expressive power between take-grant and HRU[10]. Sandhu et al eliminates most of this difference in his models(SPM, TAM, ESPM, and non-monotonic ESPM)[11]. They demonstrated that an access control model could be designed for which safety is efficiently decidable (i.e., in polynomial time) given a few restrictions. Ultimately, despite proven expressive power and safety determination, these access control models have not been adopted in practice. We claim that there are two reasons for the lack of acceptance: (a) rather complex to use due to the subtlety of the restrictions and the complex relationship and (b) it is difficult to define the safety requirements and write practical algorithms that enforce these requirements.

With RBAC[8], access decisions are based on the roles that individual users have as part of an organization. Users take on assigned roles. The process of defining roles should be based on a thorough analysis of how an organization operates and should include input from a wide spectrum of users in an organization. Thus, we declare the determining whether the system implementing access control model is safe in the given state must be resolved for every system and every state.

In spite of the mentioned lacks, the great majority of the systems (e.g. operating systems, DBMS, Firewalls) uses DAC-based security models as the basis of access control mechanism. Thus, in general case safety cannot be verified for arbitrary DAC access control system. Therefore, the safety verification is the actual problem for the security evaluation, especially for the wide-spread computer systems.

In this paper we solve the safety problem proposing an universal specification and SPR model checking tool. SPR allows the analyzer to describe the system security elements and verify the security safety.

3 Safety Evaluation Workshop

According to principals of the computer system modeling, we use the term of *security model*[1][9] as the combination of system security states, transitions through access control rules, and constraints like the state security criteria. The access control model was first formulated by Lampson[4]. Access control model can be grouped into three main classes(DAC, MAC, RBAC) according to security policies[6][7][8]. The structure of the model is that of a state machine where each state is a triple (S,O,M) , where S is a set of subjects, O is a set of objects, and M is an access matrix which has one row for each subject, one column for each object, and is such that cell $M[s,o]$ contains the access rights. Fig. 3.1 shows an overall framework of SEW to evaluate the safety of *security model*.

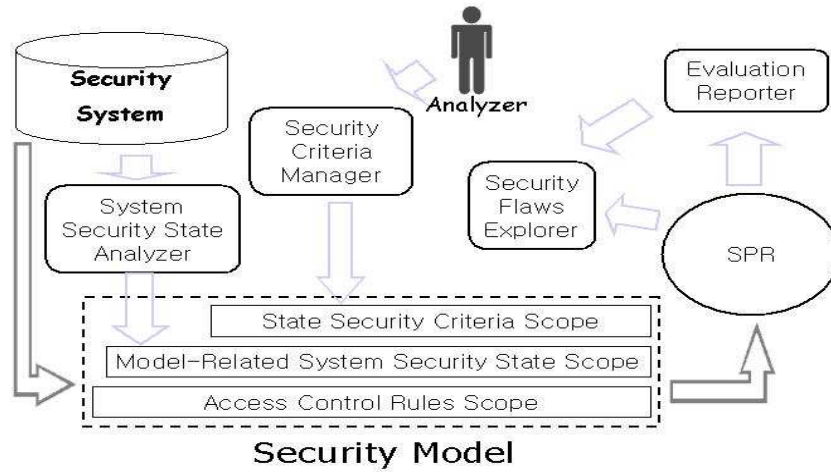


Fig.3.1. SEW(Safety Evaluation Workshop)

The type of security system which is mentioned in this paper refers to operating systems, IDSs, Firewalls, and etc. For safety evaluation of security systems, we propose SEW(Safety Evaluation Workshop) framework which consists of seven components. For the detailed information about SEW refer to [12].

- 1) System State Analyzer:
investigates the system being evaluated and builds the state of security model automatically according to the access control model.
- 2) Security Criteria Manager:
evaluator inputs the state security criteria in Security Criteria Manager.
- 3) Scopes:
 - (a) M3S (Model-related System Security) - scope:

- specifies the system security state and behavior in SPSL.
- (b) ACR (Access Control Rules) - scope:
describes access control rules in SPSL.
- (c) SSC (State Security Criteria) - scope:
expresses state security criteria in SPSL.
- 4) SPR:
formal verification tool implemented on C and SWI-Prolog[5]). SPR checks the initial system security state by the security criteria, then it generates all reachable states and checks them. The detailed information about SPR tool is described in Section 4.
- 5) SPSL(Safety Problem Specification Language):
specification language for security model with 3 scopes, based on Prolog syntaxes.
- 6) Security Flow Explorer:
demonstrates the sequence of the counter example events which leads to the security fault.
- 7) Evaluation Reporter:
produces the final report containing an access control model, a system, an initial state, access control rules, security criteria, an evaluation result, and a security flaws trace.

4 Safety Problem Resolver(SPR)

SPR is a major component of Safety Evaluation Workshop framework. It is a model checking tool for safety evaluation, which allows processing the given specifications of security systems and produces the verdict whether the security system is safe or not.

To specify the system security-related elements in security systems we have developed the SPSL(Safety Problem Specification Language). SPSL is a logical specification language to express a Model-related System Security Scope(security states), Access Control Rules Scope(access control rules), and Security Criteria Scope(security criteria) based on the Prolog-style syntax(SWI-Prolog usage)[5]. We have composed SPR tool which helps evaluator to process SPSL-based specifications of the system security and to verify whether system security policy has a safety problem. SPR checks the initial system security state by the given security criteria, then it generates all reachable states and checks them according to the criteria. The process of producing the sets of the reachable states and the evaluating of the criteria is called a *safety problem resolving*.

The definition of system and security model and the security evaluation algorithm in SPR tool can be formally described as follows;

Definition 1. A general system, $\Sigma = \{S^\Sigma, T, s_{init}^\Sigma, Q\}$, is a finite state machine where; S^Σ is the set of the system states. Q denotes the set of the safety evaluation query by system. T is the state transition function. $T: Q \times S^\Sigma \rightarrow S^\Sigma$ moves the system from on state to another. A safety evaluation query q is issued in the state s_i^Σ and moves the system to the next state $s_{i+1}^\Sigma = T(q, s_i^\Sigma)$.

Definition 2. A security model M consists of three elements, $M = \{S, R, C\}$, where; S denotes the set of the system security states defined by the model. R is the set of

the access control rules in the form of the logical predicates $r(s, s')$ defined on S and checking the the transition from s to s' meets to the access control model. C is the set of safety property in the form of logical predicates $c(s)$ defined on S and checking security of the state s .

Definition 3. A safety property can be written as $A = \{M, \Sigma, D\}$ where; M is system. Σ denotes the system. D is the mapping function, $D: S^\Sigma \rightarrow S$, which sets the relation between the system state and the system security states.

Algorithm 1. The security evaluation algorithm in SPR tool consists of three steps and it may be formalized as following.

Firstly, SPR evaluates a given system state $s_{init}^\Sigma \in S^\Sigma$ by the security criteria C ; if $\forall c \in C: c(s=D(s_{init}^\Sigma)) = \text{true}$ then a security system is secure.

Secondly, SPR proves that the system access control mechanisms realize the access control rules R ; if $\forall s_i^\Sigma, s_{i+1}^\Sigma \text{ in } S^\Sigma: s_{i+1}^\Sigma = T(q, s_i^\Sigma) \exists s_i = D(s_i^\Sigma), s_{i+1} = D(s_{i+1}^\Sigma)$ and $\forall r \in R: r(s_i, s_{i+1}) = \text{true}$ then a security system is secure.

Lastly, SPR generates the states $s_i^\Sigma \in S^\Sigma$ reachable from the given state $s_{init}^\Sigma \in S^\Sigma$ and evaluate their safety by the security criteria C ; if $\forall s_i^\Sigma \text{ in } S^\Sigma: s_i^\Sigma$ is reachable from $s_{init}^\Sigma, s_i = D(s_i^\Sigma): \forall c \in C: c(s_i) = \text{true}$ then a security system is secure.

5 Security Evaluation Example

For easy understanding of security specification and SPR's functionality technique, we show a very Simple Access Control Model(SACM) example in Fig. 3.2. Subject1, Subject2, Object1 and Object2 are in a High group. Subject3 and Object3 are involved in a Low group. In this example, Subject refers to a user, and Object points out a file. This example access control model should satisfy following safety properties: *No Read Up*, *No Write Down*. Satisfaction of these principles prevents information in high level objects to flow to objects at lower levels.

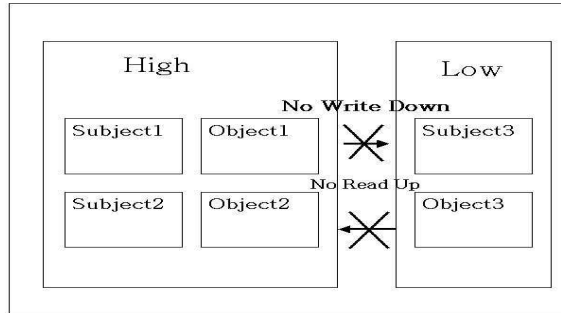


Fig. 5.2. Simple Access Control Model Example

Table 1. Simple Access Control Model's ACLs

subject \ object	Object1	Object2	Object3
High Group	read,write	read,write	read,write
Low Group			read,write
Subject1	read,write	read,write	read,write
Subject2	read,write	read,write	read
Subject3			read,write

5.1 Security States

Like in well-known operating systems, our *security states* are the collection of all entities of the system (subjects, objects) and their security attributes (access rights, ACLs, and so on). In this example model, we assume that a subject has each object in his own directory. SACM (Simple Access Control Model) includes users groups and files. User groups are divided into High and Low and they are noted as security subjects. Subject1, Subject2 and Subject3 represent three users. File system elements (files and directories) are regarded as security objects. Object1, Object2 and Object3 are noted as 3-scope files in each user's directory. The object access by subject depends on the ACLs shown in table. 1. System security states may be presented as the M3S (Model-related System Security Scope)-scope.

$$\text{Security States(M3S-scope)} = \text{Subjects} + \text{Objects} + \text{Security Attributes}$$

According to M3S-scope mentioned above, we specify SACM's security states using SPSL. Example 1 shows M3S-scope of SACM written in SPSL.

Example 1. M3S-scope of SACM

```

subjectAttr(subjectGroups) .
subject(s1, [subjectGroups(high)]) .
subject(s2, [subjectGroups(high)]) .
subject(s3, [subjectGroups(low)]) .
objectAttr(objectType) .
objectAttr(high) .
objectAttr(low) .
objectAttr(s1) .
objectAttr(s2) .
objectAttr(s3) .
object(o1, [objectType(file),
objectOwner(s1), high(rd, rp, wd, wp), low, s1(rd, rp, wd, wp),
s2(rp, rd, wd, wp), s3]) .
object(o2, [objectType(file), objectOwner(s2),
high(rd, rp, wd, wp), low, s1(rd, rp, wd, wp), s2(rd, rp, wd, wp), s3]) .
object(o3, [objectType(file), objectOwner(s3),

```

```
high(rd, rp), low(rd, rp), s1(rd, rp, wd, wp), s2(rd, rp),
s3(rd, rp, wd, wp)]).
```

Predicate *subjectAttr* means a subject security attribute. Attribute *subjectGroups* depicts the user membership in the groups. With predicate *subject* we declare an initial set of the subjects in the SACM. Predicate *subject* has two parameters: name of the subject and its attribute values. With predicate *objectAttr* we declare a set of the objects security attributes. The *objectType* attribute in Prolog list of predicates depicts type of objects. Attribute *objectOwner* describes the owner of object. Then there are five attributes with names of users and groups. With predicate *object* we declare an initial set of the objects in SACM. Predicate *object* has two parameters: name of the object and its attribute values. Therefore, the 11th line in example 1 can be interpreted as follows: “the object of *o1* is file. The owner of *o1* object is *s1*. High group has a access permission rights of *rp*, *rd*, *wd*, *wp* to *o1*. Low group possesses no access rights to *o1*. The subject of *s1* has a access rights of *rd*, *rp*, *wd*, *wp* to *o1*. The subject of *s2* possesses access rights of *rd*, *rp*, *wd*, *wp* to *o1*. The subject of *s3* has no access rights to *o1*.” In the same way any securable object of the real system can be specified. To automate this approach we have developed a special tool, e.g. State Analyzer for Windows, that investigates the system and forms the system security state.

5.2 Access Control Rules and Security Criteria

Access control rules express the restrictions on a system behavior. The system states transformation is possible after the access authorized by the system reference monitor(access control mechanism). It checks the authorization possibility against the security policy requirements represented by access control rules. In a SACM example, a subject can have the actions of *rd*, *rp*, *wd* or *wp*. The command *rd* allows the reading of directory entries, i.e. listing files and reading data stored in file. The command *rp* allows reading of privileges of the file. The command *wd* allows file creation in the directory. The command of *wp* allows modification of the privileges of the file. Such specification can be called ACR(Access Control Rules)-scope. Example 2 shows ACR-scope which contains security criteria in SACM. For want of paper space, we do not describe all of ACR-scope in SPSL.

Example 2. ACR-scope of SACM

```
testState1(S,O):- validSubject(S),isFile(O),canReadFile(S,O),
                  not(isGroupMember(S,high)),O=o1,O=o2.
testState2(S,O):- validSubject(S),isFile(O),canWriteFile(S,
                  o3),not(isGroupMember(S,low)).
```

The SPSL source code shown in example 2 above can be divided into two categories: one is for security policy requirements, the others define predicate prototypes for checking security criteria. *TestState1* and *TestState2* predicates represent the security criteria clauses for security policy requirement. *TestState1* depicts the “No Read Up” policy. *TestState2* denotes the “No Write Down” policy.

The *security criteria* allows customer or evaluator to check the secure and insecure states in the security model. The security criteria has the form of constraints which state the necessary conditions of the secure system state. The system is safe if $\bigcap_{i \in N} \overline{cr}_i = \text{true}$, where cr_i is an undesirable state criterion. The notation of \overline{cr}_i is the negation of cr_i . In other words, \overline{cr}_i which represents unsafe state should not be found in the security model.

5.3 Safety Evaluation Results Processing

We have SPR input with triple scopes(M3S-scope, ACR-scope, and SSC-scope) written in SPSL. Then we have run the executable program for SACM. After the running procedure, we have got two result files: security logical deduction trace (SRP.TRC) and evaluation report (SPR.REP). Example 3 represents the output file SPR.REP.

Example 3. SPR.REP of SACM

```
testState1 (_,_)      succeeded
testState2 (_,_)      failed
```

The result for checking *testState1* criteria is “succeeded”. It means that there is no subject in a Low group which can read any object in a High group. The evaluation verdict for *testState2* is “failed”. That means that SACM initial state is unsafe. After analyzing unsafe state, we found that *s1* in a High group has an access right permission of *wd*, *wp* to *o1* in a Low group. The wrong setting of ACLs for subjects and objects makes SACM violate its security requirements.

6 Experimental Results

The CPU time and memory usage in the model checking tool are very important, because they are primary factors which decide on the possibility of the verification of practical systems with a million reachable states.

We have implemented SPR tool on the Microsoft Windows 2000 and XP platforms. To reduce the time to verify the security model, SPR tool is optimized in the control of thread. Because the main thread has no need to monitor the child thread activity, the thread switching was reduced, and the number of Win32 API calls was decreased, too. Besides, the control over verification proceeding belongs now to Prolog thread. Thus, there are less time wasting and system calls, because the thread-control and Prolog-C data transfer operations became rare. Fig. 3.3 shows the CPU usage of SPR tool.

In Fig.3.3, we find that the quantity of the logical conclusion of Prolog depends on the CPU time(CPU is Intel Celeron-500 processor). SPR is the logical resolution machine based on Prolog language. Therefore, the major index of its productivity is dependent on the number of logical resolutions. Fig.3.4 shows the SPR memory usage.

In Fig.3.4, we find that the dynamic memory of the program is increasing with the rise of the logical conclusion, and SPR tool needs the minimum heap memory size of 500,000 bytes to be executed. In Fig.3.3 and Fig.3.4, the CPU and memory usage of SPR are the linear function. It means that SPR tool can verify the safety of practical

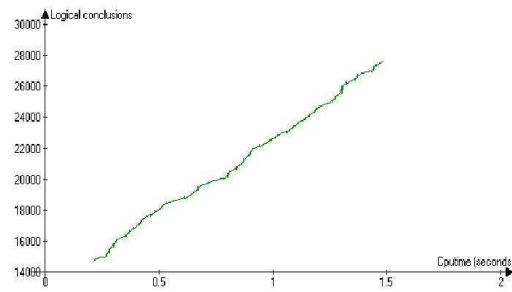


Fig. 6.3. SPR CPU usage

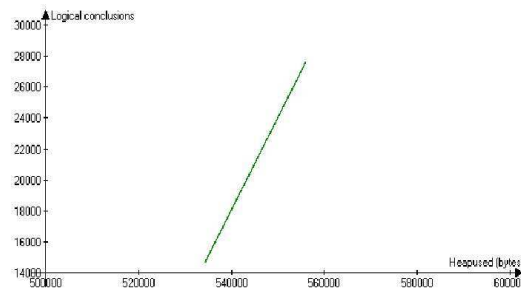


Fig. 6.4. SPR Memory usage

security systems. In practice, we extracted a access control based the security model from a Windows 2000 system. The code size of M3S scope in the security model is 15963 lines. In the case of the large security model, we confirm that SPR tool enables to verify the safety of them.

7 Conclusion

In this paper, we address formal specification and verification approach for security model. Then we illustrate SPR tool which enables to verify safety properties for security models, based on the security scopes. SPR tool helps security system's customers and evaluators to verify the safety of access control based security systems, because of these kinds of tools are rare in the security field. In addition, the experimental result of SPR tool shows that the SPR allows the safety problem resolving in practice. For future works, we will develop and elaborate the SEW components such as System State Analyzer, Security Criteria Manager, Security State Explorer to support easy modeling for system security and easy analysis for safety problems.

References

1. J. McLean, Security Model, In *Encyclopedia of Software Engineering*, Wiley Press, 1994.

2. Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, Dec 1985.
3. National Institute of Standards and Technology, *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model*, Version 2.1. CCIMB-99, Aug 1999.
4. B.W. Lampson, In *5th Princeton Symposium on Information Science and Systems*, pages 437-443, Reprinted in *ACM Operating Systems Review*, 8(1):18-24, 1974.
5. J. Wielemaker, SWI-Prolog 5.2 Reference Manual, <http://swi-prolog.org>, July 2003.
6. M.H. Harrison, W.L. Ruzzo, and J.D. Ullman, Protection in operating systems, *Communications of the ACM*, 19(8):461-471, 1976.
7. L.J. LaPadula and D.E. Bell, Secure computer systems: A mathematical model, Technical Report ESD-TR-278, VOL.2, The Mitre Corp., Bedford, MA, 1973.
8. D. Ferraiolo and R. Kuhn. Role-based access controls, In *Proc. of the 15th NIST-NCSC National Computer Security Conference*, pages 554-563, Baltimore, MD, October 1992.
9. J. Goguen and J. Meseguer, Security Policies and security models, In *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Security Press, 1982.
10. M. Bishop and L. Snyder, The transfer of information and authority in a protection system, In *Proceedings of the 7th ACM Symposium on Operating System ally Principles*, pages 45-54, 1979.
11. S. Castano, M.G. Fugini, G. Martella, P. Samarati, *Database Security*, Addison-Wesley, 1995.
12. P.D. Zegzhda, D.P.Zegzhda, M.O.Kalinin, Logical Resolving for Security Evaluation, ACNS 2003, pp.147-156, 2003.

2

² This paper is supported by Pop-iT Fund as part of the NURI(New University for Regional Innovation) in the Kumoh National Institute of Techonology

A CBD-based SSL Component Model

Young-Gab Kim¹, Lee-Sub Lee², Dongwon Jeong³, Young-Shil Kim⁴, and
Doo-Kwon Baik¹

¹ Department of Computer Science and Engineering, Korea University, Anam-dong 5-ga, Seongbuk-gu, 136-701, Seoul, Korea, {ygkim, baik}@software.korea.ac.kr

² Dept. of Computer Engineering, Kumoh National Institute of Technology, 188, Shinpyung-dong, Gumi, Gyeongbuk, eesub@kumoh.ac.kr

³ Dept. of Informatics & Statistics, Kunsan National University, San 68, Miryong-dong, Gunsan, Jeolabuk-do, 573-701, Korea, djeong@kunsan.ac.kr

⁴ Division of Computer Science & Information, Daelim College, Bisan-dong DongAn-ku Anayang-si Kyounggi-do, 431-715, Korea, pewkys@daelim.ac.kr

Abstract. The service provider of cryptography solution has to use platform-dependent cryptographic APIs for developing their security facilities. In contrast, component developers need platform-independent ways to support their development, which have nothing to do with platform's cryptographic APIs. Therefore, it is desirable to develop a set of standardized security interfaces to provide adequate security mechanisms according to the required cryptographic concepts such as confidentiality and integrity, not to the specific cryptographic algorithms. Although we can use well-known SSL implementations such as OpenSSL or JSSE in the form of embedded functions for applications, they lack of customizing facilities for component environment. The CBD-based SSL component model proposed in this paper supports easy discovery and integration for business components as well as flexible configuration of cryptographic mechanisms according to platform's security policies.

1 Introduction

As information technology through the Internet is exponentially increasing, the security issues against the open system are sharply increasing. Recently, a secure sockets layer(SSL)[1] is generally used as a information security protocol. It provides privacy over the Internet and allows the client and server applications to communicate in a way that cannot be eavesdropped. The Eric Young's OpenSSL and the Sun Microsystems's JSSE(java secure socket extension)[2] support a SSL protocol function in the form of embedded functions for the applications. However, they are not flexible to customize security facilities for component environment. To support SSL function as an application, developers who provide cryptography solutions, have to use platform-dependent cryptographic API(application program interface). This means that the developers should be an expert about security and security API for SSL. Furthermore, SSL protocol has some problems. First, it doesn't support APIs-exchange function for component environment because the SSL protocol supported by the applications is used as the form of function in the applications. Second, all data, which are transported between a client

and a server should be encrypted in a established SSL connection[3,4]. For this reason, it can be occurred overhead by transition of a encryption of data.

To overcome this limitation, we propose the SSL component model. The SSL component is capable of supporting a serutity requirement independantly apart from a cryptographic APIs of the understructure, and is executable according to the platform's security policies as the form of security component.

The rest of this paper is organized as follows: Section 2 presents background related with SSL protocol. Section 3 proposes requirements for SSL Component and Section 4 presents the SSL component model using UML notation[5, 6]. Section 5 shows comparisons between the SSL protocol and the SSL component. Section 6 concludes the paper.

2 Background

The SSL protocol was developed by Netscape Communications Corporation to provide security and privacy over the Internet. The protocol supports server and client authentication. The SSL protocol is application independent, allowing protocols like HTTP(hypertext transfer protocol), FTP(file transfer protocol), and telnet to be layered on top of it transparently. Still, SSL is optimized for HTTP, for FTP, IPsec might be preferable. The SSL protocol is able to negotiate encryption keys as well as authenticate the server before data is exchanged by the higher-level application. The SSL protocol maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes[7].

The SSL protocol is composed of two layers showed in Fig. 1.

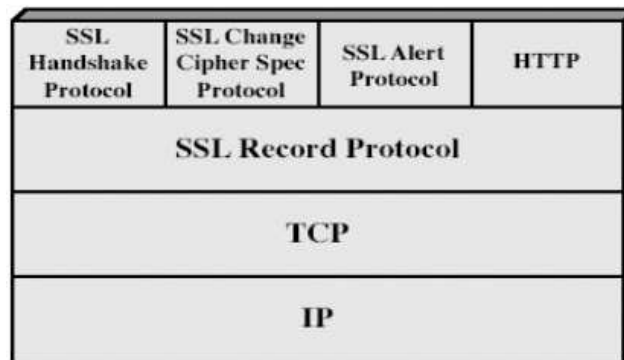


Fig. 2.1. The Structure of SSL Protocol[4]

At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher-level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption

algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher-level protocol can layer on top of the SSL Protocol transparently[4, 8].

However, as mentioned above, the SSL protocol has some limitations. First, the SSL protocol performs authentication and encryption on the data, which is transported between a client and a server. Therefore, it is expected to perform poorly when the size of data became large. As a result, it can be occurred lowering of efficiency and system overhead. Second, the SSL protocol has restricted algorithms, which don't include hash algorithm such as HAS-160[9] and SEED[10], which were created by Korea. Third, other countries cannot be supported of SSL protocol(e.g cryptographic key size) due to place restriction except for USA. Fourth, A number of minor flaws in the protocol and several new active attacks on SSL are discovered[11]. In this paper, we can overcome this limitation through the SSL component. More detail description will be presented in the following sections.

3 The Requirements for the SSL Component

As mentioned previous section, the SSL protocol do not directly adapt to the requirements of the security facilities in distributed environment. The SSL component model proposed in this paper will help developers of security application design and implement security facilities. To implement the SSL component, which support confidentiality and integrity, the SSL component should meet following requirements:

First, security programming should be required at a minimum in the code of application component, which exist or will be developed.

Second, the SSL component should be able to be customized in the form of deployment unit. At this point, the security properties can be a authentication path to decide a cipher suite, a certificate and trust.

Third, the message type defined in standard SSL protocol should be supported when we implement message protocol such as handshake, record protocol and so on.

Fourth, There are connection request of a client or a server, various key exchange between a client and a server according to the choice of cipher algorithm, request for certificate, and change of cipher spec.

Fifth, the confidentiality and integrity component, which implement the standardized security component interface, create a MAC or execute encryption in the record protocol. Therefore, the SSL component should be able to fragment a SSL message into transmission units, compress, create and encrypt a MAC, and add record header in the business component through communication channel.

Sixth, the SSL component have to maintain session state parameter such as session id, peer certificate, cipher spec, master secret and so on, and connection state parameter such as server/client random, write key, write MAC secret, IV(initial vector), sequence number and so on. Therefore, the SSL component should be implemented as EJB(enterprise java beans)[12] in the form of session bean, which has session and connection state.

4 Design of SSL Component

In this section, we design the SSL Component based on CBD(Component-Based Development) using a usecase diagram for requirement of SSL component, a sequence diagram for the message flow between SSL components, and a class diagram for the static view of SSL Component. The SSL component is capable of working together with Confidentiality Component and Integrity Component of KISA(Korea Information Security Agency)[13].

4.1 Usecase Diagram

Usecase diagram models a sequence of interactions between a user(that is, Actor) and the system, undertaken by the Actor in pursuit of a goal. As mentioned above, we use the usecase diagram to present requirement of SSL component. Therefore, we show the interactions between a business componet and the SSL component through the usecase diagram. Fig. 2 depicts the usecase scenario of SSL component.

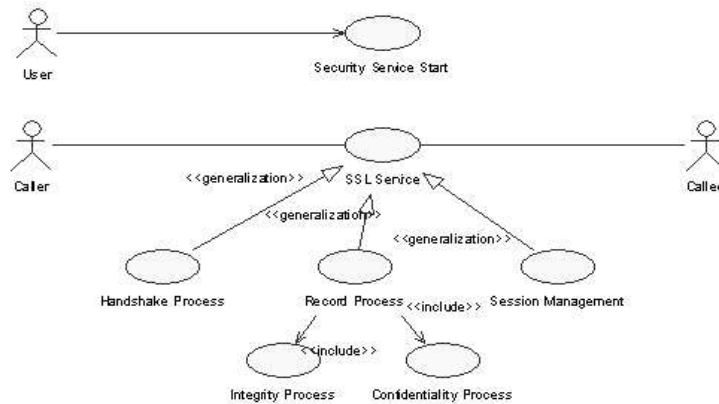


Fig. 4.2. The Scenario of SSL Component

The actor is divided into a user, who uses SSL component service, a caller, who use s a SSL client service, and a callee, who uses a SSL server service. The Security Service Start usecase start SSL component and the SSL Service usecase supply SSL protocol. The SSL Service usecase supports SSL security protocol through Handshake Process, Record Process, and Session Management.

Fig.3 shows the usecase of Handshake Process.

The Handshake Process usecase is composed of Hello Request, Key Exchange, and Cipher Spec. Exchange usecase. The Handshake Process constitutes the most complex part of the SSL component using the Hello Request, Key Exchange, and Cipher Spec usecases. It is used to initiate a session between the server and the client. Within the

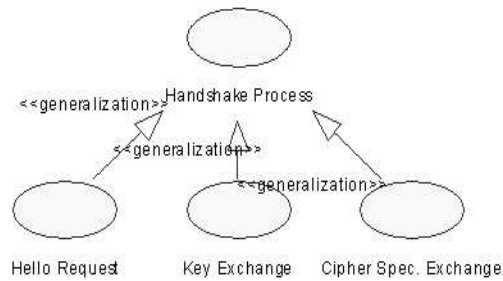


Fig. 4.3. The Usecase of Handshake Process

message of this protocol, various components such as algorithms and keys used for data encryption are negotiated. Due to this process, it is possible to authenticate the parties (that is, server and client) to each other and negotiate appropriate parameters of the session between them.

Fig. 4 shows the usecase of Record Process. The Record Process usecase is composed of MAC generation, Compression, and Encryption usecase.

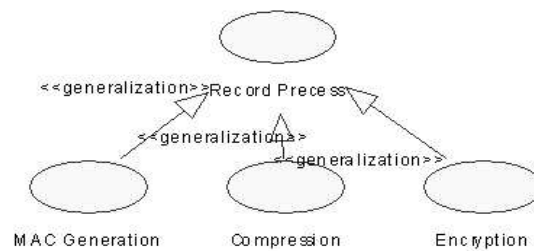


Fig. 4.4. The Usecase of Record Process

The SSL Record Process is used to transfer any data within a session - both messages and other SSL protocols (for example the handshake protocol), as well as for any application data.

4.2 Sequence Diagram

In this paper, we use a sequence diagram to show the message flow between SSL components. A Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behavior of a system.

Fig. 5 shows a sequence diagram of SSL component.

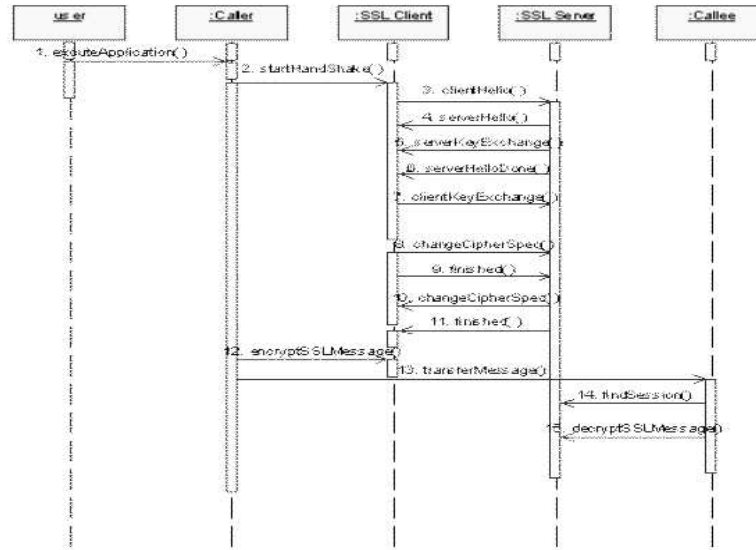


Fig. 4.5. Sequence Diagram of SSL Component

The user executes a SSL component, Security Service. A component, which wants a client service becomes a caller component. On the other hand a component, which wants a server service becomes a callee component. When the caller sends a message, which requests a SSL connection with remote method invocation(RMI), a SSL service system is executed. The SSL service in the SSL component executes a handshake process, a record process, and session management like the existing SSL protocol.

We summarize the main methods as follows:

- `executeApplication()` : executes a client component to act an application
- `requestSSLConnection()` : requests a connection with RMI to create SSL connection between caller and callee
- `startHandShake()` : starts handshake protocol to exchange a session key and algorithm
- `clientHello()` : client calls a server to start handshake protocol
- `serverHello()` : server calls a client to start handshake protocol
- `serverKeyExchange()` : server sends server's public-key
- `serverHelloDone()` : alarms an end of transmission of hello message
- `clientKeyExchange()` : client sends a client's public-key
- `changeCipherSpec()` : sends a cipher spec.
- `finished()` : alarms that the protocol finish.
- `encryptSSLMessage()` : encrypts data using a session key
- `decryptSSLMessage()` : decrypts data using a session key
- `sendData()` : sends an encrypted data

4.3 Class Diagram

In this section, we use the class diagram to describe the static and structural view of SSL component. Fig 6. depicts the relationship between SSL Component and client-server.

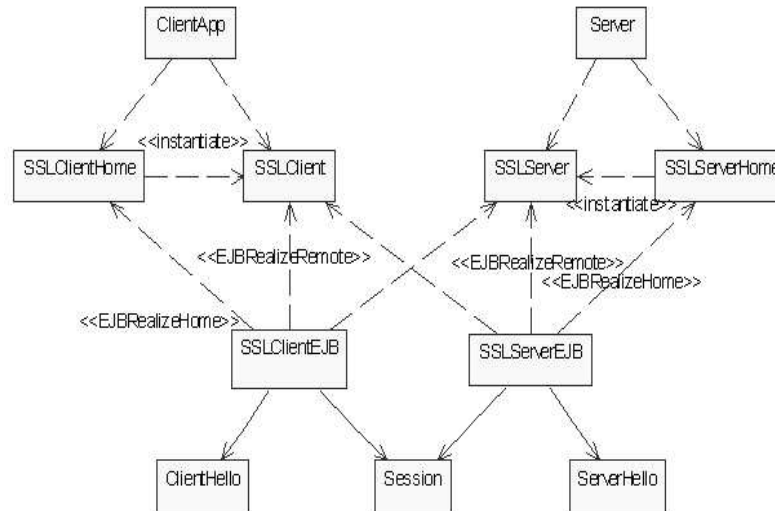


Fig. 4.6. Sequence Diagram of SSL Component

As shown above figure, SSL Component is consist of 4 structural elements: ClientApp, Server, SSLClient and SSLServer componet. In this class diagram, we omit the Confidentiality Component and the Integrity Component, which support confidentiality and integrity services using the handshake protocol and the record protocol as sub-structure of SSL Component.

5 Comparisons

When the solution providers develop the security facilities using the exiting SSL protocol, they have to use platform-dependent cryptographic APIs. That is, they should be an expert about security and security API for SSL. If not, it is possible that there are many vulnerability in the applications. Furthermore, it is expected to perform poorly when the size of data became large because the SSL protocol performs authentication and encryption on the data, which is transported between a client and a server. As a result, it cause the lowering of efficiency and system overhead. Thus, the SSL protocol has restricted algorithms, which don't include hash algorithm such as HAS-160 and SEED.

In contrast, in SSL component environment, developer needs platform-independent ways to support their development, which have nothing to do with platform's cryptographic APIs. The SSL component also guarantees the reuse of a component and

supports a security requirement independently apart from a kind of cryptographic API in the SSL component platform. Furthermore, it follows the standard of handshake protocol, record protocol, and so on in SSL protocol. Most of all, the SSL component can encrypt partial of data, and support diverse hash algorithm such as HAS-160 and SEED created by Korea. As a result, we can improve the efficiency of data transaction.

6 Conclusion and Future Work

In this paper, we proposed a SSL component model based on CBD. The SSL component provides confidentiality and integrity for any component that uses it as a secure communication facility. We can expect the compatibility of core security services and Return On Investment(ROI) by reusing the standardized security components. Furthermore, we can expect the ease of developing business components that require security facilities. The SSL component model can be applied to develop various component-based security applications such as e-commerce, e-government, and e-financial systems.

References

1. A.Freier, P Karlton, and P. Kocher: The SSL Protocol Ver 3.0 (1996)
2. Sun Microsystems, Release 1.4 of the Java 2 Platform, Standard Edition (J2SE), Sun Microsystems Inc.(2004)
3. K.Kant, R.Lyer and P.Mohapatra: Architectural Impact of Secure Socket Layer on Internet Servers. Proc. IEEE 2000 International Conference on Computer Design(2000)
4. William Stallings : Cryptography and Network Security-Principles and Practices, Wrox(2002)
5. Booch, G., Rumbaugh, J., and Jacobson, I. : The Unified Modeling Language User Guide. Addison Wesley Longman(1999)
6. John Cheesman, John Danielss: UML Components : Simple Process for Specifying Component-Based Software(The Component Software Series), Addison-Wesley(2001)
7. Matt Bishop: Computer Security Art and Science, Addison-Wesley (2002)
8. Ben Galbraith, et. al. : Professional Web Service Security, Wrox(2002)
9. TTA Standard : Hash Function Standard-Part 2: Hash Function Algorithm Standard(HAS-160), Telecommunications Technllogy Association(2000)
10. KISA: SEED Algorithm Specification. Korea Information Security Agency(1999)
11. D. Wagner and B. Schneier: Analysis of the SSL 3.0 Protocol, The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press, November (1996)
12. Sun Microsystems: Enterprise Java Beans Specification Version 2.0 Final Release, Sun Microsystems Inc.(2001)
13. KISA: The Standardization of Confidentiality and Integrity Service Component Interface, KISA(2003)

Exponentiation over $GF(2^m)$ For Public Key Crypto System using Cellular Automata

Kyo Min Ku¹, Kyeong Ju Ha², and Kee Young Yoo³

¹ Mobilab Co. Ltd, 952-3 Dongchun-dong, Buk-gu, Daegu, Korea 702-250
kmku@mobilab.co.kr

² Daegu Hanny University, 290 Yugok-dong, Gyeongsangbuk-do, Korea 712-715
kjha@dhu.ac.kr

³ Kyungpook National University, 1370 Sankyuk-dong, Buk-gu, Daegu, Korea 702-701
yook@knu.ac.kr

Abstract. This paper presents a MSB-first exponentiation over $GF(2^m)$ using a cellular automata. The AB^2 multiplication is an essential operation in modular exponentiation, which is the basic computation for most public key crypto system. For more efficient exponentiation, it is necessary to develop more fast AB^2 multiplier. We propose the AB^2 multiplier which is much more efficient in terms of time and space than that of previous researches on other architectures. And propose a new architecture for exponentiation which is more efficient in terms of space even though it is the same in terms of time. The proposed architecture of the exponentiation uses only one AB^2 multiplier. Since cellular automata architecture is simple, regular, modular and cascable, proposed architecture can be utilized efficiently for the implementation of VLSI.

1 Introduction

With the ever-increasing growth in data communication, the need for security and privacy has become a basic necessity. Cryptography is an essential requirement for communication privacy or the concealment of data in a data bank.

Finite field, $GF(2^m)$, arithmetic is fundamental to the implementation of a number of modern cryptographic systems[1]. Most public key crypto systems, for example, the Diffie-Hellman key exchange and ElGamal, are based on a modular exponentiation computation in a finite field[2][3]. Such modular exponentiation uses a modular multiplier as the basic structure for implementation. In addition, the Elliptic Curve Crypto system is based on constant multiplication[4], and the algorithms involved in the implementation of the multiplier include the LSB-first multiplication algorithm[5], MSB-first multiplication algorithm[6], and Montgomery algorithm[7].

The purpose of the current paper is to investigate and develop a simple, regular, modular, and cascable architecture for the VLSI implementation of exponentiation on $GF(2^m)$. Accordingly, Firstly a new structure is shown which facilitates AB^2 multiplication for effective exponentiation on $GF(2^m)$ using a cellular automata[8]. And then we propose new architecture for exponentiation using proposed AB^2 multiplier.

The proposed architecture for exponentiation is much more efficient in terms of space even though it is the same in terms of time[8].

The remainder of this paper is organized as follows. Chapter 2 outlines the concept of a cellular automata, while Chapter 3 shows the structure of AB^2 on $GF(2^m)$. Chapter 4 introduces the structure of the exponentiation using the AB^2 multiplier. Chapter 5 gives the analysis and final conclusions.

2 Cellular Automata

A Cellular Automata(CA) consists of numbers of interconnected cells arranged spatially in a regular manner[8][9]. The next state of a cell depends on the present states of ' k ' of its neighbors, for a k -neighborhood CA. Example of one rule of a 2-state 3-neighbor 1-dimensional CA is shown below.

State of neighbor : 111 110 101 100 011 010 001 000

Next state : 0 1 0 1 1 0 1 0 (Rule 90)

In this case, the state of the neighbors refers to 8 available states of 3 neighbors at time t . Among the 3 bits used to indicate the states, the middle bit represents the state of the cell itself, while the left and right bits indicate the states of the left and right neighbors, respectively. Rule 90 shows the state of the i^{th} cell at time $t + 1$, where 90 means the 8 bits of the next state shown in a decimal system. It can be seen that, in rule 90, the state of the cell is renewed in terms of the value resulting from the XOR of the state values of its left and right neighbors. Therefore, if it is assumed that $q_i(t)$ is the state value of the i^{th} cell at time t , rules 90 can be expressed in terms of the following equation: Rule 90 : $q_i(t + 1) = q_{(i-1)}(t) + q_{(i+1)}(t)$ where $+$ represents the XOR computation, $q_{(i-1)}$ represents the left neighbor of q_i , and $q_{(i+1)}$ represents the right neighbor of q_i .

3 AB^2 Multiplication Algorithm[8]

A concrete algorithm for implementing AB^2 multiplication in this paper is shown in Algorithm 1.

Algorithm 1 : AB^2 Multiplication Algorithm

$AB^2 (A(x), B(x), P(x))$

Input : $A(x), B(x), P(x)$

Output : $A(x)B(x)^2 \bmod P(x)$

Step 1 : $M(x) = 0;$

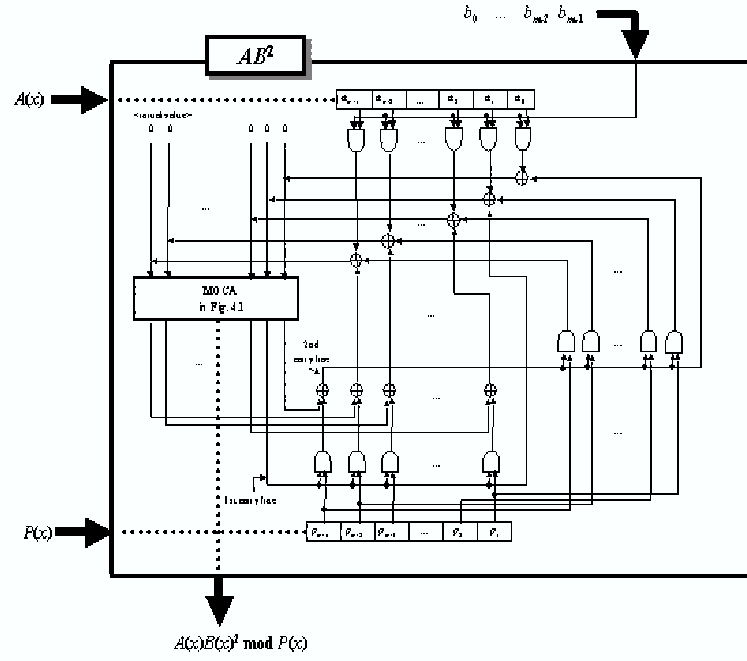


Fig. 3.1. Structure of AB^2 Multiplication using Cellular Automata.

Step 2 : for $i = m - 1$ to 0

Step 3 : $M(x) = M(x)^2 \bmod P(x) + A(x)b_i$

The entire structure is shown in Fig. 3.1. Each initial value is as follows:

- Initial values of cellular automata : all 0
- Initial values of A register : $A(x) = a_{(m-1)} \parallel a_2 \parallel a_1 \parallel a_0$
- Initial values of B register : $B(x) = b_{(m-1)} \parallel b_2 \parallel b_1 \parallel b_0$
- Initial values of P register : $P(x) = p_{(m-1)} \parallel p_2 \parallel p_1$

As such, it is possible to perform AB^2 multiplication in m clock cycles using m cells, $3m - 2$ AND gates, m 2-input XOR gates, $m - 1$ 3-input XOR gates, and 1 $m - 1$ -bit register, 2 m -bit registers if the structure shown in Fig. 3.1 is used.

4 MSB-first Exponentiation

The MSB-first exponentiation algorithm using Algorithm 1 is shown in Algorithm 2.

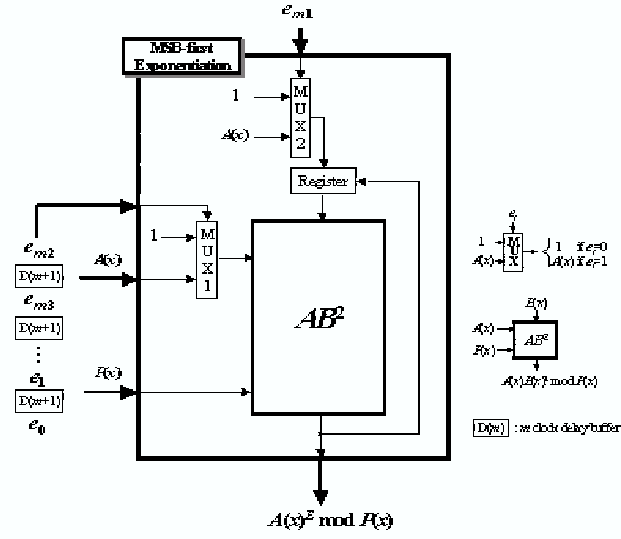


Fig. 4.2. Architecture for MSB-First Exponentiation using AB^2 Architecture.

Algorithm 2 : MSB-first $\text{EXP}(A(x), E, P(x))$

Input : $A(x), E, P(x)$

Output : $B(x) = A(x)E \bmod P(x)$

Step 1 : if $e_{(m-1)} == 1$ then $B^{(0)}(x) = A(x)$

else $B^{(0)}(x) = 0$

Step 2 : for $i = 0$ to $m - 2$

Step 3 : if $e_{(m-i-2)} == 1$

then $B^{(i+1)}(x) = AB^2(A(x), B^{(i)}(x), P(x))$

else $B^{(i+1)}(x) = AB^2(0, B^{(i)}(x), P(x))$

The architecture for exponentiation explained this paper is shown in Fig. 3.2.

The proposed structure was much more efficient in terms of space and time when compared to [9] and [10].

Structure	Systolic Array[9]	Proposed Paper
Operation	Exponentiation	Exponentiation
No. of basic cells	$(m - 1) AB^2$ multipliers	1 AB^2 multipliers*
No. of AND gates	$3m^2(m - 1)$	$(3m - 2)$
No. of XOR gates	$3m^2(m - 1)$	2-input: m 3-input: $(m - 1)$
No. of one bit latches	$11m^3 - 13m^2 + 2m$	$m^2 - m - 2$
No. of MUXes	m	2
No. of registers	0	$(m - 1)$ bit : 1 m bit : 3
Execution Time (Clock Cycles)	$3m(m - 1)$	$m^2 + m - 2$

* AB^2 multipliers : Architecture for $A(x)B(x)^2 \bmod P(x)$

Table 1. Comparison of MSB-first Exponentiation

5 Conclusions

Many cryptosystems have been developed up to the present time because of the important of the security and privacy is rapidly increased. The core computation of the cryptosystems is the exponentiation computation. In addition, the AB^2 multiplication is the basic structure for exponentiation.

This paper presented a new AB^2 multiplier in which AB^2 multiplication for effective exponentiation on $GF(2^m)$ can be performed and an architecture for exponentiation using proposed AB^2 multiplier. A new architecture for exponentiation using only 1 AB^2 multiplier proposed in this paper. It can perform the computation of MSB-first exponentiation in $m^2 + m - 2$ cycles. As a result, the proposed architecture for exponentiation is much more efficient in terms of space even though it is the same in terms of time than that of previous research based on systolic array[9]. Table 1 shows the comparison of architecture for exponentiation.

We can efficiently implement the division and the inversion architecture based on the proposed AB^2 multiplier. Furthermore, proposed architecture for exponentiation can be efficiently used for implementing public key crypto systems.

References

1. R.J. McEliece. Finite Fields for Computer Scientists and Engineering, *New York: Kluwer Academic*, 1987
2. W.Diffie and M.E.Hellman. New Directions in Cryptography, *IEEE Trans. On Info. Theory*, Vol.22 1976
3. T.ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. on Info. Theory*, VOL. 31(4), pp. 469-472, July 1985
4. A.J.Menezes. Elliptic Curve Public Key Cryptosystems, *kluwer Academic Publishers*, 1993.

5. C.S.Yeh, Irving S. Reed, T.K. Truong. Systolic Multipliers for finite Fields $GF(2^m)$, IEEE Transactions on Computers, Vol. C-33, NO. 4, pp. 357-360, April 1984
6. C.L.Wang, J.L. Lin Systolic Array Implementation of Multipliers for Finite Fields $GF(2^m)$, IEEE Transactions on Circuits and Systems, Vol.38, NO.7, pp. 796-800, July 1991.
7. P.L.Montgomery. Modular Multiplication without Trial Division, Mathematics of Computation, 44(170):519-521, April, 1985.
8. K.M.Ku, K.J.Ha, K.Y.Yoo. Design of New AB^2 Multiplier over $GF(2^m)$ using Cellular Automata. IEE Proceedings-Circuits, Devices and Systems Vol. 151, No 2 2004
9. S.W.Wei. VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in $GF(2^m)$, IEEE Trans. On Circuit & System II: Analog and Digital Signal Processing, VOL. 44, NO. 10, pp. 847-855, October, 1997.
10. C.L.Wang and J.H.Guo. New Systolic Arrays for $C + AB^2$, inversion, and division in $GF(2^m)$, IEEE Trans. On Computer, VOL. 49, NO. 10, October, 2000.

Area Efficient Multiplier based on LFSR Architecture

Jin-Ho Lee,¹ and Hyun-Sung Kim¹

Kyungil University, Computer Engineering,
712-701, Kyungsansi, Kyungpook Province, Korea
kim@kiu.ac.kr

Abstract. This paper² proposes a new LFSR multiplier for modular multiplication over $GF(2^m)$. The multiplier is based on an all one polynomial (AOP). Fenn et al. proposed two efficient LFSR multipliers, AOPM and MAOPM, using the property of AOP. AOPM is a multiplier with a result of an extended fields whereas MAOPM for an ordinary fields. They just focused on the time efficiency to derive MAOPM from AOPM. Therefore, they resulted about twice the hardware requirement from AOPM. Our main idea is contrast with the Fenn et al.'s idea. Since there are lots of applications with the strict hardware requirements, we focused on the area efficiency to derive a multiplier with the ordinary fields result. Therefore, we get a multiplier with almost the same hardware requirements with AOPM but with the additional constant clock cycles.

1 Introduction

Finite field $GF(2^m)$ arithmetic is fundamental to the implementation of a number of modern cryptographic systems and schemes of certain cryptographic systems [1,2]. The performance of elliptic curve cryptosystems is primarily determined by an efficient implementation of the arithmetic operations (addition, multiplication and inversion) in the underlying finite field [3]. Inversion can be carried out using just two modular multipliers or a power-sum modular architecture. Therefore, to reduce the complexity of elliptic curve cryptosystems, efficient architectures for a multiplication and a power-sum operation over $GF(2^m)$ are necessary.

An AOP (All One Polynomial) is used for irreducible polynomials to reduce the complexity of the field operations. Several architectures have already been developed to construct low complexity bit-serial and bit-parallel multiplications using AOP [6]-[10]. In 1989, Itoh and Tsujii designed two low complexity multipliers based on AOP and the irreducible equally spaced polynomial [6]. Since then many bit-parallel low complexity multipliers have been proposed for cryptographic applications. To increase computational time, Koc and Sunar designed multipliers with a low complexity which require m^2 AND gates and $m^2 - 1$ XOR gates [8]. In 1997, Fenn et al. presented two bit serial multipliers using LFSR (Linear Feedback Shift Register) architecture with a low area complexity [7]. Liu et al. in [9] and Lee et al. in [10] proposed bit-parallel AB^2 multipliers with the systolic architecture, respectively.

This paper proposes a new LFSR multiplier for the modular multiplication over $GF(2^m)$. The multiplier is based on an AOP. Fenn et al. proposed two efficient LFSR

² This work was supported by the research fund of Kyungil University.

multipliers, AOPM and MAOPM, using the property of AOP. AOPM is a multiplier with a result of one dimensional extended fields. Thereby, to get a result with the ordinary fields element MAOPM is proposed. They just focused on the time efficiency to derive MAOPM from AOPM. Therefore, they resulted about twice the hardware requirement from AOPM.

Our main idea is contrast with the Fenn et al.'s idea. Since there are lots of applications with the strict hardware requirements, we focused on the area efficiency to derive a multiplier with the ordinary fields result. Therefore, a multiplier is proposed with almost the same hardware requirement with AOPM's but requires additional time clock cycles. The additional clock cycles is not depended on the size of fields but fixed with a constant. Also, it can be generalized. Our multiplier could be used as a basic architecture for error-control coding, digital signal processing and cryptography.

2 Preliminaries

The public-key schemes and other applications are based on a modular exponentiation. Let B and M be elements of $\text{GF}(2^m)$, the exponentiation of M is then defined as $B = M^E$, $0 \leq E \leq n$, where $n = 2^m - 1$. For a special case, $M = \alpha$, the exponent E , which is an integer can be expressed by $E = e_{m-1}2^{m-1} + e_{m-2}2^{m-2} + \dots + e_12^1 + e_0$. The exponent also can be represented with a vector representation $[e_{m-1}e_{m-2}\dots e_1e_0]$. A popular algorithm for computing the exponentiation is the binary method [5]. Starting from the LSB(Least Significant Bit) of the exponent, the exponentiation of M can be expressed as $M^E = M^{e_0}(M^{2^1})^{e_1}(M^{2^2})^{e_2}\dots(M^{2^{m-1}})^{e_{m-1}}$. An algorithm for computing exponentiation is presented as follows :

[Algorithm 1] LSB-first Exponentiation Algorithm.

Input : $M, E, f(x)$

Output $C = M^E \bmod f(x)$

- 1: $T = M$
- 2: if $(e_0 == 1) C = T$ else $C = \alpha^0$
- 3: for $i = 1$ to $m - 1$
- 4: $T = TT \bmod f(x)$
- 5: if $(e_i == 1) C = CT \bmod f(x)$

The exponentiation can be computed using two multipliers or a multiplier and a squarer. Inversion can be regarded as a special case of the exponentiation because $M^{-1} = M^{n-1}$.

A finite field $\text{GF}(2^m)$ contains 2^m elements that are generated by an irreducible polynomial of degree m over $\text{GF}(2)$. A polynomial $f(x)$ of degree m is said to be irreducible if the smallest positive integer n for which $f(x)$ divides $x^n + 1$ is $n = 2^m - 1$ [4]. It has been shown that an AOP is irreducible if and only if $m + 1$ is a prime and 2 is a generator of the field $\text{GF}(m + 1)$ [6]. The values of m for which an AOP of degree m is irreducible are 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, and 100 for $m \leq 100$. Let $f(x) = x^m + x^{m-1} + \dots + x + 1$ be an irreducible AOP over $\text{GF}(2)$ and α be the root of $f(x)$. Then any field element $a \in \text{GF}(2^m)$ can be represented by a standard

basis such as $a = a_{m-1}\alpha^{m-1} + a_{m-2}\alpha^{m-2} + \dots + a_1\alpha^1 + a_0$, where $a_i \in GF(2)$ for $0 \leq i \leq m-1$. $\{1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^{m-1}\}$ is the standard basis of $GF(2^m)$. If it is assumed that $\{1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^m\}$ is an extended standard basis, the field element A can also be represented as $A = A_m\alpha^m + A_{m-1}\alpha^{m-1} + A_{m-2}\alpha^{m-2} + \dots + A_0$, where $A_m = 0$ and $A_i \in GF(2)$ for $0 \leq i \leq m$. Here, $a = A \pmod{f(x)}$, where $f(x)$ is an AOP of degree m , then the coefficients of a are given by $a_i = A_i + A_m \pmod{2}$, $0 \leq i \leq m-1$.

3 Bit-Serial architecture over $GF(2^m)$

This section gives a bit serial architecture for the modular AB multiplication. A new modular multiplier is proposed to improve Fenn et al. architecture in [7].

AB multiplication can be efficiently simplified based on using the AOP property of $\alpha^{m+1} = 1$ as follows

$$\begin{aligned} AB &= (A_m\alpha^m + A_{m-1}\alpha^{m-1} + \dots + A_0)(B_m\alpha^m + B_{m-1}\alpha^{m-1} + \dots + B_0) \\ &= P_m\alpha^m + P_{m-1}\alpha^{m-1} + \dots + P_0 \end{aligned}$$

For example, the multiplication of two elements $a = a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0$ and $b = b_3\alpha^3 + b_2\alpha^2 + b_1\alpha + b_0$ over $GF(2^4)$ where $\{1, \alpha, \alpha^2, \alpha^3\}$ is a standard basis, the product of a and b , $p = ab$, is given by $p = p_3\alpha^3 + p_2\alpha^2 + p_1\alpha + p_0$, $p_i \in GF(2)$ for $0 \leq i \leq 3$. Here we can define two elements A and B over an extended basis with $\{1, \alpha, \alpha^2, \alpha^3, \alpha^4\}$ as $A = A_4\alpha^4 + A_3\alpha^3 + A_2\alpha^2 + A_1\alpha + A_0$ and $B = B_4\alpha^4 + B_3\alpha^3 + B_2\alpha^2 + B_1\alpha + B_0$, where $A_i = a_i$ for $0 \leq i \leq 3$, $A_4 = 0$ and $B_i = b_i$ for $0 \leq i \leq 3$, $B_4 = 0$. Then the product $P = AB \pmod{\alpha^5 + 1}$ to be the multiplication shown in Fig. 1.

$A =$											
$\times B =$							A_4	A_3	A_2	A_1	A_0
							B_4	B_3	B_2	B_1	B_0
							A_4B_0	A_3B_0	A_2B_0	A_1B_0	A_0B_0
							A_4B_1	A_3B_1	A_2B_1	A_1B_1	A_0B_1
							A_4B_2	A_3B_2	A_2B_2	A_1B_2	A_0B_2
							A_4B_3	A_3B_3	A_2B_3	A_1B_3	A_0B_3
							A_4B_4	A_3B_4	A_2B_4	A_1B_4	A_0B_4
							P_8	P_7	P_6	P_5	P_4
							P_3	P_2	P_1	P_0	

(a) AB multiplication over $GF(2^4)$.

The multiplication in Fig. 1 uses the inputs with m bits while the outputs with $m+1$ bits. It requires an additional modular reduction after its operation. Thereby, the inputs and outputs of the algorithm with m bits are required. To achieve AB multiplication with m bits result, the operation from Fig. 1 (b) is modified as shown in Fig. 2. The value P_4 , which is the most significant bit of result, must be calculated before the computation

α^4	α^3	α^2	α^1	α^0
A_4B_0	A_3B_0	A_2B_0	A_1B_0	A_0B_0
A_3B_1	A_2B_1	A_1B_1	A_0B_1	A_4B_1
A_2B_2	A_1B_2	A_0B_2	A_4B_2	A_3B_2
A_1B_3	A_0B_3	A_4B_3	A_3B_3	A_2B_3
A_0B_4	A_4B_4	A_3B_4	A_2B_4	A_1B_4
P_4	P_3	P_2	P_1	P_0

(b) Simplified multiplication using the AOP property.

Fig.1. AB multiplication.

of the multiplication and used at an additional modular reduction as shown in Fig. 2. The most significant bit of the multiplication result over $\text{GF}(2^4)$ can be computed as follows: $P_4 = A_4B_0 + A_3B_1 + A_2B_2 + A_1B_3 + A_0B_4$.

To be an extended basis elements, the most significant bits of A and B are padded with 0 to become $A_4 = 0$ and $B_4 = 0$, respectively. Therefore, the above equation can be reformed as $P_4 = A_3B_1 + A_2B_2 + A_1B_3$.

α^3	α^2	α^1	α^0
P_4	P_4	P_4	P_4
A_3B_0	A_2B_0	A_1B_0	A_0B_0
A_2B_1	A_1B_1	A_0B_1	A_4B_1
A_1B_2	A_0B_2	A_4B_2	A_3B_2
A_0B_3	A_4B_3	A_3B_3	A_2B_3
A_4B_4	A_3B_4	A_2B_4	A_1B_4
P_3	P_2	P_1	P_0

Fig.2. Proposed algorithm for AB multiplication.

Example 1 Here, we verify the correctness of the configuration of the multiplication in Fig. 1. Let $a = x^3 + x$ and $b = x^2 + x + 1$ be elements in $\text{GF}(2^4)$, respectively. Then $p = ab = x^3 \bmod f(x)$, where $f(x) = x^4 + x^3 + x^2 + x + 1$. On the other hand, we have $A = (A_4, A_3, A_2, A_1, A_0) = (0, 1, 0, 1, 0)$ and $B = (B_4, B_3, B_2, B_1, B_0) = (0, 0, 1, 1, 1)$. Then the coefficients of P are given by

$$\begin{aligned}
P_4 &= 0 \times 1 + 1 \times 1 + 0 \times 1 + 1 \times 0 + 0 \times 0 = 1 \\
P_3 &= 1 \times 1 + 0 \times 1 + 1 \times 1 + 0 \times 0 + 0 \times 0 = 0 \\
P_2 &= 0 \times 1 + 1 \times 1 + 0 \times 1 + 0 \times 0 + 1 \times 0 = 1 \\
P_1 &= 1 \times 1 + 0 \times 1 + 0 \times 1 + 1 \times 0 + 0 \times 0 = 1 \\
P_0 &= 0 \times 1 + 0 \times 1 + 1 \times 1 + 0 \times 0 + 1 \times 0 = 1
\end{aligned}$$

Since $p_i = P_i + P_4$ for $0 \leq i \leq 3$, it follows that

$$\begin{aligned}
p_3 &= P_3 + P_4 = 0 + 1 = 1 \\
p_2 &= P_2 + P_4 = 1 + 1 = 0 \\
p_1 &= P_1 + P_4 = 1 + 1 = 0 \\
p_0 &= P_0 + P_4 = 1 + 1 = 0
\end{aligned}$$

The result is equal to $p = ab = x^3 \bmod f(x)$.

Fenn et al. in [7] proposed a modular multiplier, denoted by MAOPM, based on the ordinary modular multiplication shown in Fig. 1 and using the property of AOP as a modulus. Fig. 3 shows MAOPM for example 1 over $\text{GF}(2^4)$.

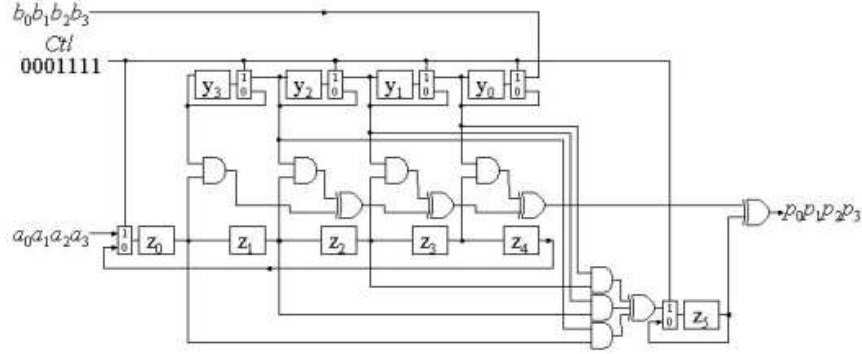


Fig.3. Fenn's MAOPM architecture over $\text{GF}(2^4)$.

MAOPM requires $2m - 1$ clock cycles with a complex hardware architecture. To get a better structure with a low hardware complexity, we designed a new modular multiplier using the proposed algorithm in Fig. 2.

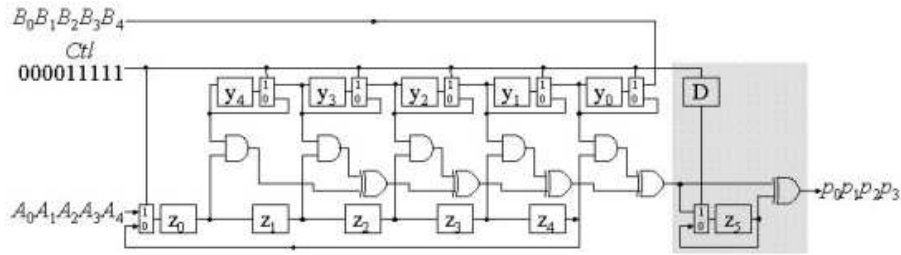


Fig.4. Proposed AB multiplier over $\text{GF}(2^4)$.

Fig. 4 shows the proposed architecture for AB multiplication. The difference between Fenn's and the proposed one is the right part of the architecture for an additional

modular reduction. Total $m + 1$ clock cycles are needed for data initialization for the z_i and y_i registers with input data A and B , respectively. At the last input clock cycle, it computes and outputs the first result. Total $m + 1$ clock cycles are needed for the data initialization and computation, respectively, but there exists a common clock between the last initial clock cycle and the first computation clock cycle. The total results are output after $2m + 1$ clock cycles over $\text{GF}(2^m)$. A control signal is required to distinguish the status of input from computation. The sequence of the control signal is composed of $m + 1$ ones and m zeroes.

4 Comparison and analysis

Proposed architecture was simulated by Altera's MAX+PLUSII. Table 1 shows a comparison of bit serial arithmetic architectures. Proposed AB multiplier is compared with a previous systolic architecture by Wang et al. and a LFSR architecture by Fenn et al. [7,11]. The proposed multiplier has significantly smaller area complexity than previous architectures. As a result, the proposed arithmetic architecture has a very good hardware complexity than the previous architectures. Therefore, if the architecture is used for some cryptographic applications, we can get a system with a great hardware complexity.

Table 1. Comparison of bit-serial multipliers.

Item Circuit	Function	Irreducible Polynomial	Number of cells	Latency	Hardware Complexity
Wang in [11]	AB	Generalized	m	$3m$	$3m$ AND $3m$ XOR $3m$ MUX $15m$ Latch
Fenn in [7]	AB	AOP	m	$2m-1$	$2m-1$ AND $2m-2$ XOR $m+2$ MUX $2m+2$ REG
Proposed multiplier	AB	AOP	$m+1$	$2m+1$	$m+1$ AND $m+1$ XOR $m+3$ MUX $2m+4$ REG

5 Conclusions

This paper presented a bit-serial modular multiplier with an irreducible AOP over $GF(2^m)$. Comparisons showed that the proposed architecture had certain advantages with the circuit complexity over previous architectures. Accordingly, the proposed multiplier can be used as a kernel circuit for public-key crypto-systems, which requires the operations of exponentiation, inversion, and division. It is easy to implement VLSI hardware and use in IC cards as it has a particularly simple architecture.

References

1. T.ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. on Info. Theory, vol. 31(4), pp. 469-472, July 1985.
2. W.Diffie, M.E.Hellman, New directions in cryptography, IEEE Trans. on Info. Theory, vol. 22, pp. 644-654, Nov. 1976.
3. A.J. Menezes, Elliptic Curve Public Key Cryptosystems, Boston, MA: Kluwer Academic Publishers, 1993.
4. R.J. McEliece, Finite Fields for Computer Scientists and Engineers, New York: Kluwer-Academic, 1987.
5. D.E. Knuth, The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Reading, MA: Addison-Welsey, 1969.
6. T. Itoh, S. Tsujii, Structure of parallel multipliers for a class of fields $GF(2^m)$, Info. Comp., Vol. 83, pp. 21-40, 1989.
7. S.T.J. Fenn, M.G. Parker, M. Benaissa, D. Tayler, Bit-serial multiplication in $GF(2^m)$ using irreducible all-one polynomial, IEE Proc. Comput. Digit. Tech., Vol. 144, No.6 pp. 391-393, 1997.
8. C. K. Koc, d B. Sunar, Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields, IEEE Trans. on Computers, Vol. 47, No. 3, pp. 353-356, March 1998.
9. C. H. Liu, N. F. Huang, C. Y. Lee, Computation of AB^2 Multiplier in $GF(2^m)$ Using and Efficient Low-Complexity Cellular Architecture, IEICE Trans. Fundamentals, Vol. E83-A, No. 12, pp. 2657-2663, Dec. 2000.
10. C.Y. Lee, E.H. Lu, J.Y. Lee, Bit-Parallel Systolic Multipliers for $GF(2^m)$ Fields Defined by All-One and Equally Spaced Polynomials, IEEE Trans. on Comp., Vol. 50, pp. 385-393, 2001.
11. C. L. Wang, J. L. Lin, Systolic Array Implementation of Multiplier for Finite Fields $GF(2^m)$, IEEE Trans. on Circuits and Systems, Vol. 38, pp. 796-800, July 1991.
12. N. Y. Kim, H. S. Kim, K. Y. Yoo, Computation of AB^2 Multiplication in $GF(2^m)$ using Low-Complexity Systolic Architecture, IEE Proc. Circuits, Devices and Systems, Vol. 150, pp. 119-123, 2003.

Efficient Authentication and Key Agreement for Client-Server Environment

Sung-Woon Lee¹, Hyun-Sung Kim², and Kee-Young Yoo³

¹ Tongmyong University of Information Technology, Dept. of Information Security,
Busan, 608-711, Rep. of Korea
staroun@tit.ac.kr

² Kyungil University, Dept. of Computer Engineering,
Kyongsansi, Kyungpook Province, 712-701, Rep. of Korea
kim@kiu.ac.kr

³ Kyungpook National University, Dept. of Computer Engineering,
Daegu, 702-701, Rep. of Korea
yook@knu.ac.kr

Abstract. In this paper, we present an efficient authenticated key agreement protocol called AKA, which provides mutual authentication and key agreement over an insecure channel between a client and a server. To increase the efficiency of the protocol, we further propose a parallelizable authenticated key agreement protocol called P-AKA by slightly modifying AKA. In our protocols, a client uses his own plaintext password to be authenticated from a server, while a server stores and uses a verifier for the client's password to authenticate him. The proposed protocols do not allow an adversary who compromises the server to directly impersonate the client. In addition, when the client wants to change his own password, he can freely do it without using other password changing schemes.

1 Introduction

It is necessary to verify the identities of communicating parties when they initiate a connection. This authentication is usually provided in combination with a key agreement protocol. Techniques for user authentication are broadly based on one or more of the following categories: (1) what a user knows, (2) what a user is, or (3) what a user has. Among them, the first category is the most widely used method due to the advantages of simplicity, convenience, adaptability, mobility, and less hardware requirement. It requires users only to remember their knowledge like a password. However, traditional password-based protocols are susceptible to off-line password guessing attacks (called dictionary attacks) since many users tend to choose memorable passwords of relatively low entropy.

Since Bellare and Merritt [1] presented a protocol called EKE for password-based authentication and key agreement which was resistant to off-line dictionary attacks, many password-based authenticated key agreement protocols have been proposed [2].

Corresponding author: Kee-Young Yoo (yook@knu.ac.kr)

Specially, augmented password-based authenticated key agreement schemes (usually called verifier-based protocol) [3, 4, 5, 6, 7] are suitable for client-server environment. In the verifier-based protocols, two parties (denoted client and server) use related password-based values to negotiate one or more shared ephemeral keys such that the shared keys are established if and only if they use values that correspond to the same password. Server uses password verification data (usually called verifier) that is derived from client's password data. The scheme forces an attacker who steals the password verification data to further perform a successful brute-force attack in order to masquerade as client.

In this paper, we present an efficient authenticated key agreement protocol called AKA, which provides mutual authentication and key agreement over an insecure channel between a client and a server. To increase the efficiency of the protocol, we further propose a parallelizable authenticated key agreement protocol called P-AKA by slightly modifying AKA. In our protocols, a client uses his own plaintext password to be authenticated from a server, while a server stores and uses a verifier for the client's password to authenticate him. The proposed protocols do not allow an adversary who compromises the server to directly impersonate the client. In addition, when the client wants to change his own password, he can freely do it without using other password changing schemes.

The remainder of this paper is organized as follows. In section 2, we propose AKA and P-AKA protocols. In section 3, we show security analysis of our protocols. In section 4, we compare them with the related protocols. Finally, section 5 gives our conclusions.

2 The proposed protocols

2.1 Notations

The following notations are used throughout this paper.

2.2 AKA

In this section, we present an efficient authenticated key agreement protocol which provides mutual authentication and key agreement over an insecure channel between a client and a server.

In our protocols, the server stores a verifier to verify a client's password. The verifier v is the information computed from a password π . We assume there is an initialization in which a client, called Alice, chooses a memorable password π , computes a verifier $v = g^{h(id, S, \pi)}$, and then sends her identity id and v to a server, called Bob, over a secure channel for registration. Bob stores (id, v) . To enhance the efficiency of the protocol, $v = g^{h(id, S, \pi)}$, where $h(\cdot)$ is a collision-free one-way hash function, and $h(id, S, \pi)^{-1}$ can be pre-computed before the protocol runs. We will omit 'mod p ' from expressions for simplicity. The steps for the proposed protocol are as follows:

1. Alice chooses $a \in_R Z_p^*$ and computes $X_A = g^a \oplus v$. Then, she sends her id and X_A to Bob.

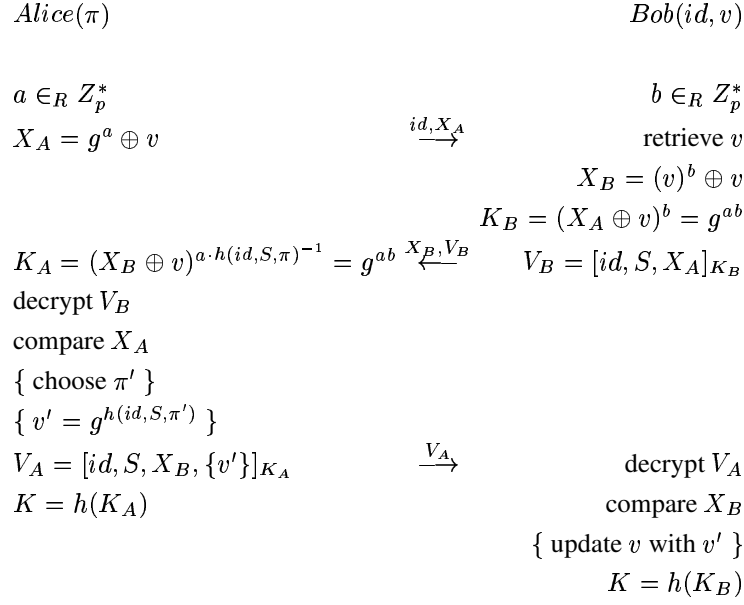
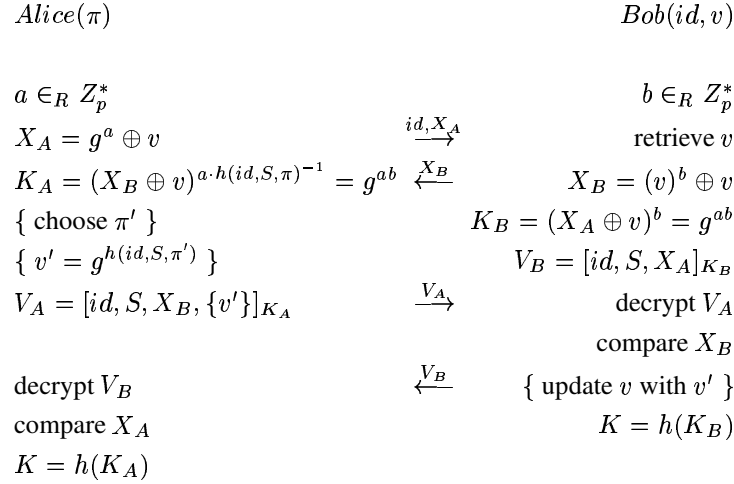
Notation Description	
id	A client's identity
S	A server's identity
π	A client's password
v	A verifier derived from a client's password
a, b	Session-independent random numbers
p	A large prime
g	A generator in the cyclic group Z_p^*
$h(\cdot)$	A secure one-way hash function
\oplus	Bit-wise exclusive-OR (XOR) operation
K	A session key
$[M]_K$	Encrypt M with key K using symmetric key algorithm
c^{-1}	Inverse of c on Z_p^*

Table 1. Notations

2. After receiving the message from Alice, Bob retrieves v from the verification table, chooses $b \in_R Z_p^*$, and computes $X_B = (v)^b \oplus v$, $K_B = (X_A \oplus v)^b = g^{ab}$, and $V_B = [id, S, X_A]_{K_B}$ in sequence. Then he sends X_B and V_B to Alice.
3. After receiving the message from Bob, Alice computes $K_A = (X_B \oplus v)^{a \cdot h(id, S, \pi)^{-1}} = g^{ab}$. Then, she decrypts V_B using K_A and checks whether the decrypted X_A is correct. If correct, she is convinced that Bob is authenticated. After that, she computes $V_A = [id, S, X_B]_{K_A}$ and sends it to Bob. When she wants to change her own password, she chooses a new password π' and computes $v' = g^{h(id, S, \pi')}$ and $V_A = [id, S, X_B, v']_{K_A}$, where v' is a new verifier for the new password.
4. After receiving the message from Alice, Bob decrypts V_A using K_B and checks whether the decrypted X_B is correct. If correct, Bob is convinced that Alice is authenticated. If v' was included in the value V_A , Bob updates v with v' in the verification table.
5. Finally, Alice and Bob compute a common session key $K = h(K_A) = h(K_B) = h(g^{ab})$, respectively.

2.3 P-AKA

AKA requires both Alice and Bob to compute a shared Diffie-Hellman key. This might take a long time due to exponentiations which are time-consuming operations, i.e., perhaps some seconds on slow device. However, the total execution time of the protocol can be speeded up if Alice and Bob can compute the time-consuming operations at the same time. Thus, we present a parallelizable key agreement protocol, called P-AKA, which can reduce the total execution time of the AKA. The idea to make AKA a parallelizable protocol is simple. P-AKA takes up message exchange of 4 steps in place of

**Fig. 2.1.** AKA protocol**Fig. 2.2.** P-AKA protocol

3 steps to quickly compute the message having influence on execution of the opposite party and to send it to him. Note that the definitions of all parameters are identical to AKA. The P-AKA protocol is given in Figure 2.

3 Security Analysis

For considering the security of our protocol, suppose that all communications among interacting parties are under control of an adversary called Eve as in [8]. That is, Eve can read the messages produced by the parties, provide her own messages to them, modify, delay, or replay them, and make new instances of any parties. The security of our protocols is based on the secure one-way hash function, the secure symmetric key algorithm, the difficulty of the discrete logarithm problem and the Diffie-Hellman problem [9]. We merely describes the security of P-AKA because AKA is almost similar with P-AKA.

1. Suppose that Eve sent a forged message X_A to Bob to masquerade Alice and received a response message X_B from Bob. However, Eve cannot compute K_A equal to K_B due to not knowing Alice's password π . Thus, Bob can detect this masquerading attack when he decrypts V_A using K_B and verifies the correctness of X_B .
2. Suppose that Eve received a message X_A from Alice to masquerade Bob. However, Eve cannot compute K_B equal to K_A due to not knowing the verifier v for Alice's password π . Thus, Alice can detect this masquerading attack when he decrypts V_B using K_A and verifies the correctness of X_A .
3. Off-line password guessing attack succeeds when there are pieces of information in communications that can be used to verify the correctness of the guessed password. Eve first eavesdrops the transmitted messages, X_A, X_B, V_A , and V_B , guesses a password candidate π' , and then tries to verify her guess by using the information. However, she has no way to verify her guess using them. Also, we consider the password guessing attack through active attacks and divide it into two cases. If Eve masquerades Alice, she may know a and $X_A = g^a$ made by herself and $X_B = (v)^b \oplus v$ sent from Bob. However, she cannot confirm the correctness of the guessed password due to not knowing b . If Eve masquerades Bob, she may know b and $X_B = g^b$ by herself and $X_A = g^a \oplus v$ and $V_A = [id, S, g^b]_{(g^{b \oplus v})^a \cdot h(id, S, \pi)^{-1}}$ sent from Alice. However, she cannot confirm the correctness of the guessed password due to not knowing a . Therefore, P-AKA is secure against the off-line password guessing attack.
4. Perfect forward secrecy is provided in the situation that even though the client's password is compromised, Eve cannot derive previous session keys. Suppose that Eve knows the password π . She tries to find previous session keys from the password and the information collected by eavesdropping in past communication sessions, i.e., π, X_A, X_B, V_A , and V_B . However, Eve has no way to compute $K = h(g^{ab})$ from the values. Therefore, P-AKA provides the property of perfect forward secrecy.
5. To be secure against the Denning-Sacco attack, the protocol should be designed such that even though a session key is compromised, Eve cannot compute the password and confirm the correctness of the guessed password. Suppose that Eve knows a session key $h(g^{ab})$. Eve tries to compute the password or confirm the correctness of the guessed password from the session key and the information collected by eavesdropping in past communication sessions, i.e., $h(g^{ab}), X_A, X_B, V_A$, and V_B . However, Eve has no way to do them from the values. Therefore, P-AKA is secure against the Denning-Sacco attack.

		A-EKE	B-SPEKE	SRP	AMP	PAK-X	AKA	P-AKA
Message exchange		5	4	4	4	3	3	4
Random number		2	3	2	2	3	2	2
Exponentiation	Alice	4	3	3	2	4	2	2
	Bob	4	4	3	3	4	2	2
	Parallel	6	6	4	3	8	4	3
hash function	Alice	2	2	4	5	5	1	1
	Bob	1	2	3	4	5	1	1
Symmetric enc./dec.		4	4	0	0	0	4	4

Table 2. Comparison with other related protocols

- The protocol being secure against stolen-verifier attack means Eve not being able to pose as a client after stealing a verifier of the client from the server. In P-AKA, if Eve gains the verification table, she may know Alice's verifier $v = g^{h(id, S, \pi)}$. However, she cannot directly pose as Alice because of not knowing $h(id, S, \pi)$. Therefore, P-AKA is secure against server compromise.

4 Efficiency Analysis

In this section, the protocol protocols are compared to the existing well-known protocols such as PAK-X[3], A-EKE[4], AMP[5], B-SPEKE[6], and SRP[7] which were submitted to IEEE 1363.2 [2]. Performance of key agreement protocols is usually approximated in terms of communication and computation loads. We compare them regarding with several efficiency factors such as the number of message exchanges, random numbers, exponentiations, hash functions, and symmetric encryption/decryption.

For the measure of total execution time, we will consider modular exponentiation, which is the most time-consuming operation in key agreement protocols. We use $E(\text{Alice:Bob})$ which means parallel execution for modular exponentiation between both parties. That is, one party is able to compute something while he or she is waiting for the other party's reply. AKA has 4E, $E(g^a: -)$, $E(-: (v)^b)$, $E(-: (X_A \oplus v)^b)$, and $E((X_B \oplus v)^{a \cdot h(id, S, \pi)^{-1}}: -)$, while P-AKA has 3E, that is, $E(g^a: -)$, $E(-: (v)^b)$, and $E((X_B \oplus v)^{a \cdot h(id, S, \pi)^{-1}}: (X_A \oplus v)^b)$. Here $;-;$ means no exponentiation.

Table 2 shows that AKA and P-AKA are very efficient compared with other key agreement protocols.

5 Conclusion

The password-based authentication protocols are the most widely used because of its advantages of simplicity, convenience, adaptability, mobility, and less hardware require-

ment. Users just need to remember simple information like a password. This paper proposed two efficient password-based authenticated key agreement protocols called AKA and P-AKA. They not only are secure against various attacks, and but also provides perfect forward secrecy. In addition, the proposed protocols are very efficient compared with other password-based key agreement protocols. In particular, when a client wants to change his own password, he can freely do it without using other password changing schemes.

Acknowledgement

This work was supported by the Brain Korea 21 Project in 2003.

References

1. S. Bellare and M. Merritt, Encrypted key exchange: Password-based protocols secure against dictionary attacks, In IEEE Symposium on Research in Security and Privacy, pp. 72-84, 1992.
2. IEEE, Standard Specifications for Public Key Cryptography, IEEE1363, 2002.
3. V. Boyko, P. MacKenzie, and S. Patel, Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman, In EUROCRYPT 2000, pp. 156-171, 2000.
4. M. Bellare, D. Pointcheval, and P. Rogaway, Authenticated Key Exchange Secure Against Dictionary Attacks, In EUROCRYPT 2000, pp. 139-155, 2000.
5. T. Kwon, Ultimate Solution to Authentication via Memorable Password, Presented to IEEE P1363a, 2000.
6. D. Jablon, Extended password key exchange protocols, WETICE Workshop on Enterprise Security, 1997.
7. T. Wu, Secure remote password protocol, Internet Society Symposium on Network and Distributed System Security, pp. 97-111, 1998.
8. M. Bellare and P. Rogaway, Random oracles are practical : A paradigm for designing efficient protocols, In 1st ACM Conference on Computer and Communications Security, pp. 62-73, 1993.
9. Schneier B, Applied cryptography. 2nded, John Wiley & Sons, Inc., 1996.

New Efficient Digit-Serial Systolic AB^2 Multiplier & Divider in $GF(2^m)$

Won-Ho Lee and Kee-Young Yoo

Department of Computer Engineering,
Kyungpook National University,
Daegu, 702-701, KOREA
purmi2@yahoo.co.kr
yook@knu.ac.kr

Abstract. This paper implements a new digit-serial systolic array for the computation of AB^2 multiplication and a new digit-serial systolic divider using the proposed systolic AB^2 multiplier in $GF(2^m)$ with the standard basis representation. The proposed systolic arrays have a significant improvement in reducing the AT complexity compared with previous architecture, although have one control signal. Furthermore, these arrays have regularity, modularity, and unidirectional data flow, and thus are well suited to VLSI implementation.

1 Introduction

The performance of an elliptic curve cryptography (ECC) is primarily determined by the efficient realization of the arithmetic operations in the underlying finite fields $GF(2^m)$. The design of circuits with high performance computing (HPC) to perform finite fields arithmetic operations is a matter of great practical concern. The important operations involved in finite fields $GF(2^m)$ are addition, multiplication, and division. Addition is very simple circuit if the field elements are presented in a polynomial form. However, the other operations are all much more complex. Therefore, coprocessors for ECC are most frequently designed to accelerate the field multiplication and division.

Numerous architectures for the arithmetic operations in $GF(2^m)$ have been reported in previous literature [1–7]. The conventional approaches for computing division in $GF(2^m)$ include the table lookup method, Euclid’s algorithm, and Fermat’s theorem based method. First, table lookup method is good for small values of m , but its high area complexity makes it difficult for VLSI implementation when m becomes large. Second, the Euclid’s algorithm finds the greatest common divisor (GCD) of two polynomials. Although this algorithm can be easily implemented using software, it would be too slow for time critical applications. Finally, Fermat’s theorem based method is using successive squaring and multiplication such as $A/B = AB^{-1} = AB^{2^m-2} = A(B(B \dots B(B(B^2) \dots)^2)^2)^2$. This method requires squaring and multiplication of $(m - 1)$ times, respectively. Therefore, the division and inversion operation can be performed by the iterative application of AB^2 multiplication.

A systolic arrays for performing the AB^2 multiplication using standard basis representation in $GF(2^m)$ have been proposed [3, 5–7]. Note that the systolic designs in [3]

and [5] have the bi-directional data flow, while the circuits in [6] and [7] have the uni-directional data flow. In this paper, we focus on the digit-serial systolic implementation of AB^2 multiplication and division in $GF(2^m)$ with the standard basis representation.

2 AB^2 Multiplication Algorithm in $GF(2^m)$

Let $A(x)$ and $B(x)$ be elements in $GF(2^m)$ with a primitive polynomial $G(x)$ of degree m , where

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0 \quad (2.1)$$

$$B(x) = \sum_{i=0}^{m-1} b_i x^i = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0 \quad (2.2)$$

$$G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i = x^m + g_{m-1}x^{m-1} + \cdots + g_1x + g_0. \quad (2.3)$$

The coefficients a_i , b_i , and g_i are the binary digits 0 and 1. As you know, the elements in $GF(2^m)$ can be represented by bit string of length m . For example, $A(x)$ can be represented by bit string $A = (a_{m-1}, a_{m-2}, \dots, a_1, a_0)$.

Define

$$P(x) = A(x)B^2(x) \bmod G(x) = p_{m-1}x^{m-1} + \cdots + p_1x + p_0. \quad (2.4)$$

Since $B^2(x) = b_{m-1}x^{2(m-1)} + b_{m-2}x^{2(m-2)} + \cdots + b_1x^2 + b_0 = B(x^2)$, we can derive

$$\begin{aligned} P(x) &= A(x)B(x^2) \bmod G(x) \\ &= \sum_{i=0}^{m-1} A(x)b_i x^{2i} \bmod G(x) \\ &= \left[\sum_{i=1}^{m-1} A(x)b_i x^{2i} \right] \bmod G(x) + A(x)b_0 \\ &= \left[\sum_{i=1}^{m-1} A(x)b_i x^{2(i-1)} \right] x^2 \bmod G(x) + A(x)b_0 \\ &= \left[\left[\sum_{i=2}^{m-1} A(x)b_i x^{2(i-1)} \right] + A(x)b_1 \right] x^2 \bmod G(x) + A(x)b_0 \\ &= \left[\left[\sum_{i=2}^{m-1} A(x)b_i x^{2(i-2)} \right] x^2 \bmod G(x) + A(x)b_1 \right] x^2 \bmod G(x) + A(x)b_0 \\ &\vdots \\ &= [\cdots [A(x)b_{m-1}]x^2 \bmod G(x) + A(x)b_{m-2}]x^2 \bmod G(x) + \\ &\quad \cdots + A(x)b_1]x^2 \bmod G(x) + A(x)b_0. \end{aligned} \quad (2.5)$$

Further expanding the last summations over i in (5), we obtain the following recursion for $P(x)$:

$$T_i(x) = T_{i-1}(x)x^2 \bmod G(x) + A(x)b_{m-i}, \quad (2 \leq i \leq m). \quad (2.6)$$

where $T_1(x) = A(x)b_{m-1}$, $P(x) = T_m(x)$, and

$$T_i(x) = t_{i,m-1}x^{m-1} + t_{i,m-2}x^{m-2} + \cdots + t_{i,1}x + t_{i,0}. \quad (2.7)$$

Substituting (7) into (6) yields

$$T_i(x) = t_{i-1,m-1}x^{m+1} \bmod G(x) + t_{i-1,m-2}x^m \bmod G(x) + \cdots + t_{i-1,1}x^3 + t_{i-1,0}x^2 + A(x)b_{m-i}, \quad (2 \leq i \leq m). \quad (2.8)$$

It is also easy to check that

$$\begin{aligned} x^m \bmod G(x) &= g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1x + g_0 \\ x^{m+1} \bmod G(x) &= g_{m-1}x^m + g_{m-2}x^{m-1} + \cdots + g_1x^2 + g_0x \\ &= (g_{m-1}g_{m-1} + g_{m-2})x^{m-1} \\ &\quad \cdots + (g_{m-1}g_1 + g_0)x + g_{m-1}g_0. \end{aligned} \quad (2.9)$$

Let $x^m \bmod G(x) \equiv g(x)$ and $x^{m+1} \bmod G(x) \equiv g'(x)$. Then with (9) and (10), we can rewrite the recursion given in (8) as follows:

$$T_i(x) = t_{i-1,m-1}g'(x) + t_{i-1,m-2}g(x) + \cdots + t_{i-1,1}x^3 + t_{i-1,0}x^2 + A(x)b_{m-i}, \quad (2 \leq i \leq m). \quad (2.11)$$

Based on (11), the AB^2 multiplication can be represented to the bit-wise recurrence equation as following algorithm:

Bit Level AB^2 Multiplication Algorithm

Input: $A(x), B(x), G(x)$

Output: $P(x) = T_m(x) = A(x)B^2(x) \bmod G(x)$

Initial: $(g'_{m-1}, \dots, g'_0) = (g_{m-1}g_{m-1} \oplus g_{m-3}, \dots, g_{m-1}g_0)$

1. **for** $i = 1$ **to** m **do**
2. **for** $j = 1$ **to** m **do**
3. **if** $i = 1$ **then**
4. $t_{i,m-j} = a_{m-j}b_{m-i}$;
5. **else**
6. **if** $j = m - 1$ **or** m **then**
7. $t_{i,m-j} = (a_{m-j}!?'b_{m-i}) \oplus (t_{i-1,m-1}!?'g'_{m-j}) \oplus (t_{i-1,m-2}!?'g_{m-j})$;
8. **else**
9. $t_{i,m-j} = t_{i-1,m-j-2} \oplus (a_{m-j}!?'b_{m-i})$
 $\quad \oplus (t_{i-1,m-1}!?'g'_{m-j}) \oplus (t_{i-1,m-2}!?'g_{m-j})$;

3 Digit-Serial Systolic Array Implementation

The dependence graph (DG) in $GF(2^m)$, obtained from the recurrence equation of above bit level AB^2 multiplication algorithm is shown in Fig. 1, where $m \times m$ basic nodes are used ($m = 4$). The cell structures of DG are shown in Fig. 2. In DG, the node means the point at which the computation occurs and the edge means the flow of data.

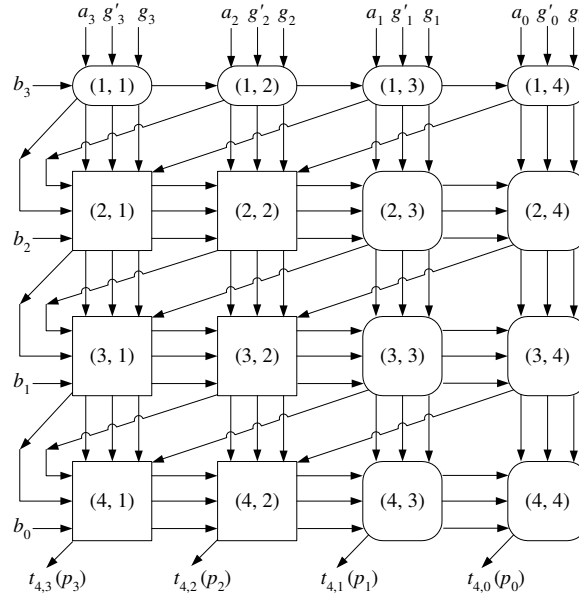


Fig. 3.1. The DG in $GF(2^m)$, where $m = 4$.

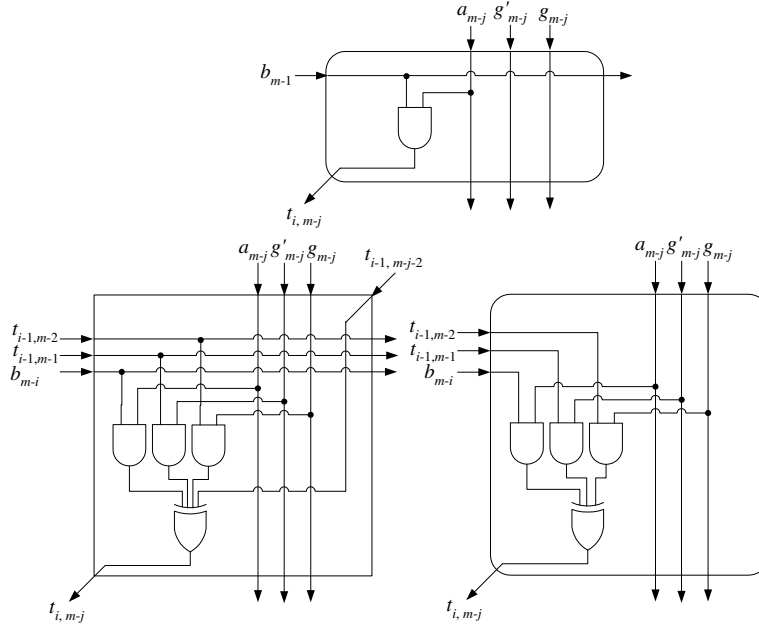


Fig. 3.2. The (i, j) cell structures in Fig. 1 ($1 \leq i, j \leq m$).

3.1 Digit-Serial Systolic AB^2 Multiplier

If we combine L adjacent basic cells in the horizontal direction to form a new cell, the DG can be modified as shown in Fig. 3, where L is the digit size ($2 \leq L \leq m - 1$). In other words, each row of the DG is partitioned into $N = \lceil m/L \rceil$ regions by combining only L basic cells in a horizontal direction together, and thus, a new modified DG consists of $m \times N$ digit cells.

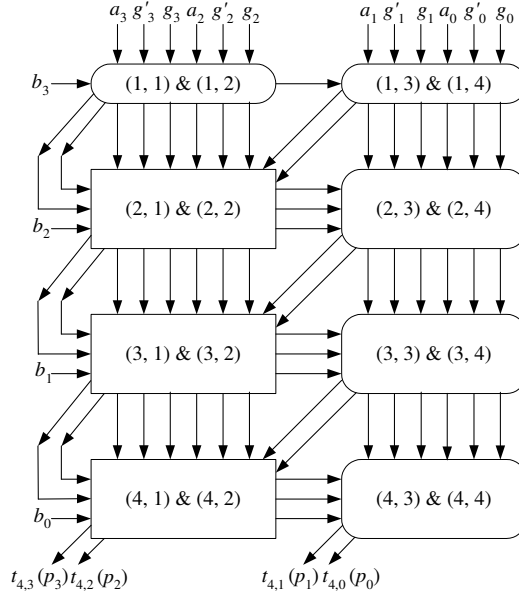


Fig. 3.3. A modified DG shown in Fig. 1, where $L = 2$.

By projecting the DG consists of $m \times N$ digit cells in an east direction following the projection procedure and cut-set systolization [8, 9], a new systolic AB^2 multiplier can be easily derived. Fig. 4 shows the digit-serial systolic array for AB^2 multiplication in $GF(2^m)$, where $m = 4$ and $L = 2$. It consists of m processing elements (PEs). The PE structures of Fig. 4 are shown in Fig. 5. The black square on the data flow means the buffer for one time step delay. As shown in Fig. 3, since the values $(b_i, t_{i,m-1}, t_{i,m-2})$ broadcasting to all the cells in each row exist, 2-to-1 multiplexers (MUX) and one-bit latches are added for this. These extra circuitry operations are controlled by a control signal (ctl). The sequence of control signal is 1^1 and 0^{N-1} , which means one bit of 1 and $(N - 1)$ bits of 0. The loading operation of the values occurs when the ctl is in logic 1.

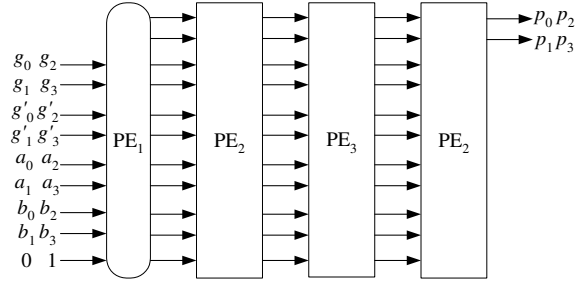


Fig. 3.4. The digit-serial systolic AB^2 multiplier in $GF(2^4)$.

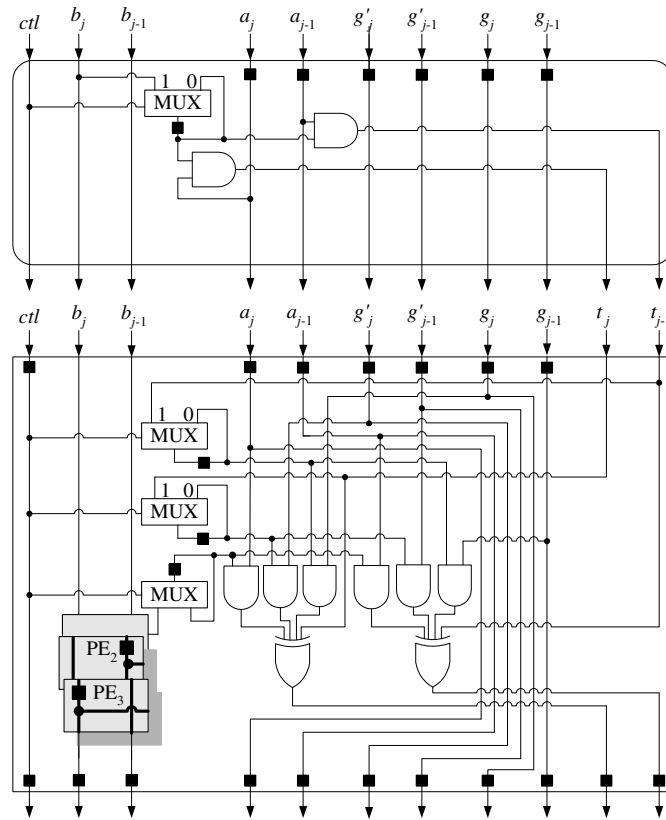


Fig. 3.5. The PE_i structures in Fig. 4.

3.2 Digit-Serial Systolic Divider

Assume that A , B , and D are three elements in $GF(2^m)$, division is performed using multiplication and multiplicative inverse, that is, $D = A/B = AB^{-1}$. Inverse can be regarded as a special case of exponentiation because

$$B^{-1} = B^{2^m-2} = (B(B(B \dots B(B(B)^2) \dots)^2)^2)^2$$

Therefore, division can be computed as following algorithm:

Division Algorithm

Input: A, B

Output: $D = A/B = AB^{-1}$

Initial: $D = B$

1. **for** $i = 1$ **to** $m - 2$ **do**
2. $D = BD^2$;
3. $D = AD^2$;

Here the result $D = A/B$ and the AB^2 multiplication can be used to compute step 2 and 3 operations. When $A = 1$, the algorithm realizes the inversion operation B^{-1} .

The above division algorithm can be implemented using digit-serial systolic AB^2 multiplier of Fig. 4, as shown in Fig. 6. This divider consists of $(m-1)$ AB^2 multipliers for $GF(2^m)$ and some delay elements, where $m = 4$ and $L = 2$.

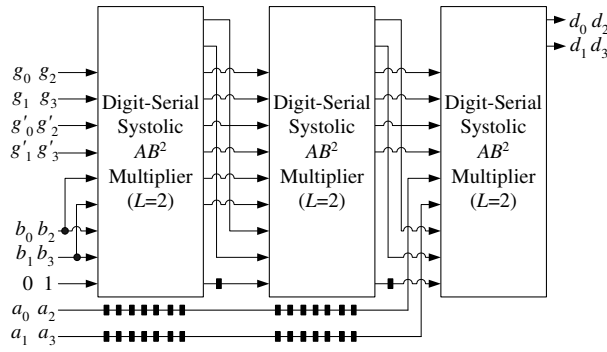


Fig. 3.6. The digit-serial systolic divider in $GF(2^m)$, where $m = 4$ and $L = 2$.

3.3 Analysis

The proposed systolic arrays were described in VHDL with ALTERA MAX PLUS-II tool, and then were simulated using FLEX 10k devices of the ALTERA family for its computation time and correctness.

Comparisons with the characteristics of the systolic architectures described by Wang et al. [6] and Lee et al. [7] in $GF(2^m)$ are listed in Table 1. In reality, the architecture

of [6] has an I/O format with a bit-parallel-input bit-parallel-output. Whereas, the proposed systolic array and the architecture of [7] have an I/O format with a digit-serial-input digit-serial-output.

Table 1. The comparison of three systolic arrays for AB^2 multiplication in $GF(2^m)$

Items	Wang et al. [6]	Lee et al. [7]	Proposed (Fig. 4)
I/O format	Bit-parallel	Digit-serial	Digit-serial
Num. of input pin	$7m$	$5L + 1$	$4L + 1$
Area complexity			
2-input AND	$3m^2$	$3mL$	$(3m - 2)L$
4-input XOR	m^2	mL	$(m - 1)L$
1-bit Latch	$8.5m^2$	$4mL + 8m + N$	$9mL + 4m - 6L - 3$
2-to-1 MUX	0	$3m$	$3m - 2$
Latency	$2m + m/2$	$(L + 2)N$	$2m + N - 1$
Critical path	$T_{AND2} + T_{XOR4} + T_L$	$L(T_{AND2} + T_{XOR4})$ $+ T_{MUX2} + T_L$	$T_{AND2} + T_{XOR4} + T_L$
Control signals	0	1	1

In order to compare the performance of the proposed systolic array with existing architectures, the following assumptions in [10] are made: 1) 4-input gate was constructed using three 2-input XOR gates. 2) $T_{XOR2} = 4.2$, $A_{XOR2} = 14$, $T_{AND2} = 2.4$, $A_{AND2} = 6$, $T_{MUX2} = 3.8$, $A_{MUX2} = 14$, $T_L = 1.4$, $A_L = 8$, where T_{GATE2} and A_{GATE2} are the time and area requirements of a 2-input gate, T_L and A_L are the delay and area of 1-bit latch and, respectively. It show the cost of each gate in terms of the number of transistors it would require when constructed with CMOS technology and the normalized delay(nanosecond; ns) of signal propagation through that particular gate. The area (A), the computation time (T), and the area-time (AT) complexity of the proposed systolic array for $GF(2^m)$ are as follows:

$$A = (132mL - 102L + 74m - 52), T = 12.2(2m + N - 1)$$

$$AT = (132mL - 102L + 74m - 52)(24.4m + 12.2N - 12.2). \quad (3.12)$$

On the other hand, the A , T , and AT complexity of the systolic arrays of references [6] and [7] are as follows:

$$A = (18 + 42 + 68)m^2 = 128m^2, T = 12.2(2m + m/2) = 30.5m$$

$$AT = (128m^2)(30.5m) = 3904m^3 \quad (3.13)$$

$$A = (92L^2 + 106L + 8)N, T = (10.8L + 5.2)(L + 2)N$$

$$AT = (993.6m^2L^2 + 3610.4m^2L + 3884m^2 + 1316.8mN + 83.2N^2). \quad (3.14)$$

For example, when $m = 160$ and $L = 2$, the AT complexity of the proposed digit-serial systolic multiplier can be reduced approximately by 98.35% and 34.56% than those of Wang et al. [6] and Lee et al. [7], respectively.

Table 2 gives some comparisons of the proposed digit-serial systolic divider with the related systolic dividers described in [5] and [6]. As shown in Table 2, the proposed systolic divider has less AT complexity than the existing systolic arrays for division in $GF(2^m)$, although it has one control signal.

Table 2. The comparison of three systolic arrays for division in $GF(2^m)$

Items	Wei [5]	Wang et al. [6]	Proposed (Fig. 6)
I/O format	Bit-parallel	Bit-parallel	Digit-serial
Data flow	Bi-directional	Unidirectional	Unidirectional
Area complexity			
2-input AND	$3m^3 - 3m^2$	$3m^3 - 3m^2$	$(3m^2 - 5m + 2)L$
2-input XOR	$m^3 - m^2$	0	0
3-input XOR	$m^3 - m^2$	0	0
4-input XOR	0	$m^3 - m^2$	$(m^2 - 2m + 1)L$
2-to-1 MUX	0	0	$3m(m - 1)$
1-bit Latch	$16m^3 - 20m^2$	$10.5m^3 - 6m^2 - 3.5m$	$(11m^2 - 20m + 8)L$ $+ 4m^2 - 6m + 1$
Latency	$2m^2 - m$	$2m^2 - 1.5m$	$2m^2 - 3m + N + 1$
Critical path	$T_{AND2} + T_{XOR3} + T_L$	$T_{AND2} + T_{XOR4} + T_L$	$T_{AND2} + T_{XOR4} + T_L$
AT complexity	$O(m^5)$	$O(m^5)$	$O(m^4)$
Control signal	0	0	1

4 Conclusion

This paper proposed efficient digit-serial systolic arrays for AB^2 multiplication and division in $GF(2^m)$ with the standard basis representation. The proposed systolic arrays have a significant improvement in reducing the AT complexity compared with previous architecture, although they have one control signal. They have regularity, modularity, and unidirectional data flow, and thus are well suited to VLSI implementation. Accordingly, the proposed arrays could be used in cryptographic hardware and IC cards.

Acknowledgments

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment).

References

1. C. L. Wang and J. L. Lin: Systolic array implementation of multipliers for finite field $GF(2^m)$. IEEE Trans. Circuits System **38** (1991) 796–800
2. C. L. Wang and J. L. Lin: A systolic architecture for inverses and divisions in $GF(2^m)$. IEEE Trans. Computer **42** (1993) 1141–1146
3. S. W. Wei: A systolic power-sum for $GF(2^m)$. IEEE Trans. Computer **43** (1994) 226–229
4. J. H. Guo and C. L. Wang: Bit-serial Systolic Array Implementation of Euclid's Algorithm for Inversion and Division in $GF(2^m)$. Proc. 1997 Int. Symp. VLSI Technology Systems and Applications (1997) 113–117
5. S. W. Wei: VLSI architectures for computing exponentiation, multiplicative inverses, and divisions in $GF(2^m)$. Proc. 1995 IEEE Int. Symp. Circuits and System (1995) 4.203–4.206
6. C. L. Wang and J. H. Guo: New Systolic Array for $C + AB^2$, Inversion and Division in $GF(2^m)$," IEEE Trans. Computer **49** (2000) 1120–1125
7. K. J. Lee, K. W. Kim, and K. Y. Yoo: Digit-serial systolic power-sum array in $GF(2^m)$. IEEE Proc. ICII 2001 - Beijing, 2001 International Conferences on, **5** (2001) 134–139
8. S. Y. Kung: VLSI array processors. Prentice Hall NJ (1988)
9. K. Y. Yoo: A Systolic Array Design Methodology for Sequential Loop Algorithms. Ph.D. Thesis Rensselaer Polytechnic Institute (1992)
10. D. D. Gajski: Principles of Digital Design. Prentice Hall Upper Saddle River NJ (1997)

Parallel Co-Processor for Ultra-fast Line Drawing

Pere Marès, Antonio B. Martínez and Joan Aranda

Departament of Automatic Control and Computer Engineering,
Universitat Politècnica de Catalunya (UPC).

Pau Gargallo, 5. 08028. Barcelona. Spain. {pere.mares,antonio.b.martinez,joan.aranda}@upc.es

Abstract. A VLSI implementation of a parallel co-processor for ultra-fast line drawing is presented. The specific designed architecture computes a parallel line drawing algorithm optimized for its hardware implementation. It allows writing simultaneously in just one clock cycle all the pixels that approximate a given straight segment. This co-processor also provides read/write random accesses and raster outputs in order to display the data serially in a graphic device. A 256x256 eight-bit pixel processor array has been implemented using 0.35 μ m standard cells. An exhaustive test and simulation results upon this design have demonstrated that a rate of 50M segments per second can be drawn, independently of their length and orientation.

1 Introduction

The problem of line drawing is how to select those pixels that provide the best approximation to the segment. Depending on the type of drawing, the best approximation is not always that which has the most rectilinear appearance. When drawing polygonal shapes the endpoints of the segments are more important to form closed figures. In other cases the desired property is constant density, regardless of the length and the angle. In interactive applications, speed is basic for drawing segments.

Some problems have been amply studied such as the closeness pixel measurements for a subjective assumption, the single and multiple-pixel thickness or the differences between edges and lines [1]. Today we have at our disposal a series of algorithms known generically as line-drawing algorithms. The standard use the incremental techniques [2], which involve iterative computation, obtaining the coordinates of each point that approximates the straight-line segment from beginning to end.

Two of the most widely known of these algorithms are the Digital Differential Analyzer (DDA) [2] and the Bresenham algorithm [3]. The former works incrementing simultaneously x and y coordinates by small steps proportionally to the slope of the segment and it generates addressable points. The latter increments at each iteration the coordinate of most variation of the line and decides, on the basis of the accumulated error between the pixel and the real segment, whether to also increment the coordinate of least variation. The computational cost is therefore proportional to the number of pixels in each segment. The problem appears in interactive applications where the number of segments to be dealt with is very large, as is the case in most related processes in graphics and computer vision. Some authors have proposed speeding up this process by N-step Incremental Straight-line Algorithms that select more than one pixel

per iteration [4],[5]. Other authors have used massively parallel machines in two ways: Distributing different vectors on different processors (*object approach*) that has the problem that the processors must share the memory [6] or using N processors to draw simultaneously M points of the same vector (*image approach*) [7][8]. In this last case new parallel algorithms are required and each processor selects its points in sequential mode. More recently it has been suggested to use the length and direction statistical distribution the lines to speed up the process [9].

From another point of view some authors have proposed to implement logic enhanced memories to accelerate graphics rasterization. Efficient special purpose graphics system architecture known as “Pixel-planes” has been proposed [10]. This architecture processes simultaneously, for all pixel, linear expressions of the form

$$F(x,y)=Ax+By+C$$

in a binary tree multiplier, and each pixel consists of an array of memory elements and a small processor that performs sequentially local operations to the pixel. A great variety of graphics algorithms can be implemented for this architecture describing pixel operations in terms of linear expressions [11]. This method yields a considerable improvement of frame buffer bandwidth. [12].

Obviously this architecture is able to draw straight lines. Nevertheless, pixel identification for the A, B, C coefficients is sequential because the pixel local processor is one bit processor.

In an attempt to provide an improvement in line drawing speed, we propose to accelerate the process of obtaining the pixels to be drawn and to shorten the display memory write time. In order to achieve these goals we aim to:

1. Design, test and simulate an implementable ASIC co-processor capable of calculating and memorizing in parallel the pixels that approximate the straight-line segments.
2. Straight-line Rasterization.

2 Line-drawing modeling

Consider a segment with positive slope $0 \leq m \leq 1$, where the coordinate of most variation is the x coordinate, as shown in Figure 1.

It can be written as :

$$\Delta y = m * (x_i - x_1) \quad (2.1)$$

Where m is the slope, $(x_i - x_1)$ and Δy are the increments, from the starting point to any point (x_i, y_j) in the segment on the x and y coordinates respectively

In the discrete plane, in order to find the pixels that approximate the segment we can use equation (1), modified as follows:

$$\Delta y_{seg} = Round(m * (x_i - x_1)) \quad (2.2)$$

Where now $(x_i - x_1)$ is the increment in the axis of x coordinates with x_i, x_1 integers. Consequently, Δy_{seg} is the increment in the axis of y coordinates from the

starting point to any point (x_i, y_j) in the segment. Note that this increment Δy_{seg} can estimate the real increment Δy with an absolute maximum error of half-pixel as shown in Fig.1.

On an other hand for any column of pixels in the discrete plane only one pixel (x_i, y_j) satisfies the expression:

$$\Delta y_{seg} = y_j - y_1 \quad (2.3)$$

If the slope is negative and falls within $-1 \leq m \leq 0$, the coordinate of most variation continues to be the x coordinate and the expressions remains unchanged.

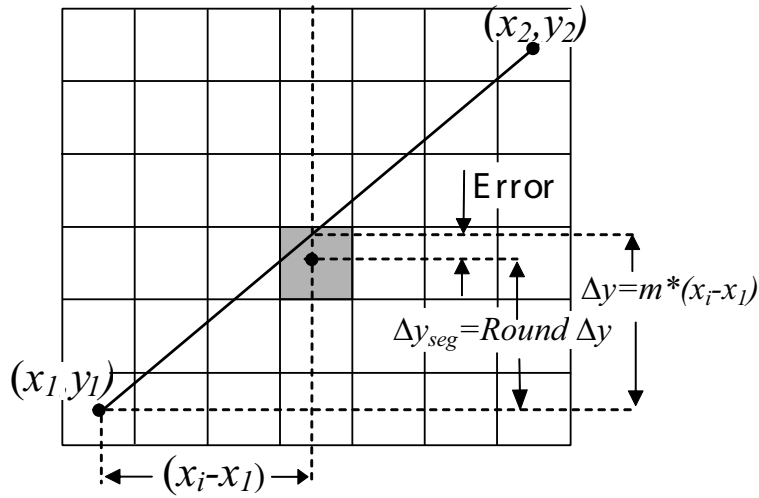


Fig. 2.1. Discretization error

In contrast, for slopes of $-\infty \leq m \leq -1$ and $1 \leq m \leq \infty$, the coordinate of most variation becomes the y coordinate. This introduces a symmetrical case:

$$\Delta x_{seg} = \text{Round}((y_j - y_1)/m) \quad (2.4)$$

Where Δx_{seg} is the increment in the axis of x coordinates from the starting point to any point (x_i, y_j) in the segment and for any row of pixels in the discrete plane only one pixel (x_i, y_j) satisfies the expression:

$$\Delta x_{seg} = x_i - x_1 \quad (2.5)$$

3 A parallel line-drawing algorithm

This expression (3) and (5) enables a test to be implemented that determines whether any pixel (x_i, y_j) belongs to the set of pixels that approximate the segment. Clearly, this test is very efficient if we dispose multiprocessor architecture with pixel processors. Below we present the above idea formalized into an algorithm in pseudocode that includes all the cases that may occur in line drawing.

The independence between the pixel tasks allows the simultaneous activation of all pixels that approximate the segment.

```

Proc Non_inc_ld ( $x_1, y_1, x_2, y_2$ ; integer)
var
 $C_{inc}[N], R_{inc}[M], i, j$  : integer;
 $m$ : double;
begin
   $m := (y_2 - y_1)/(x_2 - x_1)$ ; // Slope
  if  $(-1 \leq m \leq 1)$  then begin
    for all Column  $i$  in  $[0 \dots N]$  // Column tasks
       $C_{inc}[i] = \text{Round}(m * (i - x_1))$ ; (6)
    for all Row  $j$  in  $[0 \dots M]$  // Row tasks
       $R_{inc}[j] = j - y_1$ ; (7)
    end
  else begin
    for all Column  $i$  in  $[0 \dots N]$  // Column tasks
       $C_{inc}[i] = i - x_1$ ; (8)
    for all Row  $j$  in  $[0 \dots M]$  // Row tasks
       $R_{inc}[j] = \text{Round}((j - y_1)/m)$ ; (9)
    end
  end
  Wait tasks(); // Waiting end of Column/Row tasks
  for all Pixels( $i, j$ ) in  $[0 \dots N][0 \dots M]$  // Pixels task
    if  $C_{inc}[i] = R_{inc}[j]$  then begin
      SetPixel ( $i, j$ );
    end
  end
end

```

4 Parallel architecture design

In the above section we saw that if we have the increments of row R_{inc} and column C_{inc} , we can determine whether any pixel belongs to the set of pixels that approximate a segment, by means of a simple comparison. We can therefore design a matrix of $N \times M$ pixel processor cells, which will use these data to simultaneously determine this set of

pixels. This procedure will only be justifiable if the architecture of each processor cell is small enough to be replicated $N \times M$ times, and this depends on the resolution of the image.

4.1 Row and column calculation units

In order to implement the hardware design of the algorithm we need the units that compute the expressions (6),(7),(8),(9) that appear in the presented parallel line-drawing algorithm. The evaluation of the expressions (7) and (8) can be obtained with a simple subtractor and the optimum way to obtaining (6) and (9) is by means of a lookup table (LUT). It can be noted that we can take advantage of the fact that the row j or column i are known values for each unit (Fig. 2b). Also note that, according to the algorithm, we have two symmetrical cases depending on the slope of the segment to be drawn. The calculation units can select the appropriate operations by means of a 2:1 multiplexer that discriminates between these two cases. Nevertheless we have to keep in mind that the capacity of the LUT's may compromise the design, since they may be replicated $N + M$ times.

The first problem is that the number of slopes m in the discrete plane is very large $O(N^2)$ [13]. In order to make a feasible implementation we discretize the possible segment angle α , where $\alpha = \text{tg}^{-1}m$. Based on a perceptual assumption we limit the angle error resolution to $\Delta\alpha = 0.1^\circ$, that means we need 11 bits to code α .

Another problem is the large number of input bits in the LUT's. We can obtain a input reduction by using the following equations, where er is the approximation error.

$$C_{inc}[i] = \text{Round}(m * (i - x_1)) = \text{Round}(\text{tg} \alpha * i) + \text{Round}(\text{tg} \alpha * x_1) + er$$

$$R_{inc}[j] = \text{Round}((j - y_1)/\text{tg} \alpha) = \text{Round}(j/\text{tg} \alpha) + \text{Round}(y_1/\text{tg} \alpha) + er$$

Note that $er_{max} = 1$ or $er_{max} = -1$ according to the definition of *Round* function. This means that the algorithm will decide to increment the coordinate of least variation one pixel before or after it would have done using the calculation without the above decomposition.

As the error is not accumulative this can only change the approximation of certain specific lines, and both types of calculation yield valid approximations because they respect the slope.

Thus, it can be observed that the products $\text{tg} \alpha * x_1$ and $y_1/\text{tg} \alpha$ are independent of the column and row, and consequently could be calculated for all the columns and row in a common units, out of the calculation units.

Thus, Fig. 2(a) shows the common calculation unit for all the columns (CC), and Fig. 2(b) the dedicated calculation units of each column (DC).

It can be appreciated that the LUT of the DCi unit has been simplified by reducing the inputs to only the slope α . The discretization of α to 11 bits makes feasible the design and implementation of the processor we are proposing. The same calculations and reasoning can be applied to obtain the common calculation unit for all the rows (CR) and the dedicated calculation units of each row (DRi).

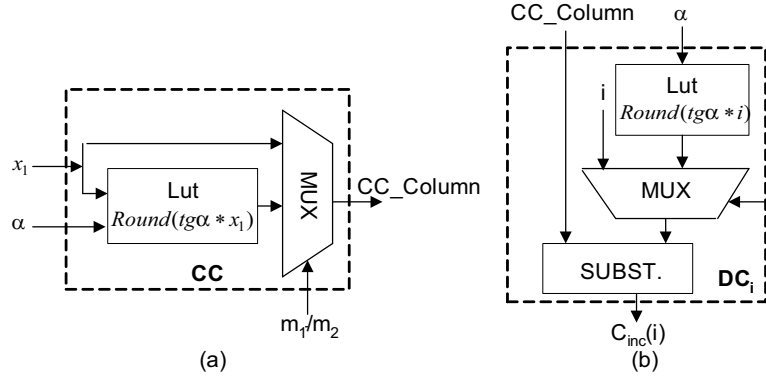


Fig. 4.2. Decomposed calculation unit (a) Common column calculation.(CC) (b) Dedicated Column calculation.(DC)

4.2 Pixel -processor design

As the figure 3 shows, the basic cell consists of a comparator that verifies equality between the row and column increments and a register for storing the color word. Such simple pixel processing makes a hardware implementation feasible.

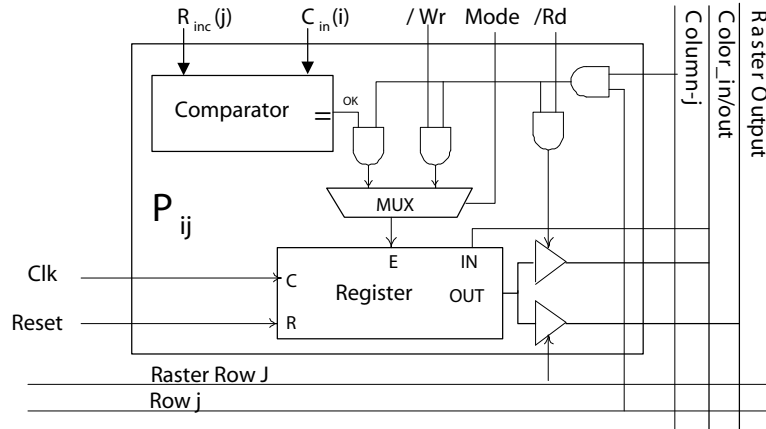


Fig. 4.3. Pixel cell with the access logic

The architecture of the pixel processor we have considered up to now performs writing operations automatically as a result of the processing of straight-line segments, on reception of the signal *ok* from the pixel cell comparator. To avoid limiting the performance of this design we have also provided it with the possibility of read or write

accesses in random positions, thus allowing values resulting from other graphic processes to be written or read. We have also given it an output that is oriented toward dumping over a raster-type device, requiring a dual port design in which the display output signals of a selected row cells are serialized in a shift register synchronized by the video signals (Fig. 4).

Thus, in order to differentiate between the read/write random access operations and the segment processing operations we have incorporated the signal *Mode* in addition to the usual signals *Rd* and *Wr*.

Internal buses. In order to distribute the results of the calculation units we have to provide the design with N horizontal buses of width $\log_2(N)$ and M vertical buses of width $\log_2(M)$ with an additional bit of sign

The dual port design of the pixel cell imposes the need to have two types of tri-state data buses: input/output and display output. These two types of buses, of width equal to the size of the color word, descend each column. The difference between them lies in the fact that the input/output buses are bi-directional and come together to form a single I/O bus (*color_in/out*), whereas the raster bus is exclusively for output and serves to load the shift register that serializes the data toward the output (*raster_out*).

Note that the number of lines crossing the matrix vertically and horizontally is large and directly proportional to the resolution of the image ($N \times M$) and the color depth. The limitations of the VLSI technologies will determine these parameters. In the Section on VLSI design, we will discuss possible solutions to these problems.

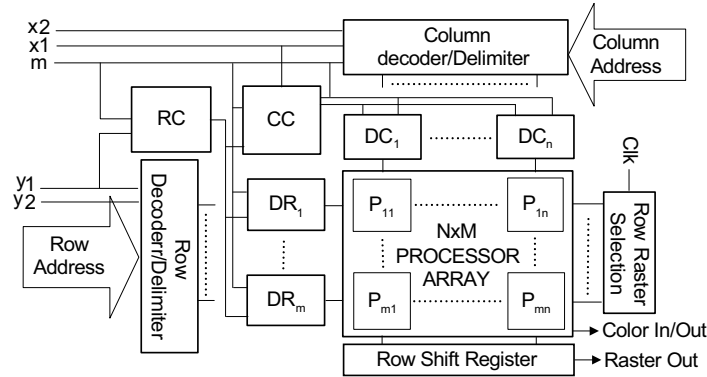


Fig. 4.4. Overall architecture: Pixel selection, calculation units, length delimiters, raster output

Pixel selection. To allow access to the matrix we have introduced the row and column decoders (Fig.4) that provide the lines *Row j*, *Column i* as shown in Fig. 3. The selected cell will connect the input and output of its register to the tri-state input/output bus (*color_in/out*, Fig.3), thus making it possible to write or read the color word, according to whether we perform a random write or read operation.

In order to be able to dump the contents into a raster device, it was necessary to design a sequential logic providing new row selection lines. This logic also uses the video synchronization signals, as the shift register. Fig. 3 shows a block diagram with these new elements.

4.3 Length of segment

With regard to the processing of straight-line segments, we still fall short of the initial specifications, since the design as it now stands fails to delimit the segment defined by its endpoints; rather, it draws the line that passes through them. In order to delimit the length of the segment, we introduce some new units which we will refer to as delimiters. For ease of design we have prepared one delimiter unit for rows and another for columns. Given the beginning and end coordinates of the segment to be drawn, they implement, for each row or column, a combinatorial function which will disable those pixel cells that do not feature among the columns and rows of the endpoints, thus preventing their registers from loading the color word.

As random access operations and straight-line segment processing are exclusive in time, we will use the row and column selection lines for this function. Consequently, the delimiting and decoding units will share these lines multiplied in time as shown in Fig 4.

5 Vlsi design

As we said before the objective of this work is to evaluate a new processor architecture to be implemented in a custom VLSI that would be able to write in parallel all the pixels that approximate a given straight segment. This will be done by, first designing the architecture, and then simulating and testing, all the signals and technologic constraints, using the Integrate Circuits development environment, which was donated by Cadence to the Europractice Group.

If a single ASIC with 24 bit true color was not feasible because our technological resources would not allow this architecture implementation, this need not be an impediment since the proposed design is color depth scalable. This means that three ASIC will be needed to form true color if the register cell is eight-bit.

The VLSI design of our processor used a standard cell of $0.35\ \mu\text{m}$ with three metal layers. This allowed us to obtain an evaluation of processor performances in relation to the drawing speed rate, the area required, the random accesses transfer rate and the serial output data.

5.1 Experimental results

For this experimental design, we sought to estimate the size of an implementable processor. We opted for an 8-bit color word as a compromise between the maximum color word size and the scalability required to achieve it. (three devices to achieve true color).

We have used a bottom-up design that improves the placement and the routing processes in order to obtain optimized layout. Technological limitations have made it necessary to place the LUT's in independent blocks.

In order to improve the resolution of the images that the architecture can process, we may increase the pixel matrix dimensions. Thus, within the same technology, the unique solution is to reduce the size of the color word, which involves an increase in the number of devices when scaling. Alternative solutions would be to migrate to a technology with more integration scale. The large replication of the pixel cell (quadratic) guarantees a severe area reduction if we obtain a full custom design of this unit.

On the other hand the simulation has showed that the “critical path time” of the processor design with this technology is about 20ns and yields a rate of 50M segment/second. This means, in the case that all the segments would be the longest, would have to write up to 12G pixel/second

6 Conclusions

This work has studied the problem of the line drawing speed rate necessary in graphic interactive applications. To speed up this process we have developed a parallel line drawing algorithm that easily allows a hardware implementation.

Based on this algorithm, we have designed a parallel line drawing processor that allows us to process and write the pixels value of the current segment in a single cycle, independently of their length and orientation. Thus, the speed drawing rate depends only on the timing constraints imposed by the VLSI design and technology.

Taking into account the large replication of the pixel cell, a standard cell implementation is not a good solution because it cannot provide an optimum use of area. Consequently, the timing performance we can obtain still falls short of reaching an optimized implementation. A real useful implementation can only be obtained using full custom design techniques that will provide improved performances.

7 Acknowledgments

We would like to thank the graduated student David Tuset for his valuable contribution to this work, mainly in the VLSI design and in the entire test to evaluate the processor presented in this paper.

References

1. Bresenham, J.E. “Pixel-Processing Fundamentals” IEEE Computer graphics and Applications. Vol.16 N° 1, pp 74-82, January 1996
2. W. M. Newman, R .F. Sproull “principles of interactive computer graphics “ Mc GRAW-HILL 1988
3. Bresenham, J.E. “Algorithm for computer Control of a Digital Plotter “ IBM System Journal Vol 4,N°,1 pp 25-30 1965
4. Gill,G.W.;. “N-step incremental straight-line algorithms” IEEE Computer Graphics and Applications , Volume: 14 , Issue: 3 , May 1994 pp.:66 – 72
5. Boyer, V.; Bourdin, J.-J.; “Auto-adaptive step straight-line algorithm” IEEE Computer Graphics and Applications, Volume: 20 , Issue: 5 , Sept.-Oct. 2000 , pp.67 – 69
6. Letellier, Laurent & alt. “High performance graphics on a SIMD linear processor array” IEEE International Symposium on Circuits and Systems Vol. 3, pp 1901-1904, 1993

7. William E. Wright "Parallelization of Bresenham's Line and Circle Algorithm" IEEE Computer graphics and Applications. Vol.10, pp 60-67, September 1990
8. Alex T. Pang "Line Drawing Algorithm for Parallel Machines" IEEE Computer graphics and Applications. Vol.10, pp 54-59, September 1990
9. Chen, J.X.; Xusheng Wang; Bresenham, J.E "The analysis and statistics of line distribution" IEEE Computer Graphics and Applications, Volume: 22 , Issue: 6 , Nov.-Dec. 2002 pp.100 - 107
10. Fuchs, and J.Poulton "Pixel-Planes: A VLSI-Oriented Design for a Raster Graphics Engine" VLSI design Vol 2, N° 3 pp 20-28 1981
11. Fuchs, and & alt. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes" SIGGRAPH(85) pp. 111-120 . 1985
12. Poulton J. and & alt. "Braking the Frame Bottleneck with Login enhanced Memories" IEEE Computer graphics & Applications Vol 12, N° 6 pp 65-74. 1992
13. J Koplowitz, ,M. Lindenbaum and A. Bruckstein "The number of digital straight lines on an $N \times N$ grid" IEEE Transactions on Information Theory, Volume: 36 , Issue: 1, Jan. 1990
pp.192 - 197

The Design Patterns of Performance-Decision Factors in High-Speed NIDS

Jongwoon Park¹, Keewan Hong¹, Kiyoong Hong¹,
Dongkyoo Kim², Bongnam No³

¹ Information Security Technology Institute, Secuve Inc.,
Ildong Bldg 3F, 60 Yangjae-Dong, Seocho-Gu, Seoul, Korea
{hizcool, kwHong, kyhong22}@secuve.com

² Ajou University, dkkim@adang.ajou.ac.kr

³ Chonnam University, bongnam@chonnam.ac.kr

Abstract. The globalization of IT Infrastructure has brought about a great increase in the volume of network users and data, and also has increased the possibility of internal and external intrusions into important assets of an organization. These kinds of changes have shown the limitations of NIDS, which detects abnormal activities through network packet. The increase in network bandwidth causes load of NIDS packet collection and analysis, which leads to packet loss. This means NIDS is unable to get necessary data to detect abnormal activities. And, the increase and changes of network service continuously reveal new vulnerability, through which the unknown new attacks will be increased by multiple. NIDS which detects abnormal activities by developing detection pattern to each attack reaches its performance limitation of increased processing load due to continuously increasing detection patterns. Therefore, this paper presents design principles considered in developing NIDS to overcome its performance limitations, and proves the effectiveness of these suggested design principles through test models.

1 Overviews

We have deduced three factors that determine NIDS performance based on NIDS reference model developed by ISO/IEC and IETF, and through simulation test. The first factor, event collection mechanism, collects necessary packet to detect abnormal activities, and determines NIDS performance according to network bandwidth, protocol and packet size distribution. The second factor, data handling mechanism, distributes and transfers collected data appropriately with analyzer which judges abnormal activities. The third factor, pattern matching mechanism, compares the detection pattern list of known attacks with collected packets, and NIDS performance depends on the number of detection pattern list and comparing mechanism. On this basis, this paper suggests four design principles - efficient event filtering, high-speed data communication, clone-based data distribution, and multi-pattern matching algorithm - according to three factors above. Finally, the effectiveness of design principles suggested in this paper to improve NIDS's performance are proved by performance test of three models, that is, one process model, role-based model, and clone-based model.

2 NIDS reference models

IETF (Internet Engineering Task Force) and ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) define the components of NIDS and data flow among them as the Fig. 2.1.

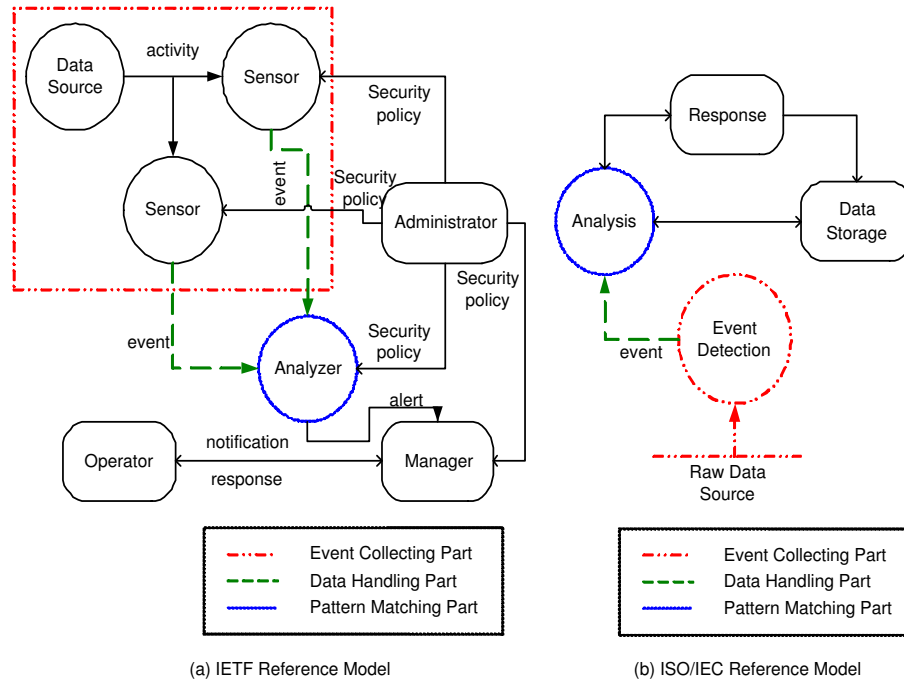


Fig. 2.1. NIDS reference model[2][3]

The above two NIDS Reference Models are consist of the common five components described in the Table. 1 and each component is conceptual design unit for NIDS development.

- Event collecting component: The process that collects data from the data source. The frequency of data collection will vary across network bandwidth, packet size and protocol distribution.
- Data handling component: The process that distributes and transfers data from event collecting component to pattern matching component for high-speed communication.
- Pattern matching component: The process that analyzes the data collected for signs of unauthorized or undesired activity or for events that might be of interest to the NIDS. The frequency of pattern matching will vary across attack signatures.

NIDS component	IETF reference model	ISO/IEC reference model
Event Collecting	Sensor	Event Detection
Data Handling	Event	Event
Pattern Matching	Analyzer	Analysis
Pre/Post Managing	Manager	Response
Event Source	Source	Source

Table 1. NIDS components

- Pre/post managing component: The process that manages the various processes of the NIDS. Management functions typically include configuration, notification, response, and reporting.
- Event source component: The raw information that an NIDS uses to detect unauthorized or undesired activity. Data source of NIDS is raw network packet.

3 NIDS performance-decision factors

The five components induced by NIDS Reference Model are implemented as shown in the Fig. 3.2, and generally classified into Manager, Event Collector, Analyzer, post-Processor, Queues according to their function.

- Manager: provides the manager with information like set-up configuration and etc. for successful NIDS performance. Generally, this kind of management event is very rare and implemented in asynchronous way, hardly affecting NIDS performance.
- Event collector: collects packets from network device to recognize abnormal events and generally realized in independent process or thread form. It directly affects NIDS capacity according to occurrence frequency of collected packets which is determined by network bandwidth, protocol distribution and packet size distribution. So, effective designing of Event Collecting Mechanism must be considered for NIDS performance.
- Analyzer: judges the abnormal events by comparing packet information collected from network device with NIDS-held attack patterns and generally implemented in the multiple number of process or thread form for parallel treatment. This directly affects NIDS capacity with the proportion of occurrence frequency of collected packet and the number of NIDS-held attack patterns. The endless increase of attack patterns lowers NIDS capacity. So, effective designing of Pattern Matching Mechanism must be considered for NIDS performance.
- Post-processor: deals with counteract to abnormal events and audit log. This kind of event is very rare and realized in asynchronous way, hardly affecting its performance.
- Queues: manages data flow between Event Collector and Analyzer and generally implemented by IPC mechanism. This directly affects NIDS capacity with the proportion of occurrence frequency of transferred packets, transferring mechanism and

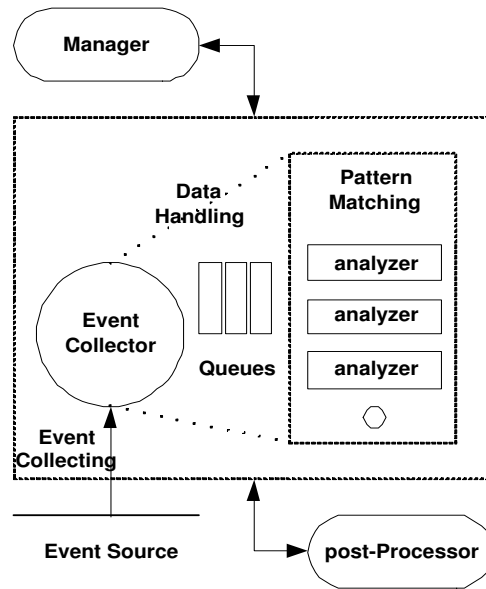


Fig. 3.2. NIDS high-level implementation model

transferred data size. So, effective Data Handling Mechanism must be considered for NIDS performance.

Among these five NIDS components, the factors affecting the performance are as the Table. 2.

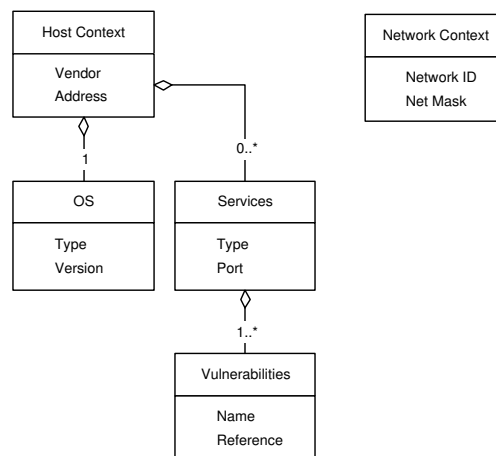
4 Design principles of high-speed NIDS

In this chapter, we suggest five principles for efficient designing of NIDS performance-defining components related with event collecting mechanism, data handling Mechanism, and pattern matching mechanism induced in chapter 3.

4.1 Need-to-Know Target Profile

NIDS must collect only necessary packet to judge abnormal activities. For this, network packet must be filtered by efficient filtering mechanism considering target range, host and network context. Also, the performance of filtering mechanism is decided by implementation layer and semantic characteristic of filtering rules. Filtering rules must not affect the result of NIDS attack detection and must be designed in such way that it can minimize the number of NIDS-collected events. For this end, this paper suggests two-levelled filtering mechanism using target profile as in the Fig. 4.3.

Factor	Causes
Event Collecting Mechanism	Network bandwidth Protocol distribution Packet size distribution
Data Handling Mechanism	Communication method Transferred data size
Pattern Matching Mechanism	Increased attack pattern Pattern matching algorithm

Table 2. NIDS components**Fig. 4.3.** Target Profile structures

Target profile defines Host context class to distinguish specific host, and Network context class to distinguish Network in NIDS detection target. Host context class consists of vendor and address attributes, and aggregated classes of OS, Services. Address attribute is used as filtering condition for NIDS to select a specific system of event collection and form the one-level filtering layer, generally being implemented in the lower level (hardware or operating system's kernel). OS class consists of type and version attributes. The two attributes are used as filtering condition of distinguishing attack-performing environments which bring about abnormal events and generally form the two-level filtering layer, being implemented in the upper level (application). Services class consists of type, port attributes and aggregated class of Vulnerabilities. The two attributes are used as the filtering condition of distinguishing attack target which bring about abnormal events, and generally form the two-level filtering layer, being implemented in the upper level (application). Vulnerabilities class consists of name and reference attributes. These two attributes are used as filtering condition for distinguishing the status of attack target which causes abnormal events and generally form the two-level filtering layer, being implemented in the upper level (application). Network context class consists of network ID and network mask attributes. The two attributes are used as filtering condition for deciding network range for NIDS to collect events, and generally form one-level filtering layer, being implemented in the lower level (hardware, or operating system's kernel).

4.2 High-speed data transmission

NIDS must make use of efficient communication mechanism to transmit reduction data of collected packet to analyzer. For this, memory communication which minimizes data share shows the most effective performance. To transmit generally collected events to analyzers, we suggest memory communication technique in NIDS parallel structure. As shown in the Fig. 4.4, memory communication technique is divided into Pool Memory Queue Model (event transmission by shared memory pool (a)) and Individual memory queue model (b). The speed of NIDS event collection and abnormal event detection determines the performance of two models. In (a) model, collected events are operated in direct push to memory pool but, must be loaded in the critical region till the analyzer reads event information. On the contrary, in (b) model, the collected events simply go through queue select and need not to be loaded in the critical region when the analyzer reads event information, thus guaranteeing more successful performance.

4.3 Clone-based data distribution

NIDS must not rely on specific features in distributing collected packets. If so, analyzer will operate according to specific role-base, which causes bottle-neck. To prevent this, this paper suggests clone-base structure. This will enable load balancing, distribution to multiple analyzers without switching [4]. Most NIDS implementations have parallel distribution model as in the Fig. 4.5-(a) model to detect abnormal events. However, each plays a designated role according to types of detection patterns which show abnormal events. For this reason, in the process of analyzing characteristics of the collected data and transmitting them to the appropriate analyzer, stagnation and data loss can arise. On

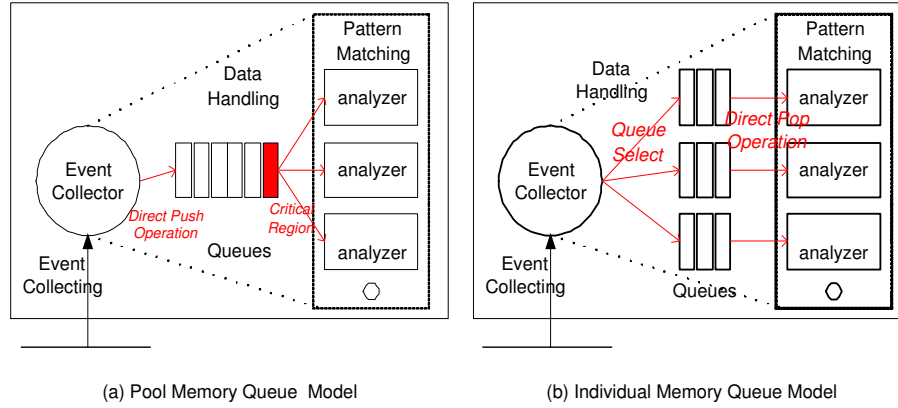


Fig. 4.4. Data transmission models

the contrary, in in the Fig. 4.5-(b) model, the role each analyzer plays are the same, thus enabling loads balancing [5] and getting rid of load problem in analyzing characteristics of collected events. So, in the process of data handling, stagnation and data loss can be minimized.

4.4 Multi-pattern matching algorithm

NIDS adapts efficient pattern matching algorithm to compare detection pattern with collected packet. But these algorithms result in different performance according to data's characteristics. So, the extension of detection pattern according to analysis of data's characteristics enables to adapt multi-pattern matching algorithm [6]. NIDS, to detect abnormal events, signatures the characteristics of attack type and compare them with compressed information of collected events. Generally most NIDS selects pattern matching algorithm which is known as the fastest one. But this misses the fact that signature shows different performance level according to its literary characteristics [7]. This characteristics give validity to the below detection pattern model of the Fig. 4.6 which is suggested to support MPM(Multi-Pattern Matching). In this model, algorithm is added to general detection pattern structure. For this, in developing detection pattern, the process of linking effective algorithm after finding out literary characteristics of signature is necessary.

5 Performance Tests

To prove the effectiveness of designing principles suggested in this paper, we consider three test models explained in the in the Table. 3. Model 1 uses Snort 1.8.0, public software to go through a series of processes from event collection to abnormal event detection. Model 2 modifies Snort 2.0 to get parallel handling structure and data transmission through Memory Queue and Model 3 adds clone-based characteristics to Model [8][9].

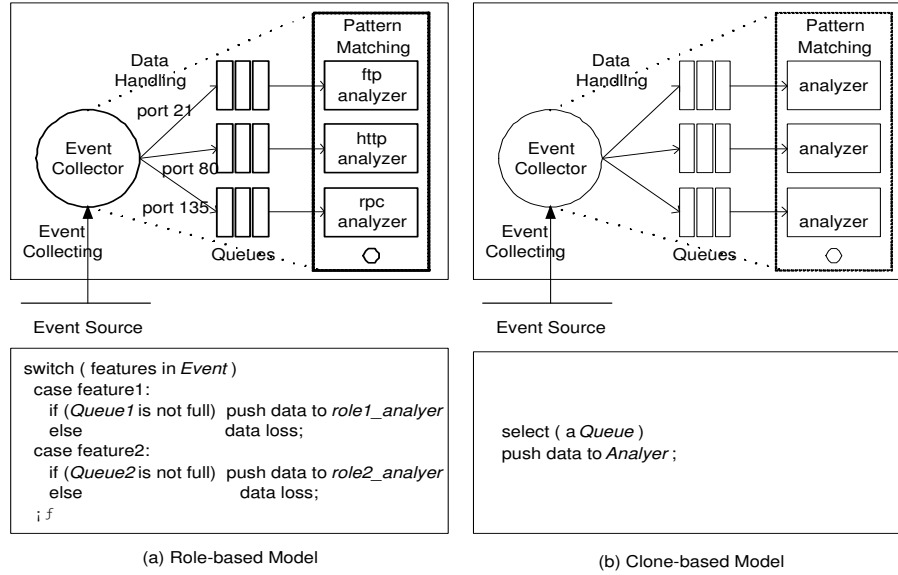


Fig. 4.5. Data distribution models

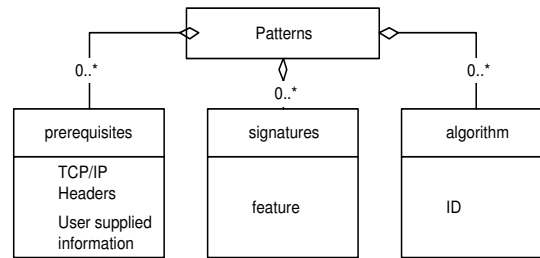


Fig. 4.6. Pattern Classes for supporting MPM(Multi-Pattern Matching)

	Model 1	Model 2	Model 3
# of threads	1	n	n
Data transmission model	N/A	memory queue	memory queue
Data distribution model	N/A	Role-based	Clone-based
Detection model	Misuse	Misuse	Misuse

Table 3. Test models

Test context was 180Mbps of background traffics with the aid of SmartBits-6000, and 590 attack events were arised by Blade's IDS Informer. Model 1, 2, 3 all holds 968 detection patterns. The Table. 4 shows each model's detection rates. We can confirm that model with all NIDS designing principles show the most successful performance.

	Model 1	Model 2	Model 3
1	278	389	442
2	299	399	420
3	281	402	412
4	293	393	439

Table 4. Test results - detection rates

6 Conclusions

With an age of high-speed in information network and incessant advent of new attack patterns like Warhol worm or flash worm, NIDS performance to secure enterprise's assets from abnormal events has come to the fore [10]. This paper induced factors causing NIDS stagnation to improve NIDS performance and presented designing principles for each factor. Each designing principle minimizes loss of data which are basis of detecting abnormal events. Also in chapter 5, we proved the validity of these designing principles with the test result of three test models.

References

1. Rebecca Base and Peter Mell. *Intrusion Detection Systems*. NIST Special Publication 800-31, November 2001.
2. *Intrusion Detection Message Exchange Requirements*. IETF/IDWG Internet-Drafts, October 22, 2002.
3. *IT Intrusion Detection Framework*. ISO/IEC TR 15947, 1998.
4. Neil Desai. *Increasing Performance in High Speed NIDS*. A look at Snort's Internals.
5. LU Sheng, GONG Jian, and RUI Suying. *A Load Balancing Algorithm for High Speed Intrusion Detection*.
6. C.J.Coit, S.Staniford, and J.McAlerney. *Towards Faster Pattern Matching for Intrusion Detection or Exceeding the Speed of Snort*. In Proc. 2nd DARPA Information Survivability Conference and Exposition, June 2001.
7. Christian Charras and Thierry Lecroq. *Handbook of Exact String Matching Algorithms*, King's College LONDON Publication.
8. Snort. *Handbook of Exact String Matching Algorithms*, King's College LONDON Publication. <http://www.snort.org>

9. Bro. *A System for Detecting Network Intruders in Real-time*. <http://www.icir.org/vern/bro-info.html>
10. Patric Wang. *Internet Worm and Vulnerability Trends*. In Proc. 7th CONCERT Workshop, pp213-228, Seoul, Korea, November 2003.

Scalable Race Visualization for Debugging Message-Passing Programs^{*}

Mi-Young Park,¹ So-Hee Park,² Su-Yun Bae,³ and Yong-Kee Jun,¹ ^{**}

¹ Gyeongsang National University, Jinju; {park, jun}@race.gsnu.ac.kr

² Kyungsung University, Busan; heeya@star.ks.ac.kr

³ SureSoft Technologies, Seoul; javaio@lycos.co.kr
South Korea

Abstract. Detecting unaffected race conditions is important to debugging message-passing programs effectively, because such a message race can affect other races to occur or not. The previous techniques to detect efficiently message races do not guarantee that all of the detected races are unaffected. In this paper, we present a new technique that traces affect-relations of the locally-first races to occur in all the monitored processes, and then visualizes the affect-relations with scalable graphs for programmers to discriminate unaffected races effectively.

1 Introduction

In asynchronous message-passing programs, a *message race* [4, 7, 12] occurs in a receive event, if two or more messages are sent over communication channels on which the receive listens and they are simultaneously in transit without guaranteeing the order of their arrivals. Message races should be detected for debugging a large class of message-passing programs [1, 5, 14] effectively, because nondeterministic order of arrivals of the racing messages causes unintended nondeterminism of programs [7, 9–11]. Especially, it is important to detect efficiently *unaffected races* before which no other races causally happened, because such races may make other affected races appear or be hidden.

The only efficient technique [11] to detect unaffected races in its second monitored execution detects their messages by halting at the receive event of the locally-first race to occur in each process. However, if a process halts at the racing receive, the process cannot send messages thereafter to notify other processes of their being affected and then does hide some chains of affect-relations among those races. This previous technique therefore does not guarantee that all of the detected races are unaffected.

To detect only unaffected races, our previous work [13] traces the states of the locally-first race to occur in every process, and then visualizes the affect-relations of all the locally-first races with event graphs to report unaffected races. However, this technique sequentially analyze all trace files of processes for determining just affected

^{*} This work was supported in part by Grant No. R05-2003-000-12345-0 from the Basic Research Program of the Korea Science and Engineering Foundation.

^{**} *Corresponding author.* Also involved in Research Institute of Computer and Information Communication (RICIC), Gyeongsang National University.

messages among all the processes, and visualizes the messages in too complex a fashion for programmers to discriminate affect-relations between every two locally-first races.

In this paper, we present a new visualization technique that traces affect-relations of the locally-first races to occur in all the monitored processes, and then visualizes the affect-relations with scalable graphs. To trace the affect-relations, we improve the second-pass algorithm of the previous technique [11] to complete the second monitored execution. In the monitored execution, we examine if each message is affected or not, and trace the affect-relations of the locally-first races. After the execution, we visualize the affect-relations with scalable graphs for programmers to discriminate unaffected races effectively.

We tested our technique in MPI [14], an industry-standard model of message-passing parallel program, using MPICH implementation [6] on a cluster system. We implemented our tracing algorithm as a C-library using MPI Profiling Interface to make it transparent to user programs, and our visualization algorithm using Java language and the *Soot* API [16].

In the following section, we introduce some notions of message races and then, in Section 3, we explain the problem of the previous techniques to detect unaffected races for debugging message-passing programs. In Section 4, we present our scalable technique to trace and visualize the affect-relations of the locally-first races to occur in all the monitored processes. In Section 5, we support our technique with some experimentation details. Finally, we conclude it with some future work in Section 6.

2 Message Races

We model asynchronous message-passing [1, 5, 14, 15] between processes as occurring over *logical channels* [11], and assume that each send or receive event specifies a set of logical channels over which it operates to send copies of one message or to receive one message from the channels. If more than one channel have a message available, the receive event nondeterministically chooses a channel among them to receive one message. We assume that any message sent over a channel is received by exactly one receive event, and all messages sent during program execution are eventually received at the corresponding receive events. This model with logical channel is general, because most message-passing schemes can be represented.

An execution of message-passing program is represented as a finite set of events and the *happen-before* relation [8] defined over those events. If an event a always occurs before another event b in all executions of the program, it satisfies that a happens before b , denoted $a \rightarrow b$. For example, if there exist two events $\{a, b\}$ executed in the same process, $a \rightarrow b \vee b \rightarrow a$ is satisfied. If there exist a send event s and the corresponding receive event r between a pair of processes, $s \rightarrow r$ is satisfied. This binary relation \rightarrow is defined over its irreflexive transitive closure; if there are three events $\{a, b, c\}$ that satisfy $a \rightarrow b \wedge b \rightarrow c$, it also satisfies $a \rightarrow c$.

Messages may arrive at a process in a nondeterministic order by various causes in the execution environment, such as variations in process scheduling and network latencies. In a large class of programs [1, 5, 14] that are intended to be deterministic, nondeterministic order of message arrivals causes unintended nondeterministic executions of

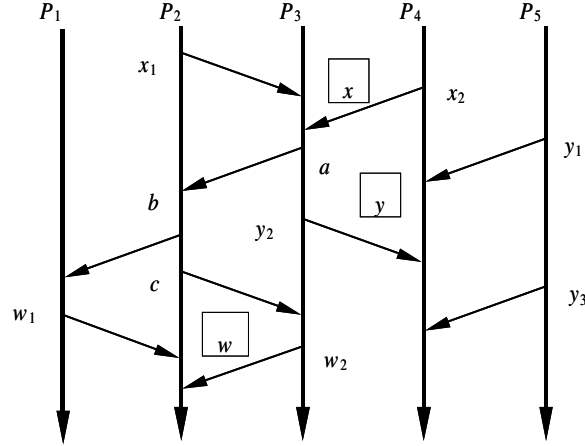


Fig. 2.1. Message Races

a program so that such race conditions of messages should be detected for debugging [7, 9–11]. A *message race* [4, 7, 12] occurs in a receive event, if two or more messages are sent over communication channels on which the receive listens and they are simultaneously in transit without guaranteeing the order of their arrivals. A message race is represented as $\langle r, M \rangle$, where r is a receive event and M is a set of racing messages toward r . Thus, r receives the message delivered first in M , and the send event s which sent a message in M does not satisfy $r \rightarrow s$. We denote a message sent by a send event s as $msg(s)$.

Figure 2.1 shows a partial order of events that occurred during an execution of message-passing program. A vertical arc in the figure represents an event stream executed by a process along with time; and a slanting arc between any two vertexes that are optionally labelled with their identifiers represents a delivery of message between a pair of send and receive operations. For instance, two processes, P_2 and P_4 , send two messages, $msg(x_1)$ and $msg(x_2)$, to P_3 respectively, and two send events, x_1 and x_2 , do not satisfy $x \rightarrow x_1 \wedge x \rightarrow x_2$ where x is a receive event occurred in P_3 . This implies that these two messages race each other toward x . This message race occurred at x in P_3 therefore can be represented as $\langle x, X \rangle$, because it consists of a set of racing messages $X = \{msg(x_1), msg(x_2)\}$ and the first event x to receive one of the racing messages.

Suppose that there exist only two message races $\{\langle m, M \rangle, \langle n, N \rangle\}$ in an execution of a program, and they satisfy $m \rightarrow s \vee m \rightarrow n$ where $msg(s) \in N$. Then $msg(s)$ is an *affected message* by $\langle m, M \rangle$ because $m \rightarrow s$; and $\langle n, N \rangle$ is an *affected race* by $\langle m, M \rangle$. And we say that $\langle m, M \rangle$ is an *unaffected race*, if there does not exist any message $msg(t) \in M$ that satisfies $n \rightarrow t$ and there exists no such $\langle n, N \rangle$ that satisfies $n \rightarrow m$. For example, figure 2.1 shows three races $\{\langle w, W \rangle, \langle x, X \rangle, \langle y, Y \rangle\}$ where $W = \{msg(w_1), msg(w_2)\}$, $X = \{msg(x_1), msg(x_2)\}$, and $Y = \{msg(y_1), msg(y_2), msg(y_3)\}$. The two races $\{\langle w, W \rangle, \langle y, Y \rangle\}$ in the figure are affected by $\langle x, X \rangle$, because $\langle w, W \rangle$

satisfies $x \rightarrow a \wedge a \rightarrow b \wedge b \rightarrow w$ followed by $x \rightarrow w$ and $msg(y_2) \in Y$ satisfies $x \rightarrow y_2$. These affected races may occur depending on the occurrence of $\langle x, X \rangle$, and may disappear when $\langle x, X \rangle$ is eliminated.

A *locally-first race* is the first race to occur in a process. Although a locally-first race is obviously not affected by any other races occurred in the local process, the race is not guaranteed to be unaffected by another race occurred in the other processes. For example, figure 2.1 shows that all of the races appeared in the figure are locally-first races, but two locally-first races $\{\langle w, W \rangle, \langle y, Y \rangle\}$ are affected by another locally-first race $\langle x, X \rangle$ occurred in the other process.

3 Related Work

Previous methods to detect message races can be classified into two classes: one set of techniques [2, 12] *to verify the existence of races*, and the other set of techniques [3, 4, 7, 11] *to detect unaffected races*. The techniques to verify the existence of races detect them at each receive event by determining if the corresponding send event and the previous receive event in the local process are mutually concurrent. However these techniques are not effective for debugging message-passing programs, because they detect only a small set of races that might be affected by other races. The techniques to detect unaffected races check if a race occurs at each receive and then if it is the first races to occur in the process. These techniques are effective to debugging, because they guarantee that the detected races are not affected by any other races occurred in the same process.

With respect to the degree of monitoring parallelism, those techniques to detect unaffected races can be classified into two classes: *One-thread-at-One-time* (OtOt) [3, 4], and *Multi-threads-at-One-time* (MtOt) [7, 11]. The OtOt technique requires the program to be executed repetitively as many as the number of processes, because it detects one locally-first race by monitoring only one process in each execution. However, the MtOt technique detects all of the locally-first races by monitoring all processes in one execution, and is classified into two classes with respect to the number of monitored executions: *one-pass* [7] and *two-pass* [11] techniques. The one-pass technique shows impractical space complexity which is dependent on the number of messages, because it checks all of the previous receive events at each receive event to detect all of the races that are related to every previous receive event. On the other hand, the two-pass technique finds some information in the first execution to detect a locally-first race of each process, and then tries to detect unaffected races by halting each process at every first racing receive in the second execution. This technique is more efficient than the one-pass technique, because it consumes space and time which are independent of the number of messages.

The two-pass technique [11] tries but does not guarantee to detect unaffected races. If a process halts at the racing receive, the process cannot send messages thereafter to notify other processes of their being affected and then does hide some chains of affect-relations among those races. This technique therefore does not guarantee that all of the detected races are unaffected, because other processes which did not receive such affected messages may report their affected races as unaffected erroneously.

For example, consider this two-pass technique for the same execution instances shown in figure 2.1. In the first execution, each process writes some information into a trace file locating its locally-first race at $w \in P_2$, $x \in P_3$, or $y \in P_4$. In the second execution, it tries to halt the three processes at the locations $\{w, x, y\}$, and then eventually receives the racing messages into their receive buffers except P_2 which stops at non-racing receive b waiting for unsent message by P_3 . This results in the three receive buffers of (P_2, P_3, P_4) to contain three sets of messages (\emptyset, X, α) respectively, where $X = \{msg(x_1), msg(x_2)\}$ and $\alpha = \{msg(y_1), msg(y_3)\} \subseteq Y$. Consequently, this two-pass technique reports two races $\{\langle x, X \rangle, \langle y, Y \rangle\}$ as unaffected, but actually $\langle y, Y \rangle$ is affected by $\langle x, X \rangle$ as shown in Figure 2.1. This kind of erroneous reports is resulted from halting at x of P_3 , and then not having delivered $msg(y_2) \in Y$ at P_4 .

To detect only unaffected races, our previous work [13] traces the states of the locally-first race to occur in every process, and then visualizes the affect-relations of all the locally-first races with event graphs to report unaffected races. In the first execution it monitors the program execution to determine if each receive event is involved in a race and if the race occurs first in the process. During the second execution, it changes its state whenever each process receives messages, and traces the states and events appeared in each process into trace files. After the execution, it analyzes trace files, and visualizes the locally-first races with the affect-relations among those races. However, this technique sequentially analyze all trace files of processes for determining just affected messages among all the processes, and visualizes the messages in too complex a fashion for programmers to discriminate affect-relations between every two locally-first races.

4 Scalable Race Visualization

To capture the affect-relations of the locally-first races in all processes, we replace the second-pass algorithm of the previous technique [11]. Figure 4.2 shows our own pass-2 algorithm to check each receive associated locally with a sequence number *recv* with a delivered message *Msg* sent by a process *send*.

Line 1-2 uses $\{cutoff, firstChan\}$, where *cutoff* reported by the pass-1 represents an approximate location of the locally-first race occurred in the current process and *firstChan* represents a channel over which the locally-first race occurred. It checks if *cutoff* happened before *recv*, examine if *firstChan* is included in *Channels* which is a set of logical channels associated with *recv*, and check if the receive has been *affecting*. The current receive *recv* must be the first racing receive *firstRecv* to occur in the process, if *recv* associated with *firstChan* is unaffected and occurred after *cutoff*.

To produce the affect-relation information that will be attached to each message to notify other processes of their being affected, line 8 updates the information *affecting* to a disjunction of the current *affecting* and *Msg[affecting]* which is attached to the received message. It is because messages to be sent hereafter may influence other processes, if either a race occurred in the current process or affected messages have been received from other processes. The line 9 checks if *Msg[affecting]* is true. In that case, it updates a global boolean array *Affector* which indicates such processes that influence

```

15 GenerateAffecters(send, recv, Msg)
16 for all i in Channels do
17   if (cutoff → recv) ∧ (firstChan = i)
18     ∧ ¬ affecting then
19     firstRecv := recv;
20     affecting := true;
21   endif
22 endfor
23 affecting := affecting ∨ Msg[affecting];
24 if (Msg[affecting] = true)
25   Affector[send] = true;
26 endif

```

Fig. 4.2. The Tracing Algorithm

the current process. The number of entries of *Affector* is as many as the number of processes monitored in an execution, and the *i*-th entry of *Affector* indicates if a process P_i influence the current process.

Figure 4.3.a shows the *Affector* of each process generated by the algorithm which is applied to the execution shown in Figure 2.1. In the Figure 4.3.a, the locally-first races occur at three receive events $\{w, x, y\}$, and affected messages are represented with dotted arcs. >From these *Affecters*, we find that P_1 receives one affected message sent by P_2 which received three affected messages from P_1 and P_3 .

An *Affector* can be represented with a directed graph, called *race graph*, in which each vertex represents a process and each arc between two vertexes represents an affect-relation. In other words, we regard an *Affector* as an inverse adjacency list which is used to represent a directed graph. In this race graph, the vertex for a process having a locally-first race is displayed by a filled circle, and the other type of vertex as an empty circle. Figure 4.3.b shows the race graphs which represent the *Affecters* shown in Figure 4.3.a. In the figure, we see that P_2, P_3 , and P_4 have locally-first races; P_2 and P_3 are affecting each other; and P_4 is affected by P_3 . The relationship of P_2 and P_3 which are affecting each other can be visually abstracted with a rectangle as shown in Figure 4.3.b.

5 Experimentation

We tested our technique in MPI [14], an industry-standard model of message-passing parallel program, using MPICH implementation [6] on a cluster system. We installed Linux in the cluster system which consists of four Compaq-Alpha processor nodes. We implemented the on-the-fly algorithms as a C-library using MPI Profiling Interface to make it transparent to user programs, and our visualization algorithm using Java language and the *Soot* API [16].

We implemented our on-the-fly algorithms in every related function in MPICH. In addition to the tracing algorithm introduced in Section 4, we implemented three nec-

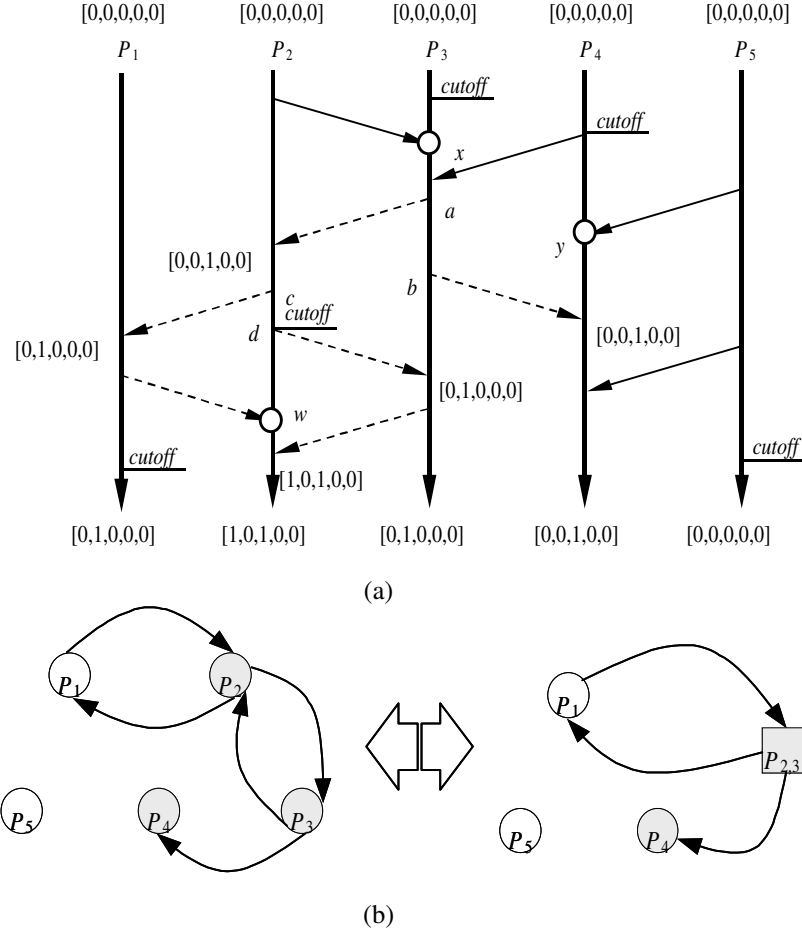


Fig. 4.3. The Affectors and Race Graphs

essary functions to support the algorithm: a function to produce a vector timestamp in each send or receive event, a function to determine and write down $\{cutoff, firstChan\}$ to a trace file in pass-1 [11], and a function to read $cutoff$ and $firstChan$ from the trace file for our pass-2 tracing algorithm. Among them, those functions related to the trace files are called only at the start or end of an execution of program, and the others are called at every send or receive event.

We implemented these functions using MPI Profiling Interface to make it transparent to user programs, so that users apply the library to their programs without modifying them. MPI Profiling Interface included in MPI specification allows anyone to intercept every call to the MPI library and perform an additional action. For this, the MPI specification states that every MPI routine is callable by an alternative name; every routine of

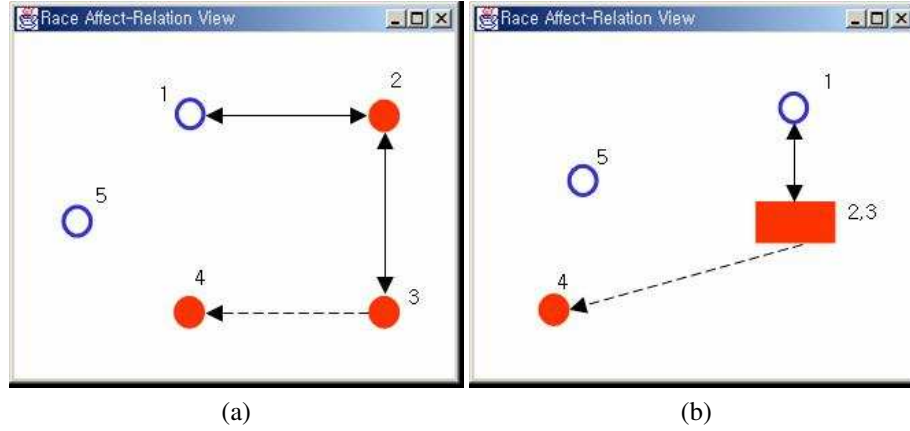


Fig. 5.4. The Scalable Graphs

the form `MPI_xxx` is also callable by the name of the form `PMPI_xxx`, allowing users to implement and experiment their own `MPI_xxx`. We wrapped a subset of the `MPI_xxx` which are related to point-to-point communication with the calls to our library functions. When a user program calls our wrapped `MPI_xxx`, this routine performs its own function and additionally executes something related to detect unaffected races.

We wrapped five MPI functions:

`MPI_Init()`, `MPI_Comm_size()`, `MPI_Send()`, `MPI_Recv()`, and `MPI_Finalize()`.

We implemented two different sets of wrapped functions for the two passes of monitored executions. Consider the pass-1 functions. `MPI_Init()` and `MPI_Comm_size()` initialize all data structures for detecting the *cutoffs*, and generating the vector timestamps. `MPI_Send()` updates the current vector timestamp, and attaches it to the message to be sent to other processes. `MPI_Recv()` updates the current vector timestamp considering the sender's timestamp received, and determines $\{cutoff, firstChan\}$. `MPI_Finalize()` stores $\{cutoff, firstChan\}$ to a trace file.

Consider the pass-2 functions. `MPI_Init()` and `MPI_Comm_size()` initialize all data structures for generating an *Affector* and read $\{cutoff, firstChan\}$ from the trace file. `MPI_Send()` updates the current vector timestamp as in the pass-1, and attaches it to the message together with the boolean value of *affecting* to be sent to other processes. `MPI_Recv()` updates the current vector timestamp considering the sender's timestamp received, determines if current receive is the first racing receive event, and updates the *Affector* with *affecting* attached in the message. `MPI_Finalize()` traces out the content of the *Affector* for the next step of race visualization.

For scalable race visualization, we implemented some Java classes for reading trace files, drawing a race graph, drawing an abstract race graph, and generating a user interface. Figure 5.4.a shows a race graph generated from the execution instance shown in Figure 2.1. In this figure, we find three processes $\{P_2, P_3, P_4\}$ in which races occurred, and the other two processes with no races. We see with ease that the race occurred in P_4 is affected by another race occurred in P_3 , which we may find with difficulties in Figure 2.1. P_2 and P_3 affect each other constructing a *strongly connected graph* for which

we display a bi-directional edge using a solid line. For scalability, each bi-directional edge connecting two vertexes can be visually abstracted to a rectangle as shown in Figure 5.4.b and vice versa.

6 Conclusion

We presented a new visualization technique that traces affect-relations of the locally-first races to occur in all the monitored processes, and then visualizes the affect-relations with scalable race graphs. To trace such affect-relations, we developed a new two-pass algorithm, by which we examine if each message is affected or not, and trace the affect-relations of the locally-first races. After the execution, we visualize the affect-relations with scalable race graphs. Our technique is effective to detect the unaffected races, because it helps users to discriminate a set of message races which includes at least one unaffected race. We have been trying to improve our visualization technique using the information associated with the source code of debugged program.

References

1. Cypher, R., and E. Leu, "The Semantics of Blocking and Nonblocking Send and Receive Primitives," *8th Intl. Parallel Processing Symp.*, pp. 729-735, IEEE, April 1994.
2. Cypher, R., and E. Leu, "Efficient Race Detection for Message-Passing Programs with Non-blocking Sends and Receives," *7th Symp. on Parallel and Distributed Processing*, pp. 534-541, IEEE, Oct. 1995.
3. Damodaran-Kamal, S. K., and J. M. Francioni, "Nondeterminacy: Testing and Debugging in Message Passing Parallel Programs," *ACM/ONR Workshop on Parallel and Distributed Debugging*, Sigplan Notices, 28(12): 118-128, ACM, Dec. 1993.
4. Damodaran-Kamal, S. K., and J. M. Francioni, "Testing Races in Parallel Programs with an OtOt Strategy," *Int'l Symp. on Software Testing and Analysis*, pp. 216-227, ACM, August 1994.
5. Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. "PVM: Parallel Virtual Machine," *A Users' Guide and Tutorial for Networked Parallel Computing*, Cambridge, MIT Press, 1994.
6. Gropp, W., and E. Lusk, *User's Guide for Mpich, A Portable Implementation of MPI*, TR-ANL-96/6, Argonne National Laboratory, 1996.
7. Kilgore, R., and C. Chase, "Re-execution of Distributed Programs to Detect Bugs Hidden by Racing Messages," *30th Annual Hawaii Int'l. Conf. on System Sciences*, Vol. 1, pp. 423-432, Jan. 1997.
8. Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, 21(7): 558-565, ACM, July 1978.
9. Lei, Y., and K. Tai, "Efficient Reachability Testing of Asynchronous Message-Passing Programs," *8th Int'l Conf. on Engineering of Complex Computer Systems* pp. 35-44, IEEE, Dec. 2002.
10. Mittal, N., and V. K. Garg, "Debugging Distributed Programs using Controlled Re-execution," *19th Annual Symp. on Principles of Distributed Computing*, pp. 239-248, ACM, Portland, Oregon, 2000.
11. Netzer, R. H. B., T. W. Brennan, and S. K. Damodaran-Kamal, "Debugging Race Conditions in Message-Passing Programs," *Sigmetrics Symp. on Parallel and Distributed Tools*, pp. 31-40, ACM, May 1996.

12. Netzer, R. H. B., and B. P. Miller, "Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs," *Supercomputing*, pp. 502-511, IEEE/ACM, Nov. 1992.
13. Park, M., and Y. Jun, "Detecting Unaffected Race Conditions in Message-Passing Programs," *11th European PVM/MPI User's Group Meeting (EuroPVM/MPI)*, Lecture Notes in Computer Science, 3241: 268-276, Springer-Verlag, Sept. 2004.
14. Snir, M., S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, 1996.
15. Tai, K. C., "Race Analysis of Traces of Asynchronous Message-Passing Programs," *Int'l. Conf. Distributed Computing Systems*, pp. 261-268, IEEE, May 1997.
16. Vallee-Rai, Raja, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot - A Java Bytecode Optimization Framework," *Conf. the Centre for Advanced Studies on Collaborative*, pp. 125-135, Sept. 1999.

Hierarchical Structures for Multi-Resolution Visualization of AMR Data

Sanghun Park

Graduate School of Digital Image & Contents
Dongguk University
Seoul 100-715, Korea
mshpark@dongguk.edu

Abstract. Adaptive mesh refinement (AMR) is a popular computational simulation technique used in various scientific and engineering fields. Although AMR data has a hierarchical multi-resolution structure implicitly, traditional direct volume rendering algorithms cannot handle the form without converting it to a sophisticated data structure. In this paper, we present hierarchical structures based on both modified k-d trees and octrees for multi-resolution display of time-varying AMR data. The proposed data structures are suitable for implementing an interactive AMR data visualization system on a general purpose PC. Experimental results show that the structures allow users to analyze the change of AMR data in process of time with properly selected level of detail.

1 Introduction

Adaptive mesh refinement (AMR) is a computational technique for improving the efficiency of numerical simulations of systems of partial differential equations. After Berger and Oliger [2] developed AMR in 1980s to simulate gas dynamics, it has become a popular computational simulation technique in various scientific computing fields. The basic idea of AMR is to refine, both in space and in time, regions of the computational domain where high resolution is needed to resolve developing features, while leaving the less interesting parts of the domain at lower resolutions. AMR techniques have been shown to be very successful in reducing the computational and storage requirements for solving many partial differential equations and used in various engineering applications where there are regions of greater interest such as global atmospheric modeling and numerical cosmology. For example, Bryan [3] shows how a hybrid approach of AMR can be applied to cosmological research.

Although AMR data has a hierarchical multi-resolution structure implicitly, it is impossible for traditional visualization techniques developed for simple mesh data to handle AMR data without any modification. Furthermore, interactive rendering of time-varying volume datasets is one of the major challenges in computer graphics. Relatively few results have been presented on visualization of AMR data. Norman et al. [6] present problems and solutions in storing, handling, visualizing, virtually navigating, and remote-serving data produced by large-scale AMR simulations. Weber et al. [8] introduce crack-free isosurface extraction methods from AMR data. They also present a

hardware-accelerated rendering interface for previewing and cell-projection based progressive refinement rendering scheme in [7]. In another paper [9], they render AMR data using the progressive cell-projection approach and level-dependent transfer function. Even though their method can produce high quality images, it takes about 23~115 seconds to render just one image from an AMR data with a $80 \times 32 \times 32$ root-grid resolution and a three-level hierarchy.

In this paper, we describe hierarchical data structures for interactive multi-resolution display of time-varying AMR data, which is designed to work efficiently when the data of interest is distributed sparsely through volume. A simple preprocessing step identifies the voxels representing features of interest. Frequently the set of voxels, arbitrarily distributed in three dimensional space, is a small fraction of the original voxel grid. An adaptive space partitioning scheme, combined with octrees to prune void spaces in the resulting search structure, is used to store the voxels of interest in a k-d tree. The tree is then efficiently splatted to render the multi-resolution voxel data. Since the structure is view independent, it can be used for animation sequences involving changes in positions of the viewer. We have applied this hierarchical structure to render volume data from time-varying AMR simulations. Experimental results obtained on a PC equipped with an NVIDIA GeForce3 graphics card demonstrate interactive rendering speed (over 20 frames per second).

The rest of this paper is organized as follows. Section 2 presents the design details of our hierarchical data structures. And Section 3 describes interactive multi-resolution display of time-varying AMR data. Experimental results are shown to analyze the performance of our hierarchical structures in Section 4. Finally this paper is concluded in Section 5.

2 Design of Hierarchical Structures for High Performance Computing

2.1 K-d trees

An AMR simulation algorithm generates a grid hierarchy data structure (like a multi-resolution tree with arbitrary grid and level). Every node and leaf of the tree is associated with a multi-resolution 3D grid. Fig. 2.1 (a) shows a simple example of raw AMR data in 2D, and (b) illustrates an implicit hierarchical representation of the data. Since an AMR data could have various shapes, sizes, and spatial resolutions, we need to convert this form to a sophisticated data structure for effective visualization.

The binary space partitioning (BSP) tree algorithm is an efficient method for calculating the visibility relationships among a static group of 3D polygons as seen from an arbitrary viewpoint. Each non-terminal node in the BSP tree represents a single partitioning plane that divides occupied space into two. A terminal node represents a region that is not further subdivided and would contain pointers to data structure representations of the objects intersecting that region. Also, the BSP structure is used as an aid to sorting planes in a scene into a back-to-front or front-to-back ordering consistent with a given viewpoint. In volume rendering, a straightforward hierarchical data structure called axis-aligned BSP trees or k-d trees is used for space subdivision. Each

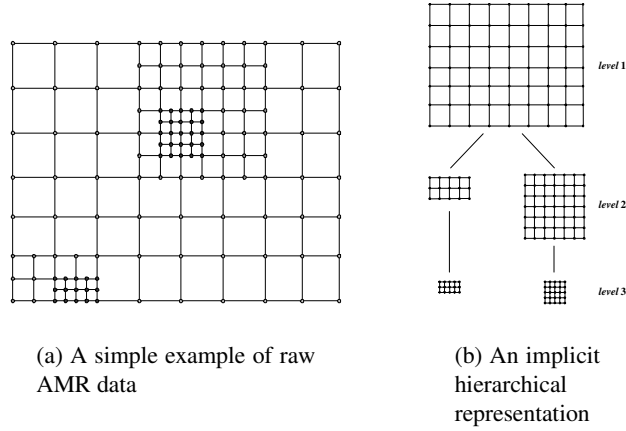


Fig. 2.1. Multi-resolution in a raw AMR data (in 2D)

non-terminal node in the k-d tree is associated with an axis aligned plane that cuts a given volume space into two, and it has a child for each subvolume. Given a k-d tree of the voxels, it is possible to find the resulting voxels in $O(\sqrt{n} + k)$ time where n is the number of voxels in the search space and k is the number of voxels in the result.

To build such a hierarchical structure in our scheme, a modified k-d tree generation algorithm is applied to a given time-varying AMR data in preprocessing steps. The first step is to determine the minimum bounding boxes surrounding each group in the AMR data space. The voxels included in a group are connected in spatial resolution of the current level and each group may include several levels of AMR data. In the next step, our scheme splits the bounding boxes into a set of bricks. We start from the root node which presents the entire volume, and bisect recursively the nodes through their largest axis, so that a nearly equal number of voxels lie in each brick. This bisection is accomplished using a modified median finding algorithm, which is an $O(n)$ average time operation per level of the tree, making the tree building process an $O(n \log_2(n))$ operation. The generated bricks have pointers to actual function value sets in each level. Fig. 2.2 (a) to (d) illustrate the modified k-d tree building algorithm applied to the simple AMR data from Fig. 2.1 (a) in 2D. And Fig. 2.2 (e) shows the generated final k-d tree. To exploit spatial coherence, our scheme constructs an octree structure for relatively large bricks.

2.2 Octrees

In our algorithm, the rendering of a selected range of isovalues, together with a transfer function is performed through splatting. While the hierarchical data structure provides an accelerated search algorithm for bricks, we further optimize our algorithm through the use of octrees at each node of the k-d tree. Each brick is represented as an octree of

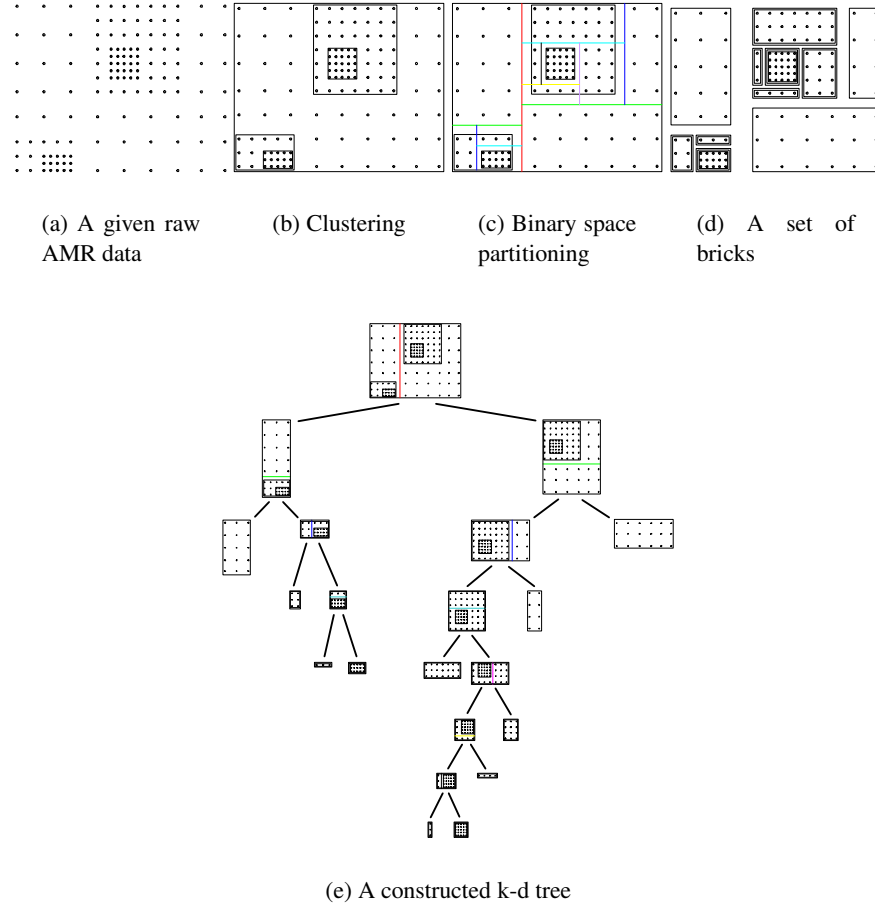


Fig. 2.2. A set of bricks and a hierarchical structure generated from the modified k-d tree building algorithm in a preprocessing step

voxels. Each voxel is then projected by transforming its position from world coordinates to screen coordinates.

In each brick and octree, we store an isovalue code to ensure that only relevant volumes are considered for the search algorithm. This isovalue code is implemented as a 32-bit number, each bit representing the presence or absence of a range of isovalues. As a preprocessing step, we compute and store the binary code. It is computed for each level of the octree. We traverse through all voxels contained in a leaf and determine the bits in the code to be turned on to indicate the presence of at least one voxel in that range. If there are n bits for the code, and we have a range of r for the voxels in the whole volume, then each bit covers a range of $\frac{r}{n}$. Once we have obtained the code for each leaf, we recursively obtain the code for a node as the boolean *OR* of the codes of

its children. The code of an octree's root is assigned to the brick containing the octree. We do not perform weighting for this code. For example, we do not assign fewer bits to non-interesting regions, hence assuming the entire range to be of equal interest to user. An *OR* operator is used to build a search code when the user changes the range of important isovalues. A simple *AND* operator is sufficient to eliminate those subvolumes whose codes do not fall under the currently selected range of isovalues. This is implemented at both the brick level and the octree level. Within an octree, each child that is not *NULL* contains such a code to help its traversal. We found this method to be considerably faster than storing min-max values. A second useful feature of the octree is the fast and natural ordering of voxels that it provides to obtain high quality images quickly. We sort bricks for each view from the k-d tree.

3 Interactive Multi-Resolution Display

3.1 Hardware Accelerated Splatting

Splatting is an object space direct volume rendering algorithm that generates high quality images [10]. A voxel's contribution is mapped directly onto the image plane, eliminating the need for interpolation. Since only interesting voxels (weighted by the discrete voxel values) are required to be represented by a 3D kernel, splatting is known as a faster volume rendering technique than ray-casting. Splatting has been used in the past to handle hierarchical error and higher dimension rendering [5, 1]. This techniques allow us to render different level data sets using footprints of various sizes. There could be regions that lack high-level data sets and can be rendered using the lower resolution, thus we need to keep a table of different footprints to render the images. The footprints themselves are discrete Gaussian footprints, which are widely used as a good approximation to the contribution of a voxel to the projection operator. By projecting the footprint to a polygon, we can exploit OpenGL 2D texture mapping hardware. Crawfis et al. propose to render each splat using texture mapping hardware [4]. This technique alleviates the CPU from the computational complexity incurred in resampling the footprint tables and compositing each splat into a frame buffer.

3.2 Rendering Algorithm

Fig. 3.3 shows our splatting algorithm of AMR data. Since creating k-d trees and octrees of the given AMR data (line 2) can be done at the preprocessing stage, they don't affect the actual run-time rendering speed. Once a k-d tree has been generated, a modified in-order traversal will yield front-to-back ordering of the bricks, the blick list *BL*, with respect to an arbitrary viewpoint (line 4). Fig. 3.4 illustrates the process of the front-to-back traversal algorithm using the k-d tree. In Fig. 3.4 (a), the numbers labeled on the bricks mean the expected traversing order for the given view vector. Fig. 3.4 (b) includes two kinds of numbers assigned to nodes in the tree. The numbers beside the circles surrounding the terminal nodes show the rendering order, and they are the same as the order in (a). Also, the other numbers written in italic font represent k-d tree traverse sequence. When the blick list *BL* is constructed, the bricks that don't contain

```

1  Load AMR data
2  Create k-d tree and octree data structure
3  Set current transfer function and viewing / rendering
   parameters
4  Determine brick list BL according to viewing direction
5  For (Bi in BL) {
6      Splatting(Bi, lod);
7      Composite the current partial image;
8  }
9  Display final image;

```

Fig. 3.3. K-d tree based splatting algorithm

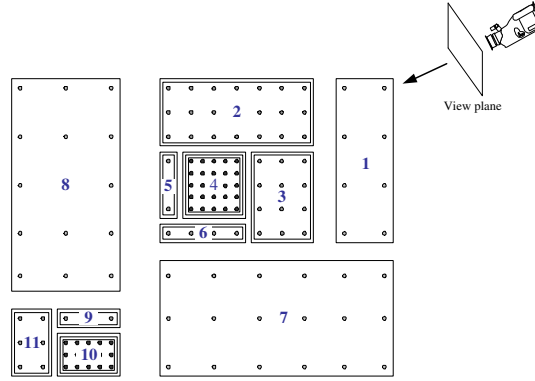
any interesting voxels are not included in the list. Because boolean encoded words are already stored in both the bricks and the octrees, a single bitwise *AND* operation is needed for checking whether the brick will be rendered or not. Splatting is then applied to each brick B_i in the sorted list of relevant bricks and the generated partial images are composited to form a final image (lines 6 and 7). Since the nodes in the brick list have pointers to function values corresponding to several levels, the rendering level of detail *lod* should be chosen, and a proper transfer function and footprint size should be used according to the *lod*. As we mentioned, it is possible to accelerate the performance of the lines 6 and 7 by texture mapping hardware.

4 Experimental Results

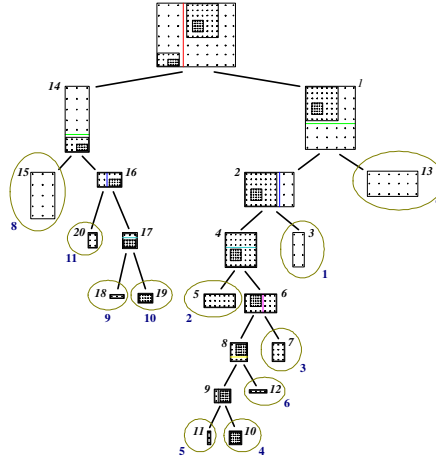
Our algorithm was implemented on a PC, equipped with a 800 MHz Intel Pentium III Processor, 256MB main memory, and a graphics card with an NVIDIA GeForce3 processor and 64MB of memory.

Time-varying AMR data is represented by $\bigcup \{f_{t,l,v}(i, j, k)\}$ where t is the timestep, l is the refinement level, and v is the index of function values. Our test data is the result from a simulation of a radiative jet colliding with a dense cloud. The simulation result is stored in time-varying AMR format with a $64 \times 64 \times 128$ finest-grid resolution and a four-level hierarchy ($0 \leq l \leq 3$). Fifteen function values are given at the nodes of the mesh in floating point format ($0 \leq v \leq 14$). We scaled the values to range from 0 to 4095. Interesting values include energy density ($v = 0$), mass density ($v = 4$), electron density ($v = 5$), and so on. The AMR data set consists of voxels sampled at different levels. While higher resolution data gives us better quality images, the lower resolution and sparsely sampled data gives faster rendering times. The image sequences in Fig. 4.5 were generated from electron density function values of the test time-varying AMR data using our technique.

Using the k-d tree data structure, we are able to effectively limit the search space. As described before, the bricks have a code representing the range of isovalues contained in the brick. This helps us to achieve considerable performance. The octree structure at each brick of the k-d tree helps to further limit the search space. It was observed that a maximum octree width of 8 was quite efficient when the number of voxels was



(a) Rendering order for a given viewing direction



(b) Front-to-back k-d tree traversal

Fig. 3.4. Determining brick list BL for a given viewpint by k-d tree traversal

greater than 40,000 to 50,000. Less dense volumes performed well with an octree size of 4. We show the results comparing the performance due to addition of the octrees and also compare the performance using 4, 8, and 16 as the maximum octree width. Fig. 4.6 (a) and (b) shows the average ratios of searched voxels and the average rendering speed resulted from rendering of our test time-varying data with about 1.26M voxels respectively.

Nearly interactive frame rates were achieved using our implementation. Although considerable gains in limiting the search space was achieved with smaller octree width

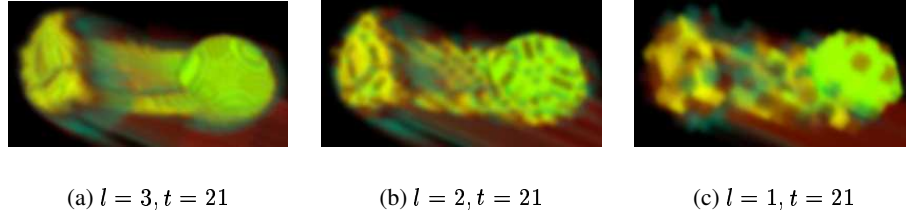


Fig. 4.5. Level of details of electron density ($v = 5$) of test AMR data

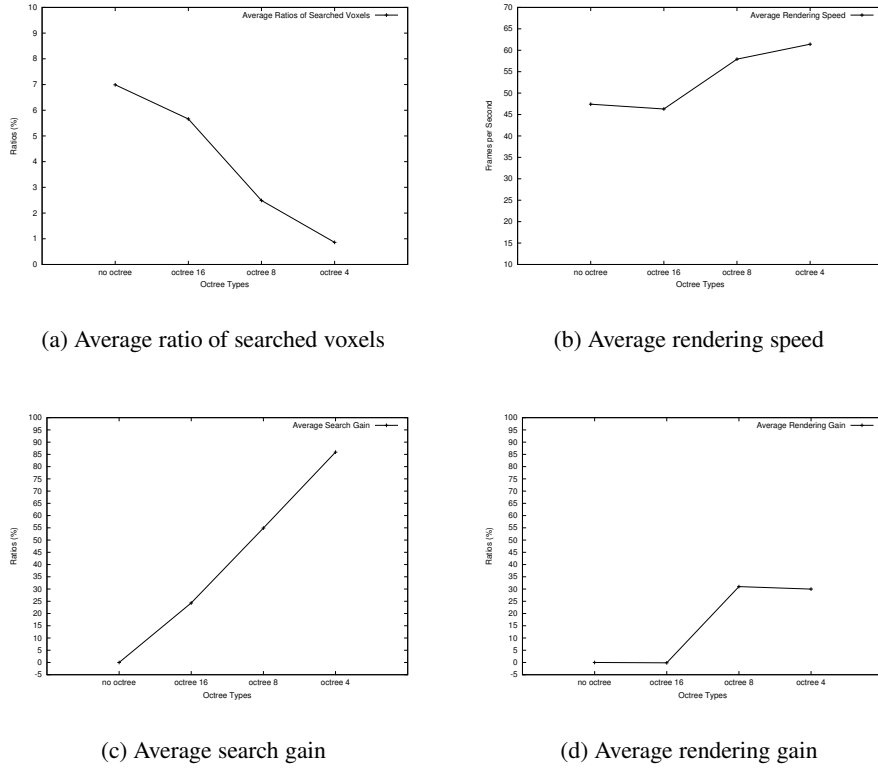


Fig. 4.6. Experimental results from interactive rendering in the finest level

values, the search time became a bottleneck. We give both the gain in the search space limitation and the speed of rendering as comparison with different octree widths. Fig. 4.6 (c) and (d) shows the resulting average search and rendering gains from generating some test images respectively. We define search gain and rendering gain as follows: $search\ gain = \frac{\eta_{without} - \eta_{with}}{\eta_{without}}$, $rendering\ gain = \frac{\tau_{without} - \tau_{with}}{\tau_{without}}$ where $\eta_{without}$ and η_{with} are

the number of voxels searched without and with octree, respectively, similarly, $\tau_{without}$ and τ_{with} are the rendering speeds without and with octree, respectively.

The data structure combination of k-d trees and octrees result in interactive rendering of AMR data sets as large as $64 \times 64 \times 128$ with highest resolutions. While we get very fast selection of regions in space where relevant voxels exist with our k-d trees, the octrees further improve search time and give a natural ordering of voxels for any view direction. Although the relevant bricks of the k-d tree are sorted for every view change, it does not prove to be a bottleneck due to the limited number of selected bricks. This is particularly true of volumetric data, where the region of interest is usually not dense throughout the object space. Hence we have partially solved the problem of quickly obtaining an ordering on the rendering primitives. This gives us significantly faster rendering speeds.

5 Concluding Remarks

We presented a hierarchical multi-resolution display scheme to render AMR data interactively. Our technique constructs k-d trees and octrees from AMR data during pre-processing. The data structures are used effectively for rendering and storing the data. The technique takes advantage of hardware accelerated 2D texture mapping to enhance rendering speed.

An important challenge is to apply our scheme to parallel rendering of AMR data. Our k-d tree based splatting scheme has a good structure to be extended to parallel rendering. The view dependent brick list can be considered as a task pool. Assume that there is a master processor and several slave processors for this parallel scheme. The master processor assigns a task to a proper slave processor and composites partial images from slave processors according to a sorted brick order. Each slave processor loads the assigned bricks, creates partial images using splatting, and then sends them to the master processor. Another challenge is to implement an encoding method exploiting temporal coherence for time-varying AMR data. If we develop lossless or lossy compression techniques, the data can be stored in compact forms and thus rendering speed to produce videos for analyzing time-varying data can be enhanced. Finally, we have designed high order normal estimation and dynamic lighting based on graphics hardware accelerated techniques to create much more interesting images.

Acknowledgement

The author would like to thank Marcelo Alvarez and Prof. Paul Shapiro of the Galaxy Formation and Intergalactic Medium Research Group at UT-Austin for providing the AMR data set. The data was from a workshop hosted by Alejandro C. Raga in Mexico City in 2002.

References

1. C. L. Bajaj, V. Pascucci, G. Rabbiolo, and D. R. Schikore. Hypervolume visualization: A challenge in simplicity. In *Proceedings of IEEE/ACM 1998 Symposium on Volume Visualization*, pages 95–102, Oct 1998.

2. M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.
3. G. L. Bryan. Fluids in the universe: adaptive mesh refinement in cosmology. *Computing in Science & Engineering*, 1(2):46–53, 1999.
4. R. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings of IEEE Visualization'93*, pages 261–267, Oct 1993.
5. D. Laur and P. Hanrahan. Hierarchical splatting: a progressive refinement algorithm for volume rendering. *Computer Graphics*, 25(4):285–288, 1991.
6. L. Norman, M., J. Shalf, S. Levy, and G. Daues. Diving deep: data-management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science & Engineering*, 1(4):36–47, 1999.
7. G. H. Weber, H. Hagen, B. Hamann, K. I. Joy, T. J. Ligocki, K.-L. Ma, and J. M. Shalf. Visualization of adaptive mesh refinement data. In *Proceedings of the SPIE (Visual Data Exploration and Analysis VIII)*, May 2001.
8. G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, and K. I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In *Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization*, pages 25–34, May 2001.
9. G. H. Weber, O. Kreylos, T. J. Ligocki, J. M. Shalf, H. Hagen, B. Hamann, K. I. Joy, and K.-L. Ma. High-quality volume rendering of adaptive mesh refinement data. In *Proceedings of the 6th International Fall Workshop on Vision, Modeling, and Visualization 2001*, pages 121–128, Nov 2001.
10. L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, 1990.

The Development of Domain Specific Languages From Scientific Libraries

Daniel Quinlan

Lawrence Livermore National Laboratory
Livermore, California USA
dquinlan@llnl.gov

Abstract. Libraries provide the application developer with convenient high-level user-defined abstractions for a specific application domain. We will describe ROSE, an object-oriented infrastructure for source-to-source translation, that provides an interface for programmers to write their own specialized translators for optimizing such abstractions. ROSE is a part of current research on telescoping languages, which provides optimizations within the use of libraries in scientific applications. This talk will describe approaches within ROSE to extend optimization techniques that are common in well defined languages, to the optimization of scientific applications using well defined libraries. We will present how high-level grammars, customized to a specific library, can be automatically generated and used to both recognize high-level abstractions within applications and also trigger the optimization of their use. The idea of higher level languages driving the generation of lower level C++ code was originally discussed by Stroustrup in 1994. The techniques presented in this talk are a special case of compiler support for high-level abstractions such as those found in object-oriented numerical libraries. Specifically in this work we utilize the semantics of the high-level abstractions and generate low-level C++ code.

A method to Derive the Cache Performance of Irregular Applications on machines with Direct Mapped Caches (Before referee)

Carsten Scholtes¹

Universität Bayreuth, Fachgruppe Informatik
Universitätsstr. 30
D-95447 Bayreuth, Deutschland
carsten.scholtes@uni-bayreuth.de

Abstract. A probabilistic method is presented to derive the cache performance of irregular applications on machines with direct mapped caches from inspection of the source code. The method has been applied to analyze both, a program to multiply a sparse matrix with a dense matrix and a program for the Cholesky-factorization of a sparse matrix. The resulting predictions are compared with measurements of the respective programs.

1 Introduction

With modern architectures, efficient use of caches is growing more and more important for performant execution of applications. Thus, given a specific architecture and a set of alternative source codes for a specific problem, it is interesting to compare the alternatives for their cache performance on the architecture under consideration.

1.1 Objectives

A method is presented which can be applied to the source code of a program to derive a set of calculations that can be evaluated in constant time and yields a probabilistic estimation of the number of cache misses to be expected when executing the program on a specific architecture. The method is applicable to architectures with one-level, direct-mapped caches. The resulting set of calculations depends on configurable parameters of the architecture like cache size, line size and the sizes of data types. Additional parameters describing the input of the application are allowed, as long as they are known beforehand or can be derived from the input in constant time. The analysis is carried out manually, although the method is designed for future automation and incorporation into a compiler. Furthermore, the method has been applied to analyze both, a program to multiply a sparse matrix with a dense matrix and a program for the Cholesky-factorization of a sparse matrix. The resulting predictions of the numbers of cache misses are compared with measurements of the respective programs.

1.2 Cache

The method presented and applied in the following considers one level of a direct mapped data cache. The memory interface is modeled as follows: The cache is subdivided into *cache lines*, which represent the minimal amount of data transferable between cache and main memory. The main memory is subdivided into *memory lines* of the same size, which are mapped round-robin to the cache lines.

As an example, a closer look is taken at the properties of an Intel Pentium 4 processor, clocked at 1.5 GHz and connected to the main memory over a system bus clocked at 100MHz. Its cache lines consist of 64 Bytes. Accesses to the 64 KByte, 4-way first level cache take between 2 and 9 CPU cycles. Accesses to the 256 / 512 KByte, 8-way second level cache take 7 CPU cycles. Fetching data from the main memory takes between 102 and 192 cycles, assuming there is no concurring activity on the system bus [13], [14], [15].

An access, which can be satisfied directly from the cache, is called a (cache) *hit*. If the cache does not contain a copy of the memory line to be accessed, the access is called a (cache) *miss*. In this case the memory line concerned is fetched from the main memory and copied to the respective cache line. An access to a memory line, that has been accessed before, is called a *reuse*. The last access to the reused memory line before the reuse is called the *source access* of the reuse. The time between source access and reuse is called the *reuse distance* of the reuse. The set of accesses occurring within the reuse distance of a reuse, thus potentially replacing the memory line to be reused, is called *interference* of the reuse.

1.3 Related work

According to the AEOS paradigm, for a specific problem domain, alternative implementations of a small kernel are collected. The most suitable implementation for a given architecture is found empirically. Projects, applying this paradigm are, for example, ATLAS [19] (BLAS library), PHiPAC [2] and FFTW [8–10]. The LAWRA project is concerned with improving the cache performance by converting fundamental numeric algorithms to recursive form [1]. M. Wolfe [20] employs a theoretical estimation of the amount of locality to guide the arrangement of nested loops in order to improve the cache performance. The examinations concentrate on minimizing the reuse distances of array accesses with affine index expressions. In [12], sets of linear diophantic equations (*cache miss equations*) are used to describe and minimize the number of misses occurring through array accesses with affine index expressions in nested loops. As parameters, the starting addresses of the arrays must be known, as well as the boundaries of the nested loops. An automatic generation of cache miss equations for suitable loops has been implemented as an extension to the SUIF compiler. In [18], the multiplication of a sparse matrix with a dense vector is examined for its cache performance, by repeatedly decomposing the algorithm into smaller subproblems. Recombining the results allows to guide optimizations such as renumbering and blocking. In [7], an automatic prediction of misses in n -fold associative caches for array accesses with affine index expressions in perfectly nested loops has been achieved by means of *area vectors*. In [6], area vectors are used to predict the numbers of misses in an associative cache during a multiplication of a sparse matrix with a dense vector and during a transposition

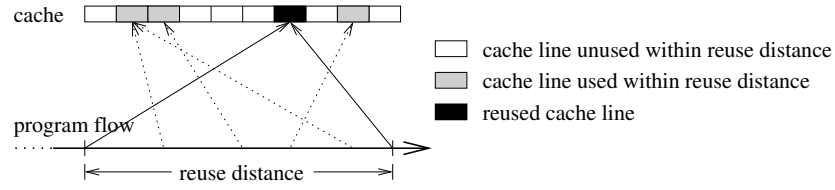


Fig. 2.1. basic assumption yields miss probability $\frac{3}{10}$

of a sparse matrix. In [3], the misses in a direct mapped cache have been predicted for different versions of the multiplication of a sparse matrix with a dense matrix. The different versions are compared with each other examining the influence of rearranging the nested loops.

2 Method

2.1 Basic Assumption

The method presented in this work is based on the following assumption [3]: The miss probability of a reuse equals the relative proportion of the cache, that has been accessed within the reuse distance.

Figure 2.1 visualizes this assumption. Depicted is a cache consisting of 10 cache lines. During the progress of the program different cache lines are accessed. Each access is marked by an arrow from the progress of the program to the cache. The reuse considered in this example and its source access are displayed as arrows with continuous lines. The dashed arrows represent interfering accesses. Within the reuse distance, 3 of the 10 cache lines are affected. Thus, according to the basic assumption, the probability for this reuse to be a miss is estimated as $3/10 = 0.3$. (Although, in this specific example, the memory line to be reused is not replaced.)

2.2 Algorithm

The algorithm in figure 2.2 outlines the method used to estimate the total number of misses.

First, all *references* (places in the program, through which memory accesses are carried out) are determined and collected in the set R . The misses are estimated individually for each reference and finally added up to the total number of misses.

For each reference $r \in R$, all important *reuse types* are determined. A reuse type of a reference r is identified by a set of accesses through the reference r containing all accesses, that have occurred before a typical distance. Since an access may belong to more than one reuse type, this distance is not necessarily the reuse distance. In order to determine the reuse distance for each access, the accesses are classified, according to their reuse types. A class is characterized by a specific set of reuse types, accounting for a reuse distance typical for all accesses in the class. Thus, the accesses of a class can be treated collectively.


```

R = set of all references;
for_all references r ∈ R {
  determination of reuse types of r;
  classification of the accesses through r; C = set of classes of r;
  for_all classes c ∈ C {
    n[c] = |c|;
    d[c] = common reuse distance in c;
    I = set of interference sets for distance d[c];
    for_all interference sets i ∈ I {
      pc[i] = affected proportion of the cache(i, d[c]);
    }
    pr[c] =  $\bigcup_{i \in I} p_c[i]$ ;
  }
  f[r] =  $\sum_{c \in C} p_r[c] \cdot n[c]$ ;
}
estimation of total number of misses =  $\sum_{r \in R} f[r]$ ;

```

Fig. 2.2. method

For each class c , its size $n[c]$ and a reuse distance $d[c]$ common for all accesses contained in the class are estimated. A miss probability $p_r[c]$ corresponding to the reuse distance $d[c]$ can then be used to estimate the number of misses caused by the accesses in the class c as $p_r[c] \cdot n[c]$. Adding up these numbers for all classes of the reference r yields the estimation of the total number $f[r]$ of misses occurring through r .

The probabilities $p_r[c]$ are estimated by means of interference sets. The interference of a reuse is the set of accesses occurring within the reuse distance. In order to derive the corresponding miss probability, first, those references are determined, through which accesses are carried out within the reuse distance. Generally, it is assumed, that the memory areas accessed through the different references are independent from each other. References, for which this is not the case, are grouped together to *interference sets*. As an example, the references r_1 and r_2 in $x = r_1[i] + r_2[i + 1]$ form an interference set. For consistency reasons, a single independent reference is also interpreted as a special, one element case of an interference set.

The proportion of the cache affected by accesses through the references of an interference set i is interpreted as the *interference probability* $p_c[i]$. Since the interference sets are designed to access independent memory areas, it is assumed that the interference probabilities are also independent from each other. Thus, the *cumulative effect* $p_c[i] \cup p_c[j]$ of two interference sets i and j can be calculated as $p_c[i] \cup p_c[j] = p_c[i] + p_c[j] - p_c[i] \cdot p_c[j]$. Since the cumulation is associative, the cumulation of n interference probabilities p_1, \dots, p_n can be written abbreviatorily as $\bigcup_{i=1}^n p_i = p_1 \cup \dots \cup p_n$.

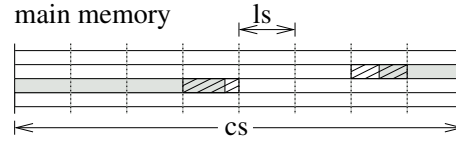


Fig. 2.3. sequential interference

With I as the set of all interference sets for a specific reuse distance $d[c]$, the total effect $p_r[c]$ of the corresponding interference on the cache can be calculated as $\bigcup_{i \in I} p_c[i]$. According to the basic assumption this is interpreted as the miss probability for a reuse with the considered reuse distance.

The interference probability of an interference set can be derived with the aid of *interference patterns* [3]. One such pattern is the *sequential interference*. This pattern is characterized by the access of a contiguous area of memory. Figure 2.3 depicts this situation, with ls denoting the size of a cache line and cs the size of the cache. The gray area represents a contiguously accessed array of n bytes. As the hatched areas indicate, more than n bytes of the cache may be affected, since an access to a specific memory address affects the entire cache line. At both ends of the accessed area between 0 and $ls - 1$ additional bytes may be concerned. Assuming that each case occurs with the same probability, this averages to $\frac{ls-1}{2}$ additional bytes on each side, totalling to $ls - 1$ additional bytes. The relative proportion $A_s(n)$ of the cache affected by such an access can therefore be estimated as $A_s(n) = \min \left\{ 1, \frac{n+ls-1}{cs} \right\}$. The result has an upper bound of 1 for the case when n is larger than the cache size.

Many other and more complex reuse patterns exist, supporting the estimation of the corresponding miss probabilities.

3 Test environment

In the following sections, the results of analyzing a matrix multiplication and a Cholesky factorization according to the method presented in section 2 are discussed. Details of the analysis can be found in [17]. The analysis yields a program which calculates an estimated number of misses. Estimations can be calculated for one level of a direct mapped cache, where the sizes of the cache, the cache line and the data types can be parameterized.

These analytically determined numbers of misses are compared with measurements. Two methods are applied to determine the number of misses empirically. This study made use of the Wisconsin Architecture Research Tool Set (WARTS) developed at the University of Wisconsin-Madison [16]. By means of the tool `qpt2` memory traces were generated which were simulated by `dineroIV` [5], yielding the total number of misses. For a more detailed analysis, the source code of the program to be analyzed was enhanced to produce a richer memory trace, which was processed by a specific cache simulator (`refsim`), yielding the numbers of misses for each reference separately.

The results of both methods differ slightly, since the compilation and instrumentation of the investigated source code can change the memory accesses carried out by the

executed program. In order to minimize such effects, the programs to be analyzed are compiled with no optimization. As compiler `gcc` 2.8.1 was used under Solaris 8.0 on a SUN Ultra-60 with 2 Ultra SPARC II processors.

4 Matrix multiplication

The cache performance of sparse matrix multiplications has been studied before. Especially in [3], three versions have been examined for their cache performance. One of these (jik) is the same as the version considered here in more detail.

4.1 Algorithm

The algorithm investigated here, is the multiplication of a sparse matrix A with a dense matrix B resulting in a dense matrix D . The sparsity of the matrix A is parameterized by the *sparsity factor* α , which is defined as the number of nonzeros divided by the number of elements in the matrix A . The dense matrices B and D are stored column wise. The sparse matrix A is stored in a compressed row storage scheme (CRS).

4.2 Experiments

The analytical estimation of misses depends on the size of the matrices and the sparsity factor α of the matrix A .

The figure 4.4 compares measured and analytically determined numbers of misses for a matrix size of 1000 elements. The sparsity factor α of the matrix A is 0.05. The x-axis shows the size of the cache and the y-axis shows the corresponding number of misses. Four graphs are displayed. The graph labeled “analysis” represents the analytically determined numbers of misses. The graph labeled “object - dineroIV” represents the numbers of misses measured by using WARTS to generate an address trace which is then evaluated by `dineroIV`. The graph labeled “source - refs” represents the numbers of misses measured by enhancing the source code with additional trace output and simulating the resulting trace with `refsim`. The graph labeled “source - dineroIV” represents the numbers of misses measured by enhancing the source code with additional trace output and simulating the resulting trace with `dineroIV`. This graph is included to demonstrate, that `refsim` and `dineroIV` deliver the same miss counts, when given the same trace.

The figure shows, that the analytically determined numbers of misses match the measured values very well. Only for larger caches there are significant differences. These can be attributed to the basic assumption. They occur, when the cache size is large enough to hold the entire data in the cache. At this point, there is virtually no more contention for cache lines and the measured numbers of misses display a sharp drop. On the other hand, according to the basic assumption, there is still a probability for a replacement miss, leading to a slower decay of the estimated number of misses. For even larger caches the graphs converge again to the number of compulsory misses.

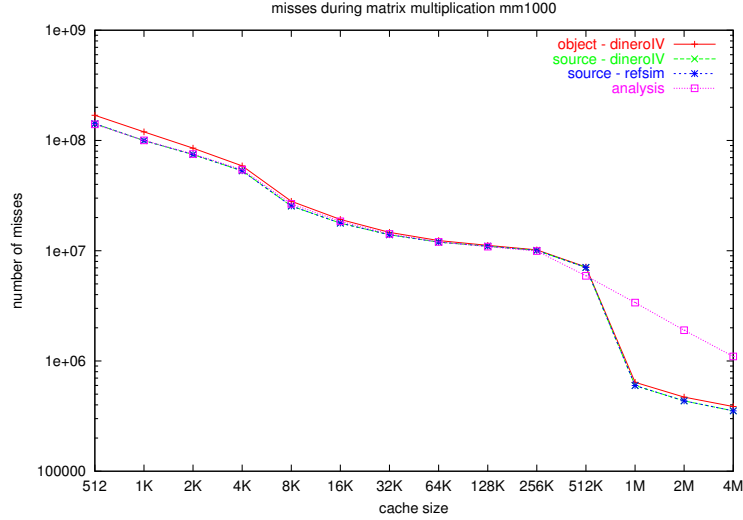


Fig. 4.4. comparison of methods for matrix size 1000

5 Cholesky factorization

In this section an implementation of a column based, right-looking Cholesky factorization of a sparse, positive definite, symmetric matrix is investigated.

5.1 Algorithm

The Cholesky factorization of a positive definite, symmetric matrix A determines a lower left triangular matrix L , such that $A = L \cdot L^T$. The investigated algorithm proceeds from left to right through the columns i of the matrix A , transforming it into the matrix L . The column i is first *completed* and then used to *modify* the columns right of itself. A *complete* operation consists of one run through the elements of the column. A *modify* operation consists of a simultaneous run through both columns concerned.

The sparsity of the matrix is exploited in several ways: The column based operations *complete* and *modify* can confine themselves to access only nonzero elements. The modification of a column j with a column i only has an effect on the modified column, if the element l_{ji} is a nonzero. Thus, many modifications can be skipped. Only nonzero elements have to be stored, lowering memory requirements.

The structure of L can efficiently be precomputed by *symbolic factorization* [11]. After initializing this structure with the lower left of the matrix A , it can be transformed in place into the matrix L . A compressed column storage scheme is used. Due to the nature of the Cholesky factorization, neighboring columns frequently share the same column structure. If this is the case, the structures are reused, further compressing the data structure.

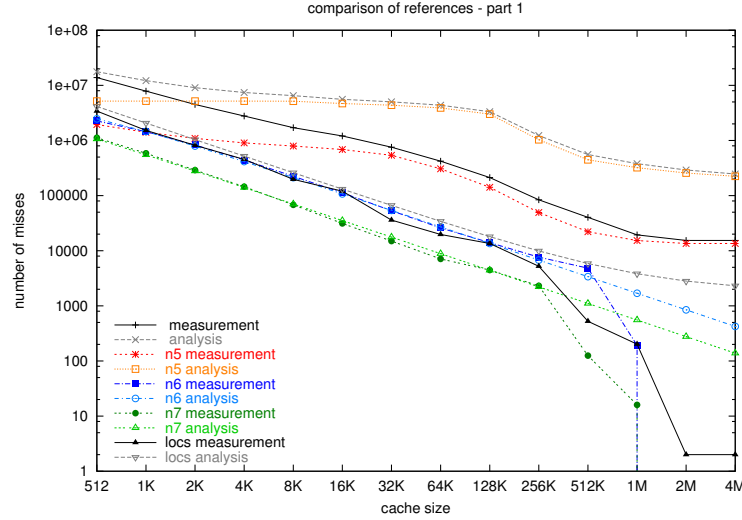


Fig. 5.5. comparison of references for matrix bcsstk14

5.2 Experiments

The memory accesses generated by the Cholesky factorization depend heavily on the nonzero structure of the matrix. Thus, apart from the size of the matrix, two additional parameters, α and β , are used, when estimating the number of misses analytically. The sparsity of the matrix is expressed by α . The value β describes the additional compression achieved by reusing the structures of neighboring columns.

Figure 5.5 shows a detailed examination of the Cholesky factorization of the matrix `bcsstk14` from the Harwell – Boeing matrix collection [4]. Measured and analytically determined numbers of misses are compared for the most important references separately as well as for the totals.

As with the matrix multiplication, the totals (analysis and measurement) reveal the shortcoming of the basic assumption, which leads to overestimations of the numbers of misses for larger cache sizes. Additionally, with the Cholesky factorization, large deviations can be observed over the whole range of cache sizes. The reference n_5 accounts for the main part of the deviations. Through this reference, the elements of the columns to be modified are read. Reuse distances for these accesses show a large variance and their estimation depends heavily on the parameter α . In the analysis, the parameter α is interpreted as the probability for any subdiagonal element to be a nonzero. Further investigations indicate that this assumption does not model the Cholesky factorization very well.

6 Conclusions and outlook

A method has been developed, that yields, based on an inspection of the source code, a set of calculations, that can be evaluated in constant time and delivers an estimation

of the number of misses to be expected. The method has been applied with very good results to the multiplication of a sparse matrix with a dense matrix. An application of the method to a sparse Cholesky factorization yields good approximations for most references, but a few references are not modeled very well.

Interesting future work includes a more detailed modeling of the Cholesky factorization, especially concerning the nonzero distribution. Furthermore, the automation of some aspects of the method can be investigated with the intention of an integration into a compiler. Also, the method shall be extended to model more complex cache architectures than a direct mapped data cache.

References

1. B. S. Andersen, F. Gustavson, and J. Wasniewski. A recursive formulation of the cholesky factorization operating on a matrix in packed storage form. Technical report, UNI•C, Lyngby, Denmark, August 1999.
2. J. Bilmes, K. Asanovic, Chee-Whye Chin, and J. Demmel. Optimizing matrix multiply using phipac: a portable, high-performance, ansi-c coding methodology. In *Proceedings of the International Conference on Supercomputing Vienna, Austria*, July 1997.
3. Ramón Doallo, Basilio B. Fraguera, and Emilio L. Zapata. Direct mapped cache performance modeling for sparse matrix operations. In *Proceedings of the 7th EUROMICRO Workshop on Parallel and Distributed Processing (PDP'99)*, pages 331–338, Funchal, Portugal, February 3-5 1999.
4. I. S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)*, 15(1):1–14, 1989.
5. Jan Edler and Mark D. Hill. *DineroIV: Trace-Driven Uniprocessor Cache Simulator*. <http://www.cs.wisc.edu/~markhill/DineroIV/>.
6. B. B. Fraguera, R. Doallo, and E. L. Zapata. Modeling set associative caches behaviour for irregular computations. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98)*, pages 192–201, Madison, Wisconsin, USA, June 22-26 1998.
7. B. B. Fraguera, R. Doallo, and E. L. Zapata. Automatic analytic modeling for the estimation of cache misses. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'99)*, pages 192–201, Newport Beach, CA, USA, October 12-16 1999.
8. M. Frigo. Fftw: An adaptive software architecture for the fft. In *Proceedings of the ICASSP Conference*, volume 3, page 1381, 1998.
9. M. Frigo. A fast fourier transform compiler. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99)*, Atlanta, GA, 1999.
10. M. Frigo and S. G. Johnson. The fastest fourier transform in the west. Technical Report MIT-LCS-TR-728, Massachusetts Institute of Technology, 1993.
11. Alan George and Joseph W-H Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey 07632, 1981.
12. S. Gosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *Proceedings of the International Conference on Supercomputing*, July 1997.
13. Intel Corporation, Intel Corporation, P.O. Box 7641, Mt. Prospect IL 60056-7641. *IA-32 Intel Architecture Optimization Reference Manual*. Order Number 248966-008.
14. Intel Corporation, Intel Corporation, P.O. Box 7641, Mt. Prospect IL 60056-7641. *IA-32 Intel Architecture Software Developer's Manual - Basic Architecture*. Order Number 245470-011.
15. Intel Corporation, Intel Corporation, P.O. Box 7641, Mt. Prospect IL 60056-7641. *IA-32 Intel Architecture Software Developer's Manual - System Programming Guide*. Order Number 245472-011.

16. James R. Larus and Eric Schnarr. EEL: Machine-independent executable editing. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation (PLDI)*, pages 291–300, Computer Science Department, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706 USA, <http://www.cs.wisc.edu/~larus/warts.html>, June 1995.
17. C. Scholtes. *Abschätzung der Fehlzugriffe bei dünn besetzten Matrixoperationen auf Architekturen mit einem direkt mapped Cache*, November 2003. Dissertation.
18. O. Temam and W. Jalby. Characterizing the behaviour of sparse algorithms on caches. In *Proceedings of the IEEE Conference on Supercomputing*, 1992.
19. R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the atlas project. Technical Report UT CS-00-448, Department of Computer Science at The University of Tennessee, September 2000.
20. M. Wolfe. *High Performance Compilers For Parallel Computing*. Addison Wesley, 1996.

Interfacing C++ member functions with C libraries

Kurt Vanmechelen and Jan Broeckhove

Department of Mathematics and Computer Sciences
University of Antwerp
B-2020 Antwerp, Belgium
{kurt.vanmechelen, jan.broeckhove}@ua.ac.be

Abstract. Interfacing functors or member functions with C libraries proves to be difficult as library routines can only accept a pointer-to-function as a callback argument. Usually this limitation is addressed by constructing an ad hoc wrapper, but this approach has several drawbacks such as the impossibility of reuse and limited support for concurrent adaptation of multiple object instances. We propose a more flexible and generic solution to the problem of mapping functor or object-member function pairs to plain C functions using recursive template programming techniques. A performance analysis of our solution is presented in order to evaluate our solution's usefulness in a high-performance computing context.

1 Introduction

In object-oriented C++ programming object-and-member-function pairs and functors occur as callable entities, besides C-style functions and function pointers [1]. Functors are simply classes that define the call operator, i.e. `operator()`, as a member function. In this contribution, we will use a class *Particle* with members *position* and *velocity* to illustrate our ideas.

```
class Particle { public:  
    double position(double time);  
    double velocity(double time);  
};
```

When two software components are developed independently, e.g. one's own code and a numerical library from a third party, they are often tied together through the *callback mechanism*. Consider for instance the computation of a particle's acceleration using the library's derivative procedure on the particle's velocity. When the library procedure, referred to as *caller*, executes, it invokes the function whose derivative must be computed. This function is the *callee*, and it is passed to the caller by way of the *callback function* argument. Thus the design of the caller also prescribes the type of the callee that it accepts. There are several approaches for the design of C++ callback libraries [2] [3] which support flexible callback constructs. They are however only applicable when both caller and callee are designed in an object-oriented fashion.

We want to look at the situation that arises when the caller is part of legacy C code. In a high-performance computing context, this scenario is commonplace as a prominent part of libraries in the HPC field only provide mechanisms to interface with C-style

functions. Relevant areas in this regard include those of quadrature and differential equation solving.

The question thus arises as to how C++ functors and member functions can be hooked into C-style callbacks. When the caller is a C procedure, as illustrated below, the callee is necessarily a pointer to function, with a type determined by its signature (argument types and the return type).

```
double derivative(double step, double x, double (*f)(double));
```

On the face of it, the member function in `Particle::velocity` has the appropriate signature, suggesting that its address can be used as callback function argument. However, a member function needs to be bound to an object instance at runtime in order to make sense, an implicit "this" pointer in its argument list points to that object. As a consequence, the member function signature is not compatible with the pointer-to-function accepted by a C-style callback.

In this contribution we want to develop a mechanism that enables object-oriented code to interface with C libraries when the connection must be made via the C-style callback. Our approach extends previous work on this mechanism for functors [4]. In the following sections we first consider the most commonly used approach, that of the ad hoc wrapper, and its limits and drawbacks. These are addressed in our approach of an adapter, which we generate through recursive template instantiation techniques. We will also present a performance analysis of our solution.

2 The ad hoc wrapper approach

This solution uses a static member function to bind the callee to the caller. An invocation of a static member function does not require a "this" pointer to provide a calling context. Pointers to such static member functions are therefore convertible to C-style function pointers that host the same method signature.

```
class ParticleWrapper { public:
    static double velocityGlue(double time)
        { return fObj->velocity(time); }
    static void setObj(Particle& obj) { fObj = &obj; }
private:
    static Particle* fObj;
};

// Bind the wrapper to particle p and call procedure
ParticleWrapper::setObj(p); double res =
derivative(0.001, 2.0, &ParticleWrapper::velocityGlue);
```

A static data member holds the reference to the callee object that will receive the adapted call. Static functions need to be defined in the wrapper for every member function that is to be adapted.

This approach is the most common way of dealing with the problem of binding member functions to C-style callbacks [5]. However, it suffers from a number of limitations. Firstly, one needs to manually define the necessary wrappers for every class and for every member function that requires adaptation. Secondly, one can only adapt


```

//The forwarding function
static R forwardCall(P1 parml)
{
    MemFuncKeyType key = (A2FMap::Instance()) [i].first;
    return (key.first->*key.second) (parml);
}
};

```

We use the *traits* [8] technique to combine all information concerning the member function's type in the `MemberFunctionTraits` class. Encapsulation of type information within a traits class increases the modularity and resulting extensibility of the template structure. The function's argument types are passed to the traits class through a `TYPELIST` construct provided by the Loki library. A `TYPELIST` is a container for types. Loki provides operations in the form of template classes to manipulate the list at compile-time.

In order to support the adaptation of the same member function for n instances of the callee class, we need to generate n static glue functions. We add an extra `int` template parameter to the wrapper class that hosts the static glue function for this purpose. The integer parameter will denote the index of the object/member function pair that will be adapted by the glue function. Every time the compiler instantiates the `Wrapper` class with a new value for i , a static glue function will be generated. To perform the instantiation process, we use the recursive template algorithm shown below.

```

template<class CT, template<class,int,int> class Glue,
        int mapMax, int i>
class GlueList { public:
    //The typelist of the previous GlueList instantiation
    typedef GlueList<CT, Glue, mapMax, i-1>::typeList pList;

    //Append a new wrapper class instantiation to the typelist
    typedef Glue<CT, mapMax, i> newGlue;
    typedef typename Append<pList, newGlue>::Result typeList;
};

//Specialization representing the base case for the recursion
template<class CT, template<class,int,int> class Glue, int mapMax>
class GlueList<CT, Glue, mapMax, 0> { public:
    typedef Glue<CT, mapMax, 0> newGlue;
    typedef TYPELIST_1(newGlue) typeList;
};

```

The i parameter specifies the number of `Glue` class instantiations that need to be made. The `GlueList` class defines a publicly available `typeList` type. At the end of the recursion, this typelist will contain all the `Glue` instantiations. In every step of the algorithm we take the list of the $i - 1$ 'th `GlueList` and append a new instantiation of `Glue` to it. The compiler continues the recursive instantiation process until i reaches 0. At this point, the specialization of the `GlueList` template for $i = 0$ is instantiated and the recursion ends.

The glue function addresses of these wrapper classes are inserted into the `IndexedMap` singleton by means of a type-iterative algorithm based on recursive template instantiation (no code shown). The algorithm iterates over the typelist constructed by the `GlueList` template. In every step of the recursion, the address of the glue function belonging to the wrapper class at the head of the list is inserted into the map. Recursion continues until the tail of the typelist equals `NullType`, indicating the end of the list.

The code fragment below demonstrates the use of our final solution by adapting the member function `velocity` of the `Particle` class defined in the introductory sec-

tion. The adapted member function is then passed on as a pointer-to-function argument of the derivative function contained in a C library.

```
//Define a 10-slot adapter and get the instance
typedef Adapter<Particle,double,TYPELIST_1(double),10> PAdapter;
PAdapter* ad = &PAdapter::Instance();

//Adapt a particle p's velocity function
PAdapter::FunctionPointerType fp=ad->adapt(p,
&Particle::velocity); double res = derivative(0.001, 2, fp);
```

4 Performance Evaluation

Our adapter provides a more generic and flexible solution to the member function adaptation problem. This section will determine the associated cost of this flexibility by comparing the performance of C callbacks using the adhoc wrapper approach versus callbacks using our adapter.

Measurements were obtained on a 2.4 GHz Pentium IV processor with 512 Kb L2 cache and 512 Mb of RAM. The adapter has been compiled and tested on the following platforms; gcc 3.2.2 and 3.3 on Solaris and SuSE Linux, Comeau 4.3 with a SunONE CC 5.1 backend on Solaris, Intel C++ 7.1 on Windows XP and SuSe Linux, Microsoft Visual 2003 C++ 7.1 and Metrowerks C++ 8.3 on Windows XP. All tests ran under a thread with critical priority. In this section, we present timings for the Visual 7.1, Intel 7.1 and gcc 3.3 compilers.

Our test setup consists of a C library function that calls back to a member function which returns the sum of two integers. We will measure the time it takes for the library function to return, i.e member function execution time is included in the measurements. In order to prevent cross-source compiler optimizations, we compiled the library source separately using the highest optimization level. We enabled automatic inlining for all compilers and optimization levels.

Intel's RDTSC [9] instruction was used to measure the execution time of the library function. The RDTSC assembly instruction returns the current 64 bit value of the Pentium's TSC (Time Stamp Counter). The TSC is reset on boot and increments every clockcycle. RDTSC reads the low-order 32 bits of the TSC into the accumulator. The RDTSC instruction does not qualify as a serializing instruction. Therefore, it may be executed out of order with respect to instructions preceding or following it. To prevent this, we issued a CPUID instruction before every call to RDTSC. CPUID returns information about the CPU and is the only serializing instruction callable from user mode. The overhead for issuing the RDTSC/CPUID instruction pair was subtracted from the measured result. The library function was called ten times. The first call includes all main memory transfer times and cache miss overhead, it serves as a warmup. We took the minimum of the other nine calls to denote the minimal execution time of the library function.

Table 1 shows the values of these measurements for different compilers, platforms and optimization levels. Optimization level O0 instructs the compiler to perform no optimizations, full compiler optimizations are performed at level O2 for the Visual compiler and at level O3 for the other compilers. The optimization flags included in the intermediate levels differ from compiler to compiler. We refer to the respective compiler manuals for a full overview.

For the Visual compiler, the extra cost of using our adapter is 12 cycles on the highest optimization level. Object code produced by the gcc compiler shows slightly higher execution times for both ad hoc and adapted cases. The extra overhead incurred by our adapter results from accessing the map structure, but more importantly, from the fact that the code for the forwarded member function did not get inlined in the glue function body, in contrast to the ad hoc case. This was determined by inspecting the generated assembly code. The table also shows that the impact of the extra statements in the adapter's wrapper function is heavily reduced by the compiler's optimizations. The OS has a small impact on the code's performance as shown by the measurements for the Intel compiler on SuSe versus those on Windows XP.

Table 1. Time per callback in clock cycles for the ad hoc case and adapted case on different compilers, platforms and optimization levels.

Optimization	VC 7.1 XP				Intel 7.1 XP				Intel 7.1 SuSe				gcc 3.3 SuSe			
	O2	O1	O0		O3	O2	O1	O0	O3	O2	O1	O0	O3	O2	O1	O0
Adapted	28	32	72		32	32	92	184	36	36	36	100	40	48	64	164
Ad hoc	16	16	16		16	16	16	32	16	16	16	40	20	20	20	40
Overhead	12	16	56		16	16	76	152	20	20	20	60	20	28	44	124

Previous work [4] using the same test setup, showed a smaller overhead of 10% for the Visual compiler when adapting a functor's call operator. In the functor case, the operator call was hard coded into the wrapper's glue function, which enabled the compiler to inline its code.

5 Conclusion

A flexible solution was presented to tackle the problem of adapting member functions to C-style function pointers. This is important in order to use legacy high-performance computing libraries with object-oriented C++ code. In contrast to the ad hoc wrapper solution, our solution allows for the adaptation of multiple object and member function types. The number of object/member function pairs that can be adapted is tunable at compile time on a type to type basis. Performance analysis has quantified the overhead of our solution compared to the ad hoc approach, and has shown the effect of compiler optimizations in this regard.

References

1. J. Barton and L. Nackman. *Scientific and engineering C++*. Addison-Wesley, 1994.
2. R. Hickey. Callbacks in C++ Using Template Functors. *C++ Report*, 7(2), pages 42-50, February 1995.

3. P. Jakubic. Callback Implementations in C++. Proceedings of the 23rd Technology of Object-Oriented Languages Conference, TOOLS-23, pages 377-406, Eds. IEEE Computer Society Press., Santa Barbara, CA, USA, July 1997.
4. J. Broeckhove and K. Vanmechelen. *Using C++ functors with legacy C libraries*. Proceedings of the 2004 International Conference on Computational Science and its Applications, ICCSA 2004, number 3046 in Lecture Notes in Computer Science, pages 514-523, Assisi, Italy, May 2004.
5. L. Haendel. The function pointer tutorials. <http://www.function-pointer.org>.
6. A. Alexandrescu. *Modern C++ Design*. Addison-Wesley, 2001.
7. D. Vandevoorde and N. Josuttis. *C++ templates*. Pearson Education, 2003.
8. N. Meyers. Traits: A New and Useful Template Technique. *C++ Report*, 7(5), pages 32-35, June 1995.
9. Intel Corporation. *IA-32 Intel(R) Architecture Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z*. Intel Corporation, 2004.

Compilation Techniques for a Chip-Multiprocessor with Two Execution Modes

Chao-Chin Wu

Department of Computer Science and Information Engineering
National Changhua University of Education, 500 Changhua, Taiwan, R.O.C.
ccwu@cc.ncue.edu.tw

Abstract. We have proposed a novel chip multiprocessor (CMP) that supports the speculative multithreading mode and the wide-superscalar execution mode. The former mode behaves like a conventional CMP, while the latter mode integrates all the processing elements into a single-logic superscalar processor. Furthermore, we extend this innovative microarchitecture to support a third execution mode, whereby the processor keeps switching between the first and second modes when executing an application, according to the characteristics of subsequent instructions. Since the speculative multithreading outperforms the wide superscalar when the former mode can exploit more parallelism from different tasks, loop optimization techniques have been proposed in this paper to exploit better loop-level parallelism. Through the assistance of our proposed compilation techniques, the system performance of the novel CMP architecture can be further improved. According to the simulation results, the best speedup is about four .

1 Introduction

Recent studies have shown that a conventional chip multiprocessor (CMP) cannot outperform a superscalar processor when executing integer operation-intensive applications [1]. Therefore, we have proposed a novel microprocessor that supports the speculative multithreading mode and the wide-superscalar execution mode [2]. Both execution modes provide a peak issue rate of sixteen instructions per cycle. The former mode behaves like a conventional CMP, while the latter mode integrates all the processing elements into a single-logic superscalar processor. Furthermore, we extend this innovative microarchitecture to support a third execution mode, whereby the processor keeps switching between the first and second modes when executing an application, according to the characteristics of subsequent instructions. As a result, this third mode can use both the advantages of a CMP and of a superscalar to execute an application. According to the performance analysis, our processor can provide an optimum system performance for all benchmark programs, regardless of workload characteristics. Furthermore, our CMP outperforms a conventional CMP, exhibiting a speedup of up to 1.32.

To improve an application's performance executing on this new microprocessor, compiler techniques can be applied to judge which parts of the application should be executed by which execution mode. The speculative multithreading outperforms the wide superscalar only when the former mode can exploit more parallelism from different tasks. If complicated data and control dependencies exist between tasks, data dependence violations may occur frequently in the speculative executions. As a result,

the speculative executions of these tasks have to be terminated and restarted, which will degrade the system performance dramatically. Therefore, in this paper we have developed a compilation technique to reduce or even eliminate the occurrences of dependence violations.

A lot of research have been focusing on how to parallelize application programs to achieve better speedups [?]. To parallelize programs, they rely on various dependence test approaches to help the analysis of data dependence [3, ?]. Since the unique feature of the speculative multithreading mode is that, at most, four tasks are executed in parallel at any time, loops with dependence distance equal to or larger than four will not incur dependence violations. Consequently, compilers may only be concerned with loops with dependence distances less than four. The goal of our optimization is to lengthen the dependence distance to be larger than three. The method is to extract several independent loop iterations from another task and then to insert these iterations into a task that is with dependence distance less than four. Of course, we have to make sure that after the above transformation, the execution result remains the same as the original program. Through the assistance of compilers, the system performance of the novel CMP architecture can be further improved. According to our simulation study, the best speedup of the compilation optimization approach is four.

The remainder of this paper is organized as follows. Section 2 introduces the main idea of the proposed compilation techniques to improve the system performance of our CMP architecture. Section 3 describes the detailed techniques for handling different dependence distances. Section 4 presents a preliminary performance analysis, and finally, Section 5 provides some concluding remarks.

2 The main Idea

First, we review the features of the speculative multithreading mode. In the speculative multithreading mode, each of the four PEs will execute one loop iteration, respectively. When encountering a loop, the first four consecutive iterations will be allocated to the four PEs in order. The first iteration executed on PE_0 is called *nonspeculative* iteration because it has no preceding iterations. The other three iterations are called *speculative* iterations because their executions may depend on the results produced by the their preceding iteration(s). When a speculative iteration requires an operand value to execute an instruction and the value has not been produced by its preceding iteration, the speculative iteration will predict a value for the operand to accelerate its execution. If the predicted value is not the same as the real value produced afterward, a *dependence violations* will occur and the speculative iteration has to be flushed and restarted. After the first iteration is completed, the fifth iteration will be executed on PE_0 . At this time, the second iteration becomes a nonspeculative one and the fifth iteration is a speculative iteration. In summary, all loop iterations will be allocated to the four PEs in a round-robin fashion. Moreover, at any time, at most four consecutive iterations will be executed in the speculative multithreading mode. Consequently, loops with dependence distances larger than three will not incur any dependence violations when being executed in the speculative multithreading mode. On the other hand, loops with dependence distances less than four may encounter dependence violations when the CMP

executes these loops in the speculative multithreading mode, which will degrade the system performance severely.

For the loops with dependence distances less than four, we propose the following compilation technique to avoid the occurrences of dependence violations in the speculative multithreading mode. The main idea is that for a loop with a dependence distance less than four, we can extract several iterations from another loop and insert them between two dependent iterations of the loop. By this way, the dependence distance may be lengthened to be larger than four. As a result, the loop can be executed in parallel without dependence violations in the CMP. We illustrate the idea by the following example. Assume that there are three consecutive iterations, said i , $i+1$ and $i+2$, in the loop L_A as shown in Fig. 2.1. For the three iterations, only iteration $i+2$ need to wait for the results produced by iteration i . Consequently, these three iterations cannot be executed in parallel without dependence violations in our proposed CMP. To allow the three iterations to be executed in parallel in our CMP without dependence violations, we can extract two independent loop iterations, said j and $j+1$, from another loop L_B and insert them between the iterations $i+1$ and $i+2$. Consequently, the original loop L_A has two more iterations, we call the augmented loop L'_A . Of course, the indexes and array subscripts of the loops L_A and L_B have to be remapped accordingly. By the insertion of two more iterations into the loop L_A , the dependent distance between iterations i and $i+2$ is lengthened from two to four. Consequently, the four consecutive iterations i , $i+1$, $i+2$ and $i+3$ as well as the four consecutive iterations $i+1$, $i+2$, $i+3$ and $i+4$ are all independent, respectively. In summary, we can lengthen the dependence distance for a loop by inserting additional independent iterations from another independent loop.

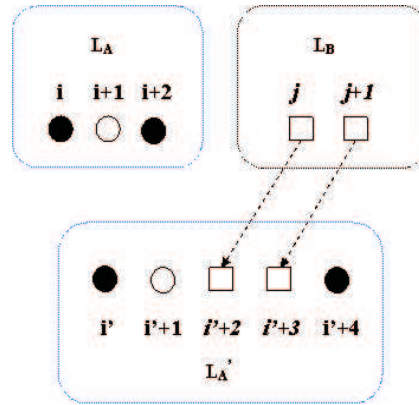


Fig. 2.1. The insertion of two independent iterations to lengthen the dependence distance. Iteration $i+2$ depends on iteration i in the loop L_A , i.e. the dependence distance is equal to two. After inserting two independent iterations j and $j+1$, belonging to the loop L_B , between iterations $i+1$ and $i+2$ in the loop L_A , the dependence distance is lengthened to be four.

To provide enough additional iterations to lengthen dependence distances, we can first apply conventional compilation techniques to transform loops into parallel loops, e.g., loop distribution, loop fusion, loop alignment, etc [3]. Moreover, since a loop with a dependence distance of four or larger will not suffer from dependence violations in the speculative multithreading mode, the parallel loops in our CMP can be redefined to be the loops whose cross-iteration dependence distances are larger than three. In the next subsection, we will explain more detailed designs for lengthening dependence distances.

3 Implementation

First, we consider the case that a loop is with a minimum dependence distance equal to three in this subsection. In other words, there are no dependence distances less than three. Although the loop may have dependence distance larger than three, we do not care about these dependencies because they will not be violated when being executed in our CMP. Obviously, when executing the loop with dependence distance of three in the speculative multithreading mode, we may encounter dependence violations. To avoid dependence violations, we require to extract independent iterations from other loops and then to insert these iterations into the loop. The iteration insertion policy is that one extracted iteration is inserted after every three consecutive iterations. We illustrate the idea by an example shown in Fig. 3.2. In the original loop as shown in Fig. 3.2(a), the iteration 3 depends on the results produced by the iteration 0, indicated by an arrow. According to the insertion policy, an independent iteration has to be inserted between iterations 2 and 3. Similarly, another independent iteration has to be inserted between iterations 5 and 6. The augmented loop with two inserted iterations is shown in Fig. 3.2(b). The inserted iterations are differentiated by two rectangles. The augmented loop indexes have to be rearranged. The two inserted iterations are with the new indexes of three and seven, respectively. After the insertion, the dependence distance is lengthened from three to four. Consequently, the augmented loop can be executed in the speculative multithreading mode without suffering from dependence violations.

It is important how to provide additional and enough independent iterations to lengthen the dependence distance to four or larger. If there are N iterations in a loop L_A , with the minimum dependence distance of three, we need to extract $\frac{N}{3}$ additional iterations from other loops. A loop L_B that can provide iterations should meet the following requirements. (1) The loop is preceding or following the loop L_A . (2) Remove the candidate iterations from the loop L_B will not influence the original dependencies in the loop L_B . (3) Move the candidate iterations to the loop L_A cannot alter the dependencies in the loop L_A . (4) Move the candidate iterations to the loop L_A cannot change the execution result. (5) The dependence distance of L_B can be any value because any two consecutive iterations will be inserted into L_A and separated by three L_A iterations. Of course, after extracting some iterations from L_B , we have to reassign the indexes of the remaining iterations for L_B .

We give a simple example to detail the proposed compilation technique as shown in Fig. 3.3. There are two *for* loops. The first one has a cross-iteration dependence with

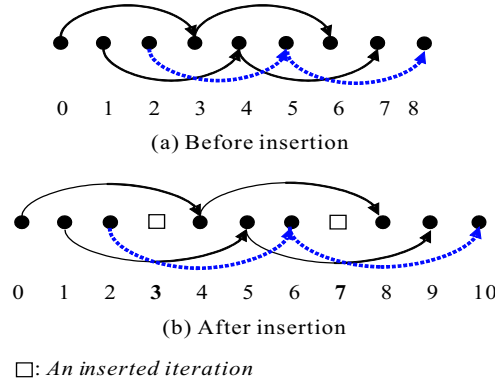


Fig. 3.2. Lengthen the dependence distance from three to four. (a) A loop with a dependence distance of three. The arrows indicate dependence relations. For example, iteration 3 depends on the results produced by iteration 0. (b) Insert one independent iteration, extracted from another loop and depicted by a rectangle, after every three consecutive iterations. The resulting dependence distance is four.

distance equal to three and the second one has no cross-iteration dependence as shown in Fig. 3.3(a). Because there are no dependencies between these two loops, we can extract $\frac{N}{3}$ iterations from the second loop. Consequently, the first loop has to execute $(N + \frac{N}{3})$ iterations and the second one has only $(M - \frac{N}{3})$ iterations left. The indexes of the two loops are modified accordingly as shown in Fig. 3.3(b). Furthermore, in the first loop, we use an *if-else* statement to execute the original statement and the inserted statement interleaving. The array subscripts of the two assignment statements in the first loop are also modified as shown in Fig. 3.3(b) to enforce the original semantics.

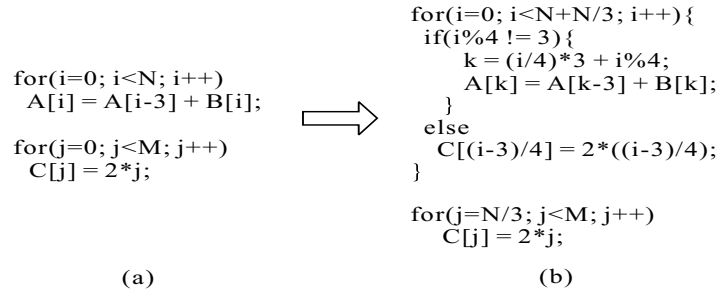


Fig. 3.3. An example of code transformation for enlarging the dependence distance. (a) The original first loop is with a dependence distance of three. (b) The transformed first loop has a dependence distance of four.

```

SUBROUTINE BUBBLE( TEN, J1, J2, J3, M, IND )
  REAL*8 TEN( M, 0:1 )
  INTEGER M, IND, J1( M, 0:1 ), J2( M, 0:1 ), J3( M, 0:1 )
  REAL*8 DIR, TEMP
  INTEGER I, JXTEMP
  IF( IND.EQ. 1 ) THEN
    DIR = +1.0D0
  ELSE
    DIR = -1.0D0
  ENDIF
  DO 100 I=1, M-1
    IF( DIR*TEN(LND).GT. DIR*TEN(I+1,IND) ) THEN
      TEMP = TEN( I+1, IND )
      TEN( I+1, IND ) = TEN( I, IND )
      TEN( I, IND ) = TEMP
      JXTEMP = J1( I+1, IND )
      J1( I+1, IND ) = J1( I, IND )
      J1( I, IND ) = JXTEMP
      JXTEMP = J2( I+1, IND )
      J2( I+1, IND ) = J2( I, IND )
      J2( I, IND ) = JXTEMP
      JXTEMP = J3( I+1, IND )
      J3( I+1, IND ) = J3( I, IND )
      J3( I, IND ) = JXTEMP
    ELSE
      RETURN
    ENDIF
  100 CONTINUE
  RETURN
END

```

(a)

```

SUBROUTINE BUBBLE( TEN, J1, J2, J3, M, IND )
  REAL*8 TEN( M, 0:1 )
  INTEGER M, IND, J1( M, 0:1 ), J2( M, 0:1 ), J3( M, 0:1 )
  INTEGER FLAG( M )
  REAL*8 DIR, TEMP
  INTEGER I, JXTEMP
  IF( IND.EQ. 1 ) THEN
    DIR = +1.0D0
  ELSE
    DIR = -1.0D0
  ENDIF
  DO 100 I=1, M-1
    IF( DIR*TEN(LND).GT. DIR*TEN(I+1,IND) ) THEN
      FLAG(I)=1
      TEMP = TEN( I+1, IND )
      TEN( I+1, IND ) = TEN( I, IND )
      TEN( I, IND ) = TEMP
    ELSE
      FLAG(I)=0
    ENDIF
  100 CONTINUE
  DO 200 I=1, M-1
    IF( FLAG(I).EQ. 1 ) THEN
      JXTEMP = J1( I+1, IND )
      J1( I+1, IND ) = J1( I, IND )
      J1( I, IND ) = JXTEMP
    ENDIF
  200 CONTINUE
  DO 300 I=1, M-1
    IF( FLAG(I).EQ. 1 ) THEN
      JXTEMP = J2( I+1, IND )
      J2( I+1, IND ) = J2( I, IND )
      J2( I, IND ) = JXTEMP
    ENDIF
  300 CONTINUE
  DO 400 I=1, M-1
    IF( FLAG(I).EQ. 1 ) THEN
      JXTEMP = J3( I+1, IND )
      J3( I+1, IND ) = J3( I, IND )
      J3( I, IND ) = JXTEMP
    ELSE
      RETURN
    ENDIF
  400 CONTINUE
  RETURN
END

```

(b)

```

SUBROUTINE BUBBLE( TEN, J1, J2, J3, M, IND )
  REAL*8 TEN( M, 0:1 )
  INTEGER M, IND, J1( M, 0:1 ), J2( M, 0:1 ), J3( M, 0:1 )
  INTEGER FLAG( M )
  REAL*8 DIR, TEMP
  INTEGER I1, I2, I3, I4, JXTEMP
  IF( IND.EQ. 1 ) THEN
    DIR = +1.0D0
  ELSE
    DIR = -1.0D0
  ENDIF
  DO 100 I=1, 3*(M-1)
    IF( MOD(I, 3).EQ. 1 ) THEN
      I1 = ( I/3 ) + MOD(I, 3)
      IF( DIR*TEN(LND).GT. DIR*TEN(I1,IND) ) THEN
        FLAG(I1)=1
        TEMP = TEN( I1+1, IND )
        TEN( I1+1, IND ) = TEN( I1, IND )
        TEN( I1, IND ) = TEMP
      ELSE
        FLAG(I1)=0
      ENDIF
    ELSE IF( MOD(I, 3).EQ. 2 ) THEN
      I2 = ( I/3 ) + MOD(I, 3)
      IF( FLAG(I2).EQ. 1 ) THEN
        JXTEMP = J1( I2+1, IND )
        J1( I2+1, IND ) = J1( I2, IND )
        J1( I2, IND ) = JXTEMP
      ENDIF
    ELSE IF( MOD(I, 3).EQ. 3 ) THEN
      I3 = ( I/3 ) + MOD(I, 3)
      IF( FLAG(I3).EQ. 1 ) THEN
        JXTEMP = J2( I3+1, IND )
        J2( I3+1, IND ) = J2( I3, IND )
        J2( I3, IND ) = JXTEMP
      ENDIF
    ELSE IF( MOD(I, 3).EQ. 0 ) THEN
      I4 = ( I/3 ) + MOD(I, 3)
      IF( FLAG(I4).EQ. 1 ) THEN
        JXTEMP = J3( I4+1, IND )
        J3( I4+1, IND ) = J3( I4, IND )
        J3( I4, IND ) = JXTEMP
      ELSE
        RETURN
      ENDIF
    ENDIF
  100 CONTINUE
  RETURN
END

```

(c)

Fig. 3.4. A code transformation for enlarging the loop dependence distance of a bubble sort subroutine. (a) The original loop is with a dependence distance of one. (b) After we apply the extended loop distribution transformation to the original loop, there are four loops and each is with dependence distance of one. (c) After applying our proposed method, we have the transformed loop with a dependence distance of four.

If a loop is with a minimum dependence distance of two, we have to insert two additional independent iterations after every two consecutive iterations of the loop. In addition, to allow four consecutive iterations in the augmented loop to be executed in parallel without incurring dependence violations in the speculative multithreading mode, a candidate loop for providing additional iterations must have a dependence distance larger than one.

Finally, if a loop is with a minimum dependence distance of one, we have to insert three additional independent iterations between every two consecutive iterations of the loop. Moreover, a candidate loop for providing additional iterations must have a dependence distance larger than two.

Furthermore, in the below we will present how to transform a more complicated real application code to enlarge its cross-iteration dependence distance. The example is a *bubble sort* subroutine extracted from the *mgrid* application in the SPEC CPU2000 benchmark suite. The *bubble sort* subroutine as shown in Fig. 3.4(a) consists of only one loop and there is an *if-else* statement inside it. Because our proposed approach works only when there are multiple loops, we apply the extended loop distribution method to the original code as shown in Fig. 3.4(b). The conventional loop distribution method transforms a loop into two parts: one has no cross-iteration dependence and the other does [3, ?]. However, in our case we will split a loop into several small loops partitioned

by dependence graphs. It is not necessary that one of the small loops has to be with no cross-iteration dependence. In the example case as shown in Fig. 3.4(b), each one of the four new small loop has a dependence distance of one. Furthermore, the *if-else* statement in the original loop body is converted according to the mechanism proposed in the Reference [13]. Finally, we can merge all the four new small loops into a single one loop and let the four new loop be executed in turn in the merged loop as shown in Fig. 3.4(c). As a result, the dependence distance of the merged loop is four, which will avoid any dependence violations in the speculative multithreading mode. Of course, to preserve the correctness of the whole application program, both the loop index and the related array subscripts have to be modified.

4 Performance Analysis

We have constructed a simulator in C language to analyze the performance gain of our proposed compilation techniques. The simulation model is as follows. There are four PEs in the CMP architecture. A loop is comprised of four loop iterations, each iteration consists of fifty million instructions. If no cross-iteration dependence exists or the dependence distance is larger than four, fifty million cycles are required to finish the execution of the loop. This is the idea case after applying our method. On the other hand, for a dependence distance of one, PE_1 , PE_2 and PE_3 depend on PE_0 , PE_1 and PE_2 respectively. For a dependence distance of two, PE_2 and PE_3 depend on PE_0 and PE_1 respectively. Finally, for a dependence distance of three, PE_3 depends on PE_0 . A simulation parameter called *dependence_probability* defines the probability that whether or not an instruction executed on a PE depends on the results produced by another PE. Moreover, the *dependence_probability* is varied according to the following formula during the execution of a program:

$$\frac{(\%PE_c + (1 - \%PE_p))}{2} \times \%D. \quad (4.1)$$

In the above formula, the variable $\%D$ represents a simulation parameter called *probability_rate* that remains constant when executing a program. $\%PE_c$ represents the percentage of the remaining instruction count in the PE where the instruction resides. $\%PE_p$ represents the percentage of the remaining instruction count in the PE where the dependent result will be produced. According to the formula, the instruction at the beginning of an iteration has the highest possibility that it depends on an instruction on another PE. Furthermore, it has higher possibility that the dependent result will be produced by an instruction that is more close to the end of an iteration. The distribution of instruction dependencies modelled by the above formula is happened frequently in a real program.

Moreover, the probability that a dependent instruction will encounter a dependence violation is defined by the following formula:

$$\frac{(\%PE_c + \%PE_p)}{2} \times \%V. \quad (4.2)$$

In the above formula, the variable $\%V$ represents a simulation parameter called *violation_rate*. According to the formula, the instruction at the beginning of an iteration

has the highest possibility that it will encounter a dependence violation. Furthermore, it has lower possibility that the producer instruction on another PE will encounter a dependence violation when the producer instruction is more close to the end of an iteration.

First, we compare the performances for different violation_rates by the case that the dependence distance is one. Regardless of the violation_rate, our approach gets better performance when the dependence probability becomes higher as shown in Fig. 4.5(a). The best speedup of our approach is about four for those violation_rates larger than or equal to 0.0001. It means that if there are too many dependencies between PEs, executing four iterations in parallel has little performance gain. Apparently, these four iterations are executed one by one although speculation techniques are applied. However, by adopting our approach, the dependence distance can be lengthened to four, resulting in no dependencies between these four iterations. As a result, four iterations can be executed in parallel and finished at the same time.

Next, we investigate the impacts of the dependence distance as shown in Fig. 4.5(b). Obviously, the dependence distance of one imposes the largest amount of dependencies in the original loop. Consequently, applying our approach to lengthen the dependence distance can provide excellent performances. The best speedup of our approach is about four that is the upper bound for the dependence distance of one. On the other hand, for the dependence distances of two and three, the upper bound of the speedup is only two. For the dependence distances of two, PE_2 and PE_3 depend on PE_0 and PE_1 respectively. The worst case before applying our approach is that the speculative executions of PE_2 and PE_3 are always in vain before the completions of PE_0 and PE_1 . As a consequence, the CMP will first finish the executions of PE_0 and PE_1 at the same time, and then complete the executions of the PE_2 and PE_3 at the same. As for the dependence distance of three, the worse case is that the execution of PE_3 will keep restarting before the completions of PE_0 , PE_1 and PE_2 . In other words, after PE_0 , PE_1 and PE_2 all finish their executions at the same time, PE_3 can start its execution. Therefore, the best speedup after adopting our approach are two as shown in Fig. 4.5(b) when the dependence distance is two or three.

5 Conclusions

We have developed a novel CMP architecture that supports two execution modes: the speculative multithreading mode and the wide superscalar mode. The advantage is that the CMP will switch to the appropriate execution mode according to the subsequent instructions. As a result, our CMP can provide superior performance regardless of the types of workloads. Because the speculative multithreading mode can exploit both the instruction- and the thread-level parallelism, it is easier to have higher degree of parallelism than the wide superscalar mode. The best case is that each four consecutive threads are totally cross-thread independent. That is, the dependence distance has to be larger than three. Therefore, in this paper we have proposed a compilation technique to lengthen the dependence distance to four. To lengthen the dependence distance for a loop, we extract some iterations from other loops that are independent with the loop. The candidate loops that can provide iterations must meet several requirements to main-

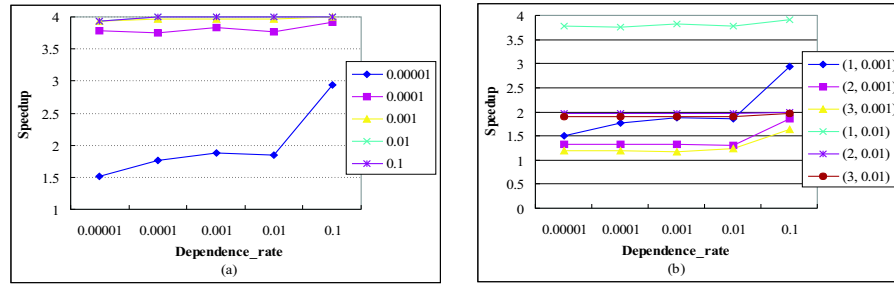


Fig. 4.5. Comparisons of the speedups of our compilation technique under different cases. (a) The speedups for different violation rates. The dependence distance is one. (b) The speedups for different dependence distances. (S , L) in the legend represent the dependence distance is S and the violation_rate is L .

tain the program correctness. According to the performance analysis, the best speedup of our approach is four.

Acknowledgements This research was supported by the National Science Council of R.O.C. under contract number: NSC-92-2213-E-018-002.

References

1. V. Krishnan and J. Torrellas. A Chip-Multiprocessor new block Architecture with Speculative Multithreading. *IEEE Trans. on Computers*, 48(9):866–880, 1999.
2. C.-C. Wu. Embedding a Superscalar Processor onto a Chip Multiprocessor. *Microprocessors and Microsystems*, 28(4):147–156, 2004.
3. R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann, San Francisco, USA, 2001.
4. A. Bhowmik and M. Franklin. A General Compiler Framework for Speculative Multithreaded Processors. *IEEE Trans. Para. & Distri. Syst.*, 15(8):713–724, 2004.
5. Roch Georges Archambault and R.J. Blainey. *United States Patent 6,651,246*, November 18, 2003.
6. F. Dang, H. Yu, L. Rauchwerger. The R-LRPD test: Speculative Parallelization of Partially Parallel Loops. *Proc. Int'l Parallel and Distributed Processing Symposium*, pages 20–29, 2002.
7. C.-Z. Xu and V. Chaudhary. Time Stamp Algorithms for Runtime Parallelization of DOACROSS Loops with Dynamic Dependences. *IEEE Trans. Para. & Distri. Syst.*, 12(5):433–450, 2001.
8. K.W. Harris and W.B. Noyce. *United States Patent 5,481,723*, January 2, 1996.
9. K. Psarris and K. Kyriakopoulos. An Experimental Evaluation of Data Dependence Analysis Techniques. *IEEE Trans. Para. & Distri. Syst.*, 15(3):196–213, 2004.
10. P. Bulic and V. Gustin. Fast Dependence Analysis in a Multimedia Vectorizing Compiler. *Proc. Euromicro Conf. Parallel, Distributed and Network-Based Processing*, pages 176–183, 2004.
11. M. Mineo, T. Uehara, S. Saito, Y. Kunieda. A New practical Array Data Dependence Analysis for Parallelizing Compilers. *Proc. Innovative Architecture for Future Generation High-Performance Processors and Systems*, pages 78–87, 2003.

12. K. Kyriakopoulos and K. Psarris. Data Dependence Analysis for Complex Loop Regions. *Proc. Int'l Conf. Parallel Processing*, pages 195–204, 2001.
13. K. Kennedy and K.S. McKinley. Loop Distribution with Arbitrary Control Flow. *Proc. Int'l Conf. Supercomputing*, pages 407–416, 1990.

Explicit Formulas and Library of Images of Electromagnetic Fields for Anisotropic Materials

Valery Yakhno, Tatyana Yakhno and Mustafa Kasap

Dokuz Eylul University, 35100, Izmir, Turkey
{valery.yakhno, mustafa.kasap}@deu.edu.tr
yakhno@cs.deu.edu.tr

Abstract. An initial value problem for the dynamic Maxwell system with zero initial data and the point density of the electric current is considered. This problem describes the electromagnetic wave propagations in homogeneous nondispersive anisotropic materials. Explicit formula for the Fourier image of the solution of this problem is obtained by symbolic transformation methods in MATLAB. Using this formula and the numerical inverse Fourier transform the electromagnetic waves are simulated in different homogeneous, non-dispersive anisotropic materials. Images of wave propagations and animated movies were stored in a library which can be used for classification of the electromagnetic wave propagation in anisotropic materials.

1 Introduction

Mathematical modelling of anisotropic materials is an important component of scientific computing. Electromagnetic fields inside materials depend on properties of these materials. Modelling the wave propagation is based on the variety of methods and approaches. The most developed ones are based on numerical algorithms, for example, finite element methods. Advantages and disadvantages of these methods are well known [1, 2]. Some special cases turn out well and there exists models with explicit formulas describing the wave propagation. Usually these formulas are quite cumbersome and it is almost impossible to calculate them by hands.

In this case accumulated methods of symbolic algebra can help us a lot. Using symbolic transformations some parts of computational steps can be automated and moreover, the formulas can be in the form suitable for numerical algorithms later on.

In the present paper we are demonstrating an approach of modelling the electromagnetic wave propagation in crystals. Crystals are quite important class of anisotropic materials with "good" qualities: they are homogeneous and non-dispersive. For this class of materials we suggest an approach which integrates symbolic transformations of explicit formulas and numerical calculations. This approach has quite good computational performance and allows us to get high accuracy animations of wave propagations.

The paper is organized as follows. Section 2 contains a brief description of a mathematical model of the electromagnetic wave propagation, a state of a problem and steps of its solving. Section 3 describes main steps in MATLAB that were used for symbolic and numerical procedures of calculation. Section 4 describes how to interpret the result of experiments. The last section contains description of the library as a web application.

2 Mathematical Model, Problem and Scheme of its Solving

We consider the anisotropic dielectrics (crystals), which are homogeneous and non-dispersive. The propagation of electromagnetic waves in these media is described by Maxwell's system in which the dielectric permittivity can be given as a 3×3 matrix. Let $x = (x_1, x_2, x_3)$ be the space variable from R^3 , t be time variable from R , then Maxwell's equations are

$$(1) \quad \operatorname{curl}_x H = \epsilon \frac{\partial E}{\partial t} + j, \quad \operatorname{curl}_x E = -\mu \frac{\partial H}{\partial t},$$

$$(2) \quad \operatorname{div}_x(\epsilon E) = \rho, \quad \operatorname{div}_x(\mu H) = 0,$$

where $E = (E_1, E_2, E_3)$, $H = (H_1, H_2, H_3)$ are electric and magnetic intensity vectors, $E_k = E_k(x, t)$, $H_k = H_k(x, t)$, $k = 1, 2, 3$; $j = (j_1, j_2, j_3)$ is the density of the electric current, $j_k = j_k(x, t)$, $k = 1, 2, 3$; μ is the magnetic permeability, ϵ is the dielectric permittivity, ρ is the density of electric charges. We assume the conservation law of charges is given and $\mu = 1$, $\epsilon = (\epsilon_{ij})_{3 \times 3}$ is a symmetric positive definite matrix with constant elements.

Differentiating the first relation of (1) with respect to t and using the second equation of (1) we find

$$(3) \quad -\operatorname{curl}_x \operatorname{curl}_x E = \epsilon \frac{\partial^2 E}{\partial t^2} + \frac{\partial j}{\partial t}, \quad x \in R^3, t \in R.$$

We suppose that $j = -e\delta(x)\theta(t)$, where e is an arbitrary unit vector; $\delta(x) = \delta(x_1)\delta(x_2)\delta(x_3)$ is the Dirac delta function with the support at the point $x_1 = 0, x_2 = 0, x_3 = 0$; $\theta(t)$ is the Heaviside step function. The equation (3) is considered with the data

$$(4) \quad E|_{t < 0} = 0.$$

To find $E(x, t)$ which satisfies (3), (4) is the main problem of this section.

This problem can be written in the term of the Fourier image of $E(x, t)$ with respect to space variable x as follows

$$(5) \quad \epsilon \frac{\partial^2 \tilde{E}}{\partial t^2} + S(\nu)\tilde{E} = e\delta(t), \quad t \in R, \quad \nu \in R^3.$$

$$(6) \quad \tilde{E}|_{t < 0} = 0, \quad \nu \in R^3,$$

where

$$(7) \quad S(\nu) = \begin{pmatrix} \nu_2^2 + \nu_3^2 & -\nu_1\nu_2 & -\nu_1\nu_3 \\ -\nu_1\nu_2 & \nu_1^2 + \nu_3^2 & -\nu_2\nu_3 \\ -\nu_1\nu_3 & -\nu_2\nu_3 & \nu_1^2 + \nu_2^2 \end{pmatrix},$$

$$\tilde{E}(\nu, t) = \int_{R^3} E(x, t) e^{i\nu x} dx, \quad \nu = (\nu_1, \nu_2, \nu_3), \quad i^2 = -1.$$

The solution $\tilde{E}(\nu, t)$ of (5), (6) for $t > 0$ is given by formulas

$$(8) \quad \begin{aligned} \tilde{E}(\nu, t) &= T(\nu)Y(\nu, t), \\ Y(\nu, t) &= \text{column}(Y_1(\nu, t), Y_2(\nu, t), Y_3(\nu, t)), \end{aligned}$$

$$Y_n(\nu, t) = - \begin{cases} [T^T(\nu)e]_n (d_n(\nu))^{-\frac{1}{2}} \sin(\sqrt{d_n(\nu)}t) & , d_n(\nu) > 0 \\ [T^T(\nu)e]_n t & , d_n(\nu) = 0 \end{cases} \quad h = 1, 2, 3.$$

Here the matrix $T(\nu)$ and its transposed matrix $T^T(\nu)$ are non-singular and such that

$$(9) \quad T^T(\nu)\epsilon T(\nu) = I, \quad T^T(\nu)S(\nu)T(\nu) = D(\nu),$$

where I is the unit 3×3 matrix, $D(\nu) = \text{diag}(d_k(\nu))$, $k = 1, 2, 3$ is a diagonal matrix with non-negative elements.

Remark. We note that according to the matrix theory [3] for any symmetric positive definite ϵ and symmetric positive semi-definite $S(\nu)$ there exists matrices $T(\nu)$, $D(\nu)$ satisfying (9).

The first part of solving the problem (3), (4) is to find by symbolic transformations matrices $T(\nu)$, $T^T(\nu)$, $D(\nu) = \text{diag}(d_1(\nu), d_2(\nu), d_3(\nu))$ satisfying (9) if ϵ and $S(\nu)$ are given. This part of solving contains several steps. The first step is to find $\epsilon^{\frac{1}{2}}$ and $\epsilon^{-\frac{1}{2}}$ if ϵ is given symmetric positive definite. The given matrix ϵ is equivalent to a diagonal matrix of its eigenvalues, that is, there exists an orthogonal matrix P such that

$$P^T P = M \equiv \text{diag}(\mu_1, \mu_2, \mu_3), \quad \mu_k > 0, \quad k = 1, 2, 3; \quad P^T = P^{-1},$$

where P^T is the transposed matrix to P and P^{-1} is the inverse matrix to P . The matrix $M^{\frac{1}{2}}$ is defined by the rule $M^{\frac{1}{2}} = \text{diag}(\sqrt{\mu_1}, \sqrt{\mu_2}, \sqrt{\mu_3})$. The matrix $\epsilon^{\frac{1}{2}}$ is found by the formula $\epsilon^{\frac{1}{2}} = P M^{\frac{1}{2}} P^T$, and $\epsilon^{-\frac{1}{2}}$ is the inverse matrix to $\epsilon^{\frac{1}{2}}$.

The second step is to determine $D(\nu) = \text{diag}(d_1(\nu), d_2(\nu), d_3(\nu))$ if $\epsilon^{-\frac{1}{2}}$ is known and $S(\nu)$ is given by (7). For this step we consider the matrix $\epsilon^{-\frac{1}{2}} S \epsilon^{\frac{1}{2}}$ which is symmetric positive semidefinite. This matrix is congruent to a diagonal matrix of its eigenval-

ues, that is, there exists an orthogonal matrix $Q(\nu)$ such that $Q^T(\nu)[\epsilon^{-\frac{1}{2}}S(\nu)\epsilon^{-\frac{1}{2}}]Q(\nu) = \text{diag}(d_1(\nu), d_2(\nu), d_3(\nu))$, where $d_k(\nu) \geq 0$, $k = 1, 2, 3$; $Q^T(\nu) = Q^{-1}(\nu)$. We denote $D(\nu) = \text{diag}(d_1(\nu), d_2(\nu), d_3(\nu))$.

In the third step we define $T(\nu)$ as $T(\nu) = \epsilon^{-\frac{1}{2}}Q(\nu)$ and the matrix $T^T(\nu)$ is transposed to $T(\nu)$.

The second part of solving (3), (4) is to determine numerically the pre-image $E(x, t)$ if the image $\tilde{E}(\nu, t)$ is given by (8). The magnetic field $H(x, t)$ is found from the second equation (1) and data $H|_{t<0} = 0$ if $E(x, t)$ is known.

3 Computational Steps

MATLAB R13 [4] was used as an integrated environment for symbolic transformations, numeric computation and an animation of the solution obtained in Section 2.

The computation of the explicit presentation for the matrices $T(\nu)$, $T^T(\nu)$, $D(\nu)$ was realized by symbolic transformation in MATLAB. Obtained formulas for these matrices are cumbersome, they take several printed pages and it is almost impossible to calculate them by hands. As a result we got the explicit form of the formula (8) for the image $\tilde{E}(\nu, t)$ of the Fourier transform with the respect to space variables of the electric field $E(x, t)$.

In the next step we have to calculate 3-D inverse Fourier transform of \tilde{E} with respect to $\nu = (\nu_1, \nu_2, \nu_3)$. Because of the complexity of the explicit formulas for $T(\nu)$, $T^T(\nu)$, $D(\nu)$ containing three independent variables, the symbolic calculation of this inverse Fourier transform was unsuccessful. That was the reason why the numerical calculation of the inverse Fourier transform was realized on this step.

As the last step we generate animations for different ϵ values and collect the results in the library of images [5].

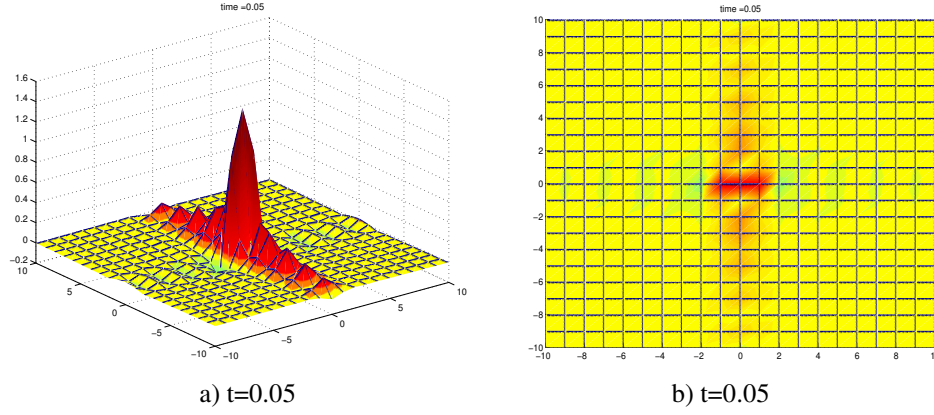
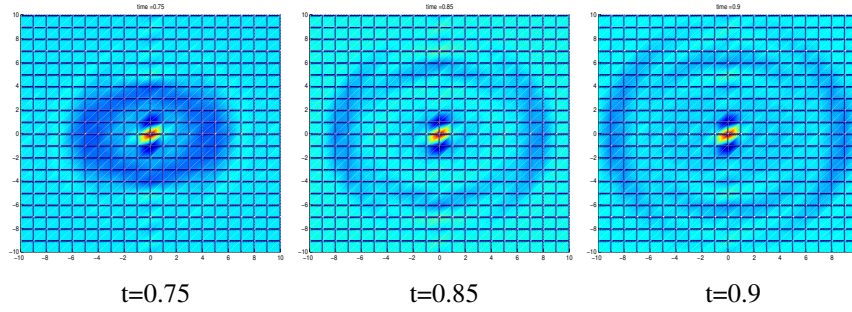
4 Examples of Simulations

In this section we present the images of the wave propagations in anisotropic dielectrics belonging to different crystal structures [6]. These pictures are obtained by fixing one of the space variables in the component $E_1(x, t)$ of the solution $E(x, t)$ of (3), (4). For experiments we take the current density in the form $j = e\delta(x)\theta(t)$, where $e = (1, 0, 0)$. This electric source is concentrated at the point $(0, 0, 0)$ and works permanently with the same power.

The figure 4.1 contains the visualization of the component $E_1(x_1, x_2, 1, t)$ of the solution $E(x, t)$ for $t \cong 0$. Figure 4.1.a is a 3-D graph of $E_1(x_1, x_2, 1, 0)$. The vertical axis is values of E_1 , the horizontal axes are x_1, x_2 . Figure 4.1.b is 2-D level plot of the same surface of $E_1(x_1, x_2, 1, 0)$. We note that graphs of $E_1(x_1, x_2, 1, 0)$ for different dielectrics are equal.

The figures 5.2 - 5.5 contain four screen shots of the wave propagations in dielectrics [7]:

- Mercurous Sulfides(HgS), $\epsilon = \text{diag}(18, 18, 32.5)$,

Fig. 4.1. Initial state $E_1(x_1, x_2, 1, 0)$ Fig. 5.2. Propagation in Mercurous Sulfide, $E_1(x_1, x_2, 1, t)$

- Strontium Niobate($Sr_2Nb_2O_7$), $\epsilon = \text{diag}(75, 46, 43)$,
- Magnesium Niobate($MgNb_2O_6$), $\epsilon = \text{diag}(16.4, 20.9, 32.4)$,
- Barium Peroxid (BaO_2), $\epsilon = \text{diag}(10.7, 10.7, 10.7)$.

These figures are 2-D level plots of $E_1(x_1, x_2, 1, t)$ for the different time.

5 Library of Wave Propagation Images and Animations

Generating images and animations of wave propagation is a time consuming operation. That is why we collected the created images and animations in the library for the reuse. In this library the different types of crystals were classified according to the types of anisotropy and for each of them the sequence of images and animated movies was created. This library can serve as a collection of patterns and samples when we analyze the structure of anisotropic materials or evaluate the performance of the numerical methods.

To make the library accessible from anywhere, Internet technologies are used to develop an application. The proposed structure (Figure 5.6) of the system consists of a web, a database, a media streaming server and a media encoding service. The web server is used to run a web application that will serve as a library. The database

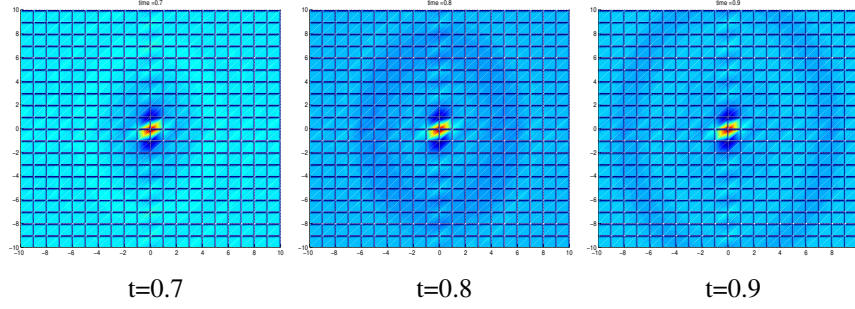


Fig. 5.3. Propagation in Strontium Niobate, $E_1(x_1, x_2, 1, t)$

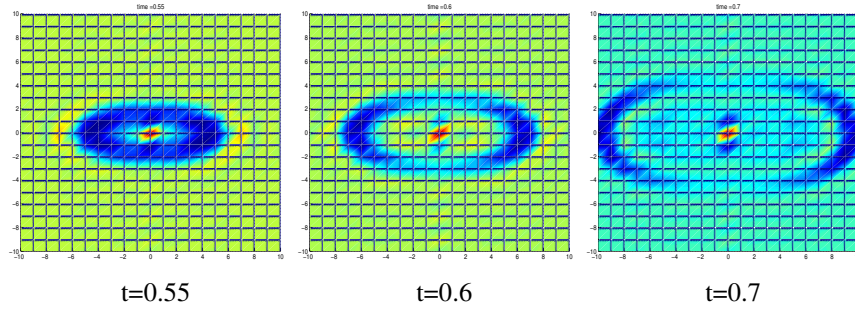


Fig. 5.4. Propagation in Magnesium Niobate, $E_1(x_1, x_2, 1, t)$

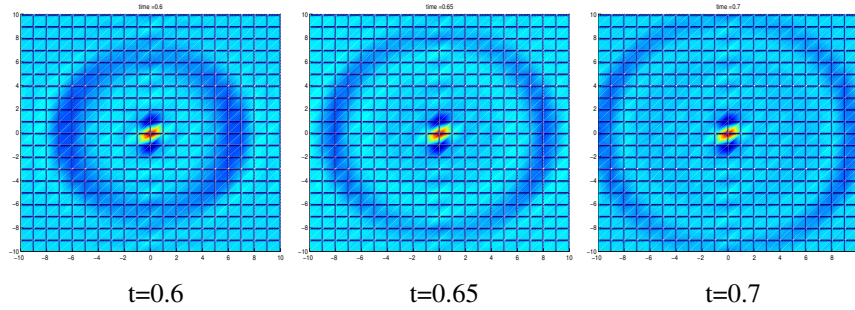


Fig. 5.5. Propagation in Barium Peroxid, $E_1(x_1, x_2, 1, t)$

server is used to store the information related to dielectrics. The media streaming server is used for streaming animation files over the Internet without considering connection, bandwidth problems. The media encoding service is used to convert animation files into the stream server file format.

The site map of the web application can be found on <http://mkasap.cs.deu.edu.tr/3DSim> and is defined on Figure 5.7. The library web application which starts with a selection page (experiments.htm) consists of

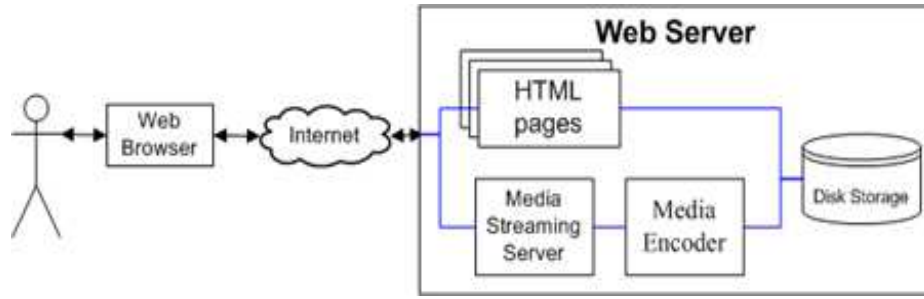


Fig. 5.6. System structure

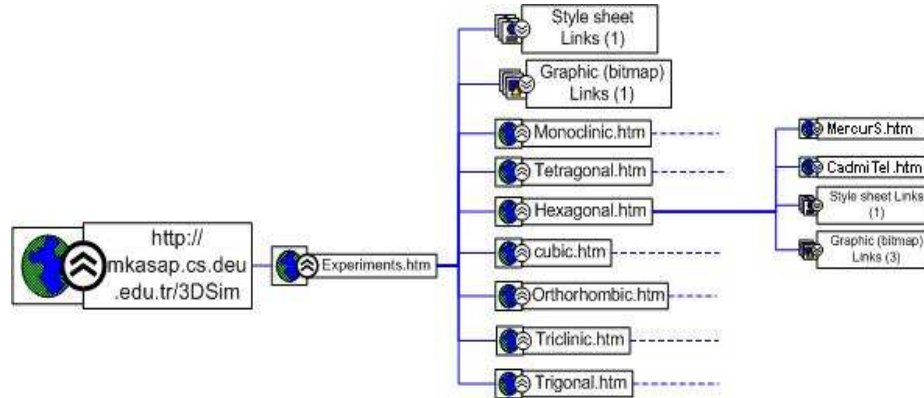


Fig. 5.7. Site map

seven well known types of crystals taking from crystallographic system [6]: Cubic, Triclinic, Tetragonal, Trigonal, Orthorhombic, Hexagonal, Monoclinic.

When a user selects one type, he will get a list of particular crystals, for example, Mercurous Sulfide, Cadmium Telluride etc. are listed under Hexagonal structure.

Selecting a crystal the user is coming to the page of the experiment results. On this page there are two types of visualization. One is a Top view, another is 3-D view and for both of them $E_1(x_1, x_2, 1, t)$ component and $H_1(x_1, x_2, 1, t)$ component are shown for different time values. In figure 5.8 a part of the web page is illustrated. Images for time series from 0.1 to 0.45 can be seen.

For animations there is no time series, its a single file. Structure of the animation window is illustrated on Figure 5.9.

To see how the propagation occurs, all images combined to form an animation file. In other words, all the images that were generated construct the frames of the animation file. Because the size of ordinary animation files is very large it is impossible to show them over Internet. In this point streaming technologies are used. Generated animation files are compressed and sent as a byte stream over the net. Compression operation is performed by media encoding services. Then encoded files are published by media

Animation			
Top View		3D View	
E Component	H Component	E Component	H Component
Animation	Animation	Animation	Animation
Images with Dielectric Constant defined above			
Top View		3D View	
E Component	H Component	E Component	H Component
Time 0.1	Time 0.1	Time 0.1	Time 0.1
Time 0.15	Time 0.15	Time 0.15	Time 0.15
Time 0.2	Time 0.2	Time 0.2	Time 0.2
Time 0.25	Time 0.25	Time 0.25	Time 0.25
Time 0.3	Time 0.3	Time 0.3	Time 0.3
Time 0.35	Time 0.35	Time 0.35	Time 0.35
Time 0.4	Time 0.4	Time 0.4	Time 0.4
Time 0.45	Time 0.45	Time 0.45	Time 0.45

Fig.5.8. Experiment result page.

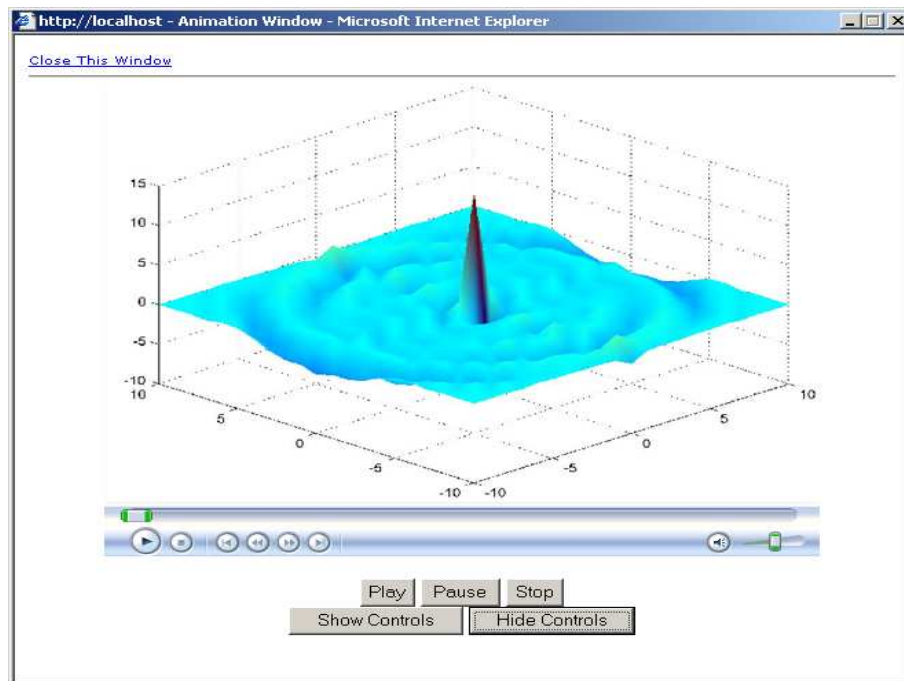


Fig.5.9. Animation window.

streaming server. By this way anyone with any type of Internet connection can watch the propagation animations without concern about connection problems.

Animation files are generated by MATLAB. The following code snippet is used to generate the animation files.

```
// Open a new image window, set measurement standards
fig = figure;
set(fig, 'PaperUnits', 'points');

// Set the size and position of the window to 800x600
// pixel located on 0, 0 screen coordinates
rect = get(gcf, 'Position');
rect(1) = 0; rect(2) = 0; rect(3) = 800; rect(4) = 600;
set(gcf, 'Position', rect);

// Create an AVI file with "CinePak" encoder, set the
// frame per second parameter
aviobj = avifile('Experiment01.avi'), ...
    'compression', 'CinePak', 'fps', 2);

// Perform following 4 lines for all images that will be
// used in the animation
A = imread('ImageX.jpg');
image(A);
frame = getframe(gcf);
aviobj = addframe(aviobj, frame);

// Close all opened resources
aviobj = close(aviobj);
close all;
```

Resulting animation file is encoded to a new format which is supported by Windows media server. Encoding operation is performed by Windows Media Encoder Service [8]. This tool converts the file into a digital media which continuously flows across the Internet.

6 Conclusion

Electromagnetic fields arising from the point sources were simulated by explicit formulas for generalized solutions of the Cauchy problem for Maxwell system for anisotropic crystals.

We note that the simulation electromagnetic fields based on explicit formulas is the best one. But unfortunately it is impossible to find explicit formulas for inhomogeneous anisotropic media. For this we need to construct approximate solutions by numerical procedures and methods and then simulate electromagnetic fields. All these procedures are time consuming and require heavy computations. That is why parallel implementation becomes one of the important issue for our future research.

7 Acknowledgement

This work has been partially supported by Dokuz Eylül University of Turkey under the research grant 03.KB.FEN.049.

References

1. Cohen G. C., Heikkola E., Joly P. Neittaan Maki P. *Mathematical and Numerical Aspects of Waves Propagation*, Springer Verlag, Berlin, 2003.
2. Monk P., *Finite Element Methods for Maxwell's Equations*, Clarendon Press, Oxford, 2003.
3. Goldberg J. L., *Matrix theory with applications*, McGraw-Hill International Editions, 1992.
4. MATLAB Version 6.5 Release 13, The Mathworks, Inc.
5. Yakhno V. G., Yakhno T. M., Kasap M. *Simulation of Electromagnetic Wave Propagation in Anisotropic Media.*, Selcuk Journal of Applied Mathematics, V.4, N.2, p.113-122, 2003.
6. Ramo S., Whinnery J. R., Duzer T., *Fields and waves in Communication electronics*, John Wiley and Sons, New York, 1994.
7. Young K.F., Frederikse H.P.R., J. Phys. Chem., Ref. Data, 2, 313, 1973.
8. Microsoft product documentation, September 2004,
<http://www.microsoft.com/windows/windowsmedia/default.aspx>.

Author Index

- Achenie, Luke E., III
Alcalde, Bernardo F. K., 83
Aranda, Joan, 153
Aversa, Rocco, III
- Baden, Scott B., III, 1
Bae, Su-Yun, 173
Bai, Zhaojun, III
Baik, Doo-Kwon, 115
Beckmann, Olav, 2
Bella, Gino, 33
Benkner, Siegfried, 3
Bhatt, Mehul, 43
Bojańczyk, Adam W., III
Brent, Richard P., II
Broeckhove, Jan, 204
- Cho, Sang-Young, 24
Choi, Daeseon, 13
Choi, Jin-Young, 105
Chung, Yoojin, 24
Claudio, Dalcidio M., 83
Corliss, George F., I
Cunha, José C., IV
- Dammann, Bernd, II
Di Martino, Beniamino, III
Dillon, Tharam, 43
Diverio, Tiarajú A., 83
Doerner, Karl F., 3
Dongarra, Jack, I, II
Duff, Iain, II
- Eun-Yeung Choi, 23
- Filippone, Salvatore, 33
Flahive, Andrew, 43
- Georgiev, Krassimir, IV
Gustavson, Fred G., II, III
- Hölbjg, Carlos A., 83
Ha, Keum-Sook, 69
HA, Kyeoung Ju, 79, 123
Hansen, Per Christian, II
Hartl, Richard F., 3
Hartmann, Wolfgang M., V
Hong, Keewan, 163
Hong, Kiyoo, 163
Hülsemann, Frank, V
- Jalili-Kharaajoo, Mahdi, 51
Jeon, Il-Soo, 91
Jeong, Dongwon, 115
Jin, Seunghun, 13
Jun, Yong-Kee, 69, 173
Jun-Cheol Jeon, 59
- Kalinin, Maxim O., 105
Kang, In-Hye, 105
Kang, Pil-Yong, 105
Kasap, Mustafa, 221
Kee-Young Yoo, 59
Kelly, Paul H J, 2
Kiechle, Guenter, 3
Kim, Chul-Hwan, 24
Kim, Dongkyoo, 163
Kim, Hyun-Sung, 23, 91, 98, 129, 136
Kim, Il-Gon, 105
Kim, Taihoon, V
Kim, Young-Gab, 115
Kim, Young-Shil, 115
Kowarschik, Markus, V
Kreinovich, Vladik, III
Krogh, Henrik, VI
Ku, Kyo Min, 79, 123
Kwon, Ho-Yeol, V
Kyo-Min Ku, 59
Kågström, Bo, II
- Lee, Jin-Ho, 129
Lee, Lee-Sub, 115
Lee, Sung-Woon, 98, 136
Lee, Won-Ho, 143
Li, Rencang, III
Lucka, Maria, 3
- Madsen, Henrik, II
Madsen, Kaj, I, III
Marès, Pere, 153
Martínez, Antonio B., 153
Messmer, Peter, IV
Morandi Jr., Paulo S., 83
- No, Bongnam, 163
- Park, Jongwoon, 163
Park, Mi-Young, 173
Park, Sanghun, 183
Park, So-Hee, 173
Probst, Kirsten, VI

Quinlan, Daniel, 193

Raghavan, Padma, IV

Rahayu, Wenny, 43

Rana, Omer F., IV

Reid, John K., II

Rossi, Nicola, 33

Roudsari, F. Habibipour, 51

Sadri, Mohammadreza, 51

Scholtes, Carsten, 194

Sloot, Peter, II

Somerville, Paul N., V

Sosonkina, Masha, IV

Taniar, David, 43

Thøgersen, Dorthé, VI

Toledo, Sivan, IV

Ubertini, Stefano, 33

Vanmechelen, Broeckhove, 204

Walther, Andrea, I

Waśniewski, Jerzy, I, III

Wouters, Carlo, 43

Wu, Chao-Chin, 211

Yakhno, Tatyana, 221

Yakhno, Valery, 221

Yang, Laurence Tianruo, III

Yi, Wan S., 105

Yoo, Kee Young, 79, 123

Yoo, Kee-Young, 23, 69, 136, 143

Yoon, Hyunsoo, 13

Zegzhda, Dmitry P., 105

Zegzhda, Peter D., 105

Zlatev, Zahari, III, IV