Proceedings

Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies

PDCAT 2008

Proceedings

Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies

Dunedin, New Zealand
1–4 December, 2008

Editors
Zhiyi Huang, Zhiwei Xu, Nathan Rountree, Laurent Lefevre,
Hong Shen, John Hine, and Yi Pan

Sponsors
KAREN — The Kiwi Advanced Research and Education Network
University of Otago
World 45

CPS Conference Publishing Services

Los Alamitos, California
Washington • Tokyo

IEEE computer society

IEEE Computer Society Order Number P3443
BMS Part Number CFP08536-PRT
ISBN 978-0-7695-3443-5
Library of Congress Number 2008934987

Additional copies may be ordered from:

IEEE Computer Society	IEEE Service Center	IEEE Computer Society
Customer Service Center	445 Hoes Lane	Asia/Pacific Office
10662 Los Vaqueros Circle	P.O. Box 1331	Watanabe Bldg., 1-4-2
P.O. Box 3014	Piscataway, NJ 08855-1331	Minami-Aoyama
Los Alamitos, CA 90720-1314	Tel: + 1 732 981 0060	Minato-ku, Tokyo 107-0062
Tel: + 1 800 272 6657	Fax: + 1 732 981 9667	JAPAN
Fax: + 1 714 821 4641	http://shop.ieee.org/store/	Tel: + 81 3 3408 3118
http://computer.org/cspress	customer-service@ieee.org	Fax: + 81 3 3408 3553
csbooks@computer.org		tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Lisa O'Conner
Cover art production by Alex Torres
Printed in the United States of America by The Printing House

IEEE computer society

CPS Conference Publishing Services

IEEE Computer Society
***Conference Publishing Services* (CPS)**
http://www.computer.org/cps

Table of Contents

PDCAT 2008

PDCAT 2008 Workshops

Message from the General Chairs

Welcome to PDCAT'08, the ninth International Conference on Parallel and Distributed Computing, Applications and Technologies. And welcome to New Zealand, to Dunedin and to the University of Otago, your hosts for this year's conference. It is appropriate that PDCAT come to New Zealand for the first time this year as the country develops its capabilities with KAREN, the Kiwi Advanced Research and Education Network.

Dunedin is New Zealand's oldest city. It was first settled, largely by Scots, about 150 years ago and is known as Edinburgh of the South. Do take some time to explore Dunedin and at least the nearby areas of Otago. For example the Taieri Gorge Railway departs from Dunedin Rail Station and the Royal Albatross Colony is an easy drive or tour from Dunedin city.

Your local Organizing Committee and Programme Committee have put together an exciting week here in Dunedin. The programme holds much of interest and for the first time the conference is supported by four specialized workshops. Of course, there are also those social occasions that are so important to networking with your colleagues.

The workshops are an innovation for PDCAT this year and we hope they will set a successful precedent for future years. The workshops range from the 3rd Annual New Zealand Workshop in High Performance and Grid Computing to workshops with overseas organisers such as the 1st International Workshop on High Performance Data Grids. We wish the workshop organisers all the best.

A conference such as PDCAT is not possible without the efforts of many people. Obviously the authors, the programme committee and the reviewers are essential to the success of any conference. Our particular thanks go to Zhiyi Huang and Zhiwei Xu as programme chairs for assembling the programme and proceedings. We also thank the local organisers for all the work that goes into making a conference such as PDCAT a success. Finally we thank our sponsors, KAREN, World45 and the University of Otago for their support.

We hope you will find PDCAT'08 and your visit to Dunedin both exciting and enjoyable.

John Hine
Yi Pan
General Chairs

Message from the Programme Committee Chairs

Welcome to participation in PDCAT'08, the Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, held in Dunedin, New Zealand during December 1-4, 2008. Over the years, PDCAT has become a major forum for scientists, engineers, and practitioners throughout the world to present their latest research, results, ideas, developments and applications in all areas of parallel and distributed computing.

This year we received 110 submissions from 22 different countries and areas across the world. Out of the 110 submissions, we have only accepted 39 regular papers and 19 short papers. The acceptance rate for regular papers is 35%. The task of paper selection was a major effort that involved more than 70 members of the Programme Committee and 36 external reviewers. Such an effort enabled us to have a thorough review process, and ensured a high-quality programme. With an average of 2.97 reviews per paper, extensive feedback was provided to all who submitted.

The programme consists of sessions ranging over Grid Computing Systems, Parallel/Distributed Algorithms, Parallel/Distributed Architecture, High Performance Computing, Interconnection Networks, Fault-tolerance and Reliability, Task Scheduling and Resource Allocation, to Sensor Networks, Computer Networks, Intelligent Computing, and Formal Methods and Programming Languages. It also covers hot topics such as Power-aware Computing.

Along with the sessions above, an innovation this year is to run four workshops: Sensor Networks and Ambient Intelligence, High Performance Data Grids, Programming Parallel Applications for Multi-cores, and the New Zealand Workshop on High Performance and Grid Computing.

Needless to say, the success of a large conference such as PDCAT is impossible without the involvement and support of a lot of people. First, we would like to thank the PC members, external reviewers, and the workshop organisers for their hard work. Without their effort, the planned programme would be impossible. We also would like to thank our international advisory team that includes Hong Shen, John Hine, Yi Pan, Tony McGregor, David Munro, Krys Pawlikowski, Laurent Lefevre, and Albert Y. Zomaya. Last but not least, we are indebted to a dedicated team from the University of Otago, which includes Paul Werstein, Nathan Rountree, Cameron Kerr, Kaye Saunders, Dave Robertson, and Bronwyn Atterbury.

Finally, we hope all participants will enjoy the programme as well as Dunedin and New Zealand. Wishing you all a blessed period of time at PDCAT'08!

Zhiyi Huang and Zhiwei Xu
PDCAT'08 Programme Chairs

Conference Organization

Program Committee Members

Hamid Arabnia, *University of Georgia, USA*
Sasitharan Balasubramaniam, *Waterford Institute of Technology, Ireland*
Maria Elena Villapol Blanco, *Central University of Venezuela, Venezuela*
Peter Cappello, *University of California, Santa Barbara, USA*
Xiaojun Cao, *Georgia State University, USA*
Stuart Charters, *Lincoln University, New Zealand*
Wenguang Chen, *Tsinghua University, China*
Xueqi Cheng, *Chinese Academy of Sciences, China*
Francis Chin, *University of Hong Kong, China*
Tony Dale, *Canterbury University, New Zealand*
Noria Foukia, *University of Otago, New Zealand*
Steven Gordon, *Thammasat University, Thailand*
Teofilo Gonzalez, *University of California, Santa Barbara, USA*
Minyi Guo, *University of Aizu, Japan*
Aaron Harwood, *University of Melbourne, Australia*
Syed Faisal Hasan, *Dhaka University, Bangladesh*
Susumu Horiguchi, *Tohoku University, Japan*
Cruz Izu, *University of Adelaide, Australia*
Brendan Jennings, *Waterford Institute of Technology, Ireland*
Hai Jin, *Huazhong University of Sci. & Tech., China*
Slava Kitaeff, *Auckland University of Technology*
Lars Kristensen, *University of Aarhus, Denmark*
Francis Lau, *University of Hong Kong, Hong Kong*
SingLing Lee, *National Chung Cheng University, Taiwan*
Yamin Li, *Hosei University, Japan*
Yingshu Li, *Georgia State University, USA*
Weifa Liang, *Australian Nat. University, Australia*
Chuang Lin, *Tsinghua University, China*
Zhaobin Liu, *Dalian Maritime University, China*
Koji Nakano*, Hiroshima University, Japan*
Mariusz Nowostawski, *World45 Ltd., New Zealand*
Anne Hee Hiong Ngu, *Texas State University, USA*
Marcin Paprzycki, *Polish Academy of Sciences, Poland*
Shietung Peng, *Hosei University, Japan*
Marius Portmann, *University of Queensland, Australia*
Elizabeth Post, *Lincoln University, New Zealand*
Omer Rana, *Cardiff University, UK*
Ricky Robinson, *NICTA, Australia*
Frode Eika Sandnes, *Oslo University College, Norway*
Micheal Sheng, *University of Adelaide, Australia*
Oliver Sinnen, *University of Auckland, New Zealand*
Leonel Sousa, *Technical University of Lisbon, Portugal*
Yutaka Takahashi, *Kyoto University, Japan*
Rodney Topor, *Griffith University, Australia*
Andrew Trotman, *University of Otago, New Zealand*
Sven Ubik, *CESNET, Czech Republic*
Andrew Wendelborn, *University of Adelaide, Australia*
Ryan Wishart, *NICTA, Australia*
Jozef Wozniak, *Gdansk University of Technology, Poland*
Nong Xiao, *University of Sci &Tec of Def, China*
Chao-Tung Yang, *Tunghai University, Taiwan*
Laurence T. Yang, *St Francis Xavier University, Canada*
Xinfeng Ye, *Auckland University, New Zealand*
Peinan Zhang, *Intel China Research, China*

External Reviewers
Samuel Antão, *Portugal*
Morteza Biglari-Abhari, *New Zealand*
Ravikesh Chandra, *New Zealand*
Jamie Curtis, *New Zealand*
Ales Friedl, *Czech Republic*
Nasser Giacaman, *New Zealand*
Krzysztof Gierlowski, *Poland*
Lindsay Groves, *New Zealand*
Tomas Hruby, *New Zealand*
Qihang Huang, *New Zealand*
Liu Jia, *China*
Jerzy Konorski, *Poland*
Adrian Kosowski, *Poland*
Xitong Li, *Australia*
Simone Ludwig, *United Kingdom*
Svetislav Momcilovic, *Portugal*
Lei Ni, *Australia*
Krzysztof Nowicki, *Poland*
Rodion Podoronzy, *USA*
Frederico Pratas, *Portugal*
Apan Qasem, *USA*
Imran Rao, *Australia*
Nathan Rountree, *New Zealand*
Hong Shen, *Australia*
Zhiyuan Shao, *China*
Yao Shen, *Japan*
Xuanhua Shi, *China*
Martin Simek, *Czech Republic*
Vladimir Smotlacha, *Czech Republic*
Paul Werstein, *New Zealand*
Yanbo Wu, *Australia*
Shihong Xu, *Australia*
Petr Zejdl, *Czech Republic*
Daqiang Zhang, *Japan*
Ran Zheng, *China*
Deqing Zou, *China*

2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies

Multi-core Defense System (MSDS) for Protecting Computer Infrastructure against DDoS attacks

Ashley Chonka, *Member, IEEE,* Soon Keow Chong, and Wanlei Zhou, *Member, IEEE*,
School of Engineering & Information Technology
Deakin University
Geelong, 3220, Australia

Yang Xiang, *Member, IEEE*
School of Management and Information Systems
Central Queensland University
Rockhampton, 4702, Australia

{ashley, skchon,wanlei}@deakin.edu.au and y.xiang@cqu.edu.au

Abstract

Distributed Denial of Service attacks is one of the most challenging areas to deal with in Security. Not only do security managers have to deal with flood and vulnerability attacks. They also have to consider whether they are from legitimate or malicious attackers. In our previous work we developed a framework called bodyguard, which is to help security software developers from the current serialized paradigm, to a multi-core paradigm. In this paper, we update our research work by moving our bodyguard paradigm, into our new Ubiquitous Multi-Core Framework. From this shift, we show a marked improvement from our previous result of 20% to 110% speedup performance with an average cost of 1.5ms. We also conducted a second series of experiments, which we trained up Neural Network, and tested it against actual DDoS attack traffic. From these experiments, we were able to achieve an average of 93.36%, of this attack traffic.

Index Terms — Multicore, Ubiquitous Multicore framework, Farmer, Bodyguard Framework

1. Introduction

Today's internet has evolved into high-speed backbones and local-wide area networks, which link millions of end-users to many critical services. Majority of today's businesses rely upon these critical services to function at full capacity, so that they can achieve a greater customer base and profits. A DDoS (Distributed Denial of Service) attack puts these critical systems under the series threat of collapse and loss for these businesses.

The two major challenges in defending against these attacks, is to firstly have a defense system that has detection that is sensitive and accurate, while at the same time filtering and monitoring the defense system. Unfortunately, most defense systems, such as traceback [1][2], logging [3][4] and messaging [5][6] are just not sensitive enough to be able separate out legitimate traffic from attack. Another major problem with these defense systems is that, they themselves are usually susceptible to the same DDoS attacks, that they are trying defending against [7].

In our previous paper [8] we introduced a defense system called Farmer, after the Kevin Costner Movie 'Bodyguard'. Farmer was built and developed based on our Bodyguard Framework. This Framework is an abstract paradigm, which groups class of applications based on what functions they provide to the system (Security related or Multi-media related). Once these applications are grouped they are then placed own prospective core process, within a Multi-Core system. For example, with Farmer, we separated out different parts of security procedures (IP reconstruction, filter attack traffic, monitoring farmer) and placed them on separate core processors within an Intel Quad-Core system. In our former results, we achieved an overall speedup performance increase of 20% for our defense system. We also gained a number of advantages by applying this framework, which are, firstly the ability of our defense system to defend itself against an attack in real-time, the ability to record and analysis traffic almost simultaneously, protect the defense system by having its own redundant defense system, monitor and troubleshoot the system if problems arise.

In this paper we discuss current updates with our bodyguard framework, now call Ubiquitous Multicore Framework (UM), and continue our experimentation of the system. Section Two briefly covers the related work done in Multicore. The details of UM framework and how it is applied to the bodyguard framework Section Three. Section Four presents the experiments and evaluation that were conducted on our system. Lastly, Section Five covers the conclusion and future work.

2. Related Work

In this section, we discuss very briefly multicore and multimedia, and the two areas where our multicore framework has been applied.

2.1. Multicore and Multimedia

Multi-core systems can be defined as a system that has two or more processing cores integrated into a single chip [11][12][13]. Through this design, each processing cores has their own private cache (L1) and a shared common cache (L2). The shared cache and main memory share the bandwidth between all the processing cores. Multimedia co-processor interface was developed by [14], in which they used a multicore system to offload task management jobs from MPU or DSP. From their evaluations conducted on a JPEG file, Ou et al. achieved an overall performance increase of 57%, while they kept their overhead to 1.56% of the DSP core. In comparison with our UM framework [10], our framework is more abstract, by applying applications (not separate sections of a file) to separate core processors.

2.2. Multi-classifier SPAM filter

To follow up on [8], we then applied our multicore framework to a multi-classifier SPAM filter [9]. We found that if you ran each classifier process in parallel with each other, it greatly improved the performance of our multi-classifier architecture, in the areas of false positives reduction and increase accuracy. Further,

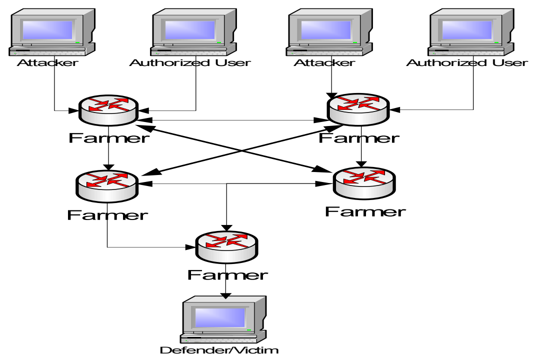

Figure 1. System Architecture of Farmer

advantages that our multicore framework provided, is as follows:

- Reduced computation burden of the overall mail server.
- Reduced memory storage, email messages are processed independently from other classifiers.
- When one of the classifiers becomes idle it will directly go into training mode, thereby optimizing resource usage.
- Is robust as the adaptive selection can still provide accurate email classification if one of the core fails.

3. Background

3.1 Farmer System Design

The bodyguard framework is distributed on each router in the network; in order to provide overall protection (Figure 1). Each Bodyguard is a source end (provides security before traffic leaves the router) and destination end protector (provides security as the traffic enters the network). Another feature in Figure 1 is that each bodyguard is connected to each other. There are three main reasons for this; to allow bodyguards to send updated security information to each other (new attacks that each has encountered, for example), send security information down to the next hop for checking application data as it comes into the router (This is to provide better performance, by breaking up the security and application data), monitors the performance of each other (So if a successful attack brings down a bodyguard, the next hop router is prepared to handle the security). Farmer includes the two parts of the bodyguard framework, the side bodyguard (SB) and front bodyguard (FB) (Figure 2). The side bodyguard is the main component of the framework, is to protect the system, while allowing application/s to run at full performance potential.

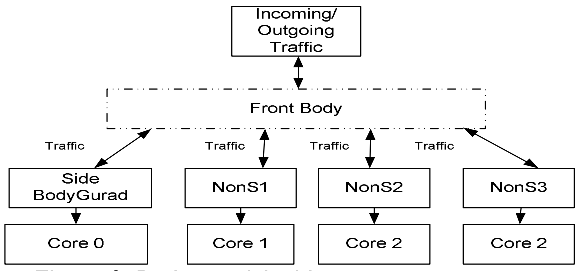

Figure 2. Bodyguard Architecture

3.2 Ubiquitous Multicore (UM) Framework

The Ubiquitous Multicore Framework is built from a divide-and-conquer approach [15], by dividing our applications into specifics classes and places them on separate core processors (Figure 3). Each application will run in parallel with each other, and exchange information when necessary. The application core assigner (ACA), assigns the class applications either on behalf of the user, or the user can select from the core(s) that are available. Once each application is assigned to a core, depending on the application program, a number of jobs or threads can then be executed on this core processor.

3.3 Applied UM Algorithm to Bodyguard Framework

In this paper, we further our research development by updating our bodyguard framework by incorporating it into our UM algorithm. We also include a mathematical partition model (MPM), that we adapted and modified from [19][20], so that we can evaluate our algorithm. The MPM, is to used to give a clearer picture of how the bodyguard framework is partitioned, on a multi-core system. But just separating and assigning our bodyguard tells us nothing about the speedup performance, if any is achieved, or what the overhead costs in terms of this partition are. So we have also included formulas (4,5,6) to accomplish this.

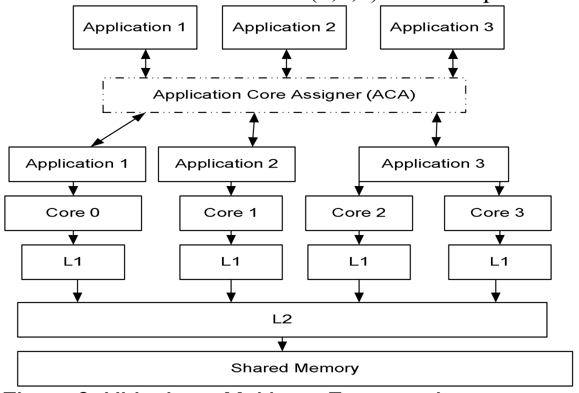

Figure 3. Ubiquitous Multicore Framework

3.4 Mathematical Partition Model

The MPM is adapted and modified from the partition analysis of [19][20], in which they analysed the speedup performance, computation and communication cost and execution times of their partition. To partition the application correctly we use three phases, communication, computation and communication.

Phase 1:

$$t_{comm} = (p-1)(t_{stup} + t_d) \tag{1}$$

Phase 2:

$$t_{comp} \leq \frac{mp*n}{p-1} \tag{2}$$

Phase 3:

$$t_{comm} = u(t_{stup} + vt_d) \tag{3}$$

In order to maintain the highest speedup and computation/communication ratio we use the Overall Execution Time(4), Speedup factor (5), C/C ration (6):

$$t_p \leq \frac{mp*n}{p-1} + (p-1)(t_{stup} + t_d) + k \tag{4}$$

$$\frac{t_s}{t_p} = \frac{mp*n}{\frac{mp*n}{p-1} + (p-1)(t_{stup} + t_d) + k} \tag{5}$$

$$\frac{mp*n}{(p-1)((p-1)(t_{stup} + t_d) + k)} \tag{6}$$

4. Performance Evaluation

4.1 Performance Analysis

To assess the performance of our multicore system, we compared the two kernel benchmarks. The hardware on the multicore system had Intel Core 2 Quad Q6600 2.4GHz Quad Core Processor, 2 GB of RAM and 2 300GB SATA hard-drives. The kernel

```
Double PI, H, sum, x, f,a
Call MPI_INIT
Call MPI_COMM_RANK
Call MPI_COMM_SIZE

Enter number of processors
Enter number of intervals
Send intervals to the
number of processors
Calculate Interval Size
Send Back Interval Results
Put Intervals Results back
together
Print Results
```

Figure 4. Pseudo Code for MPI "Perfect" parallel program.

under measurement was 2.6.22.14.72 fc6. To gather computational data, we included timers with our application, in order to record execution times. Communication time is depended upon the number of messages, the size of the message and the interconnection speed. We have decided to set the standard to 1ms and computational data is assumed to be .1ms less then execution time.

4.2. Simulation Setup

4.2.1 Benchmark factors

Once we have the execution times t_s, computational time t_{comp}, and communication time t_{com}, we can establish the speedup factor (formula 7) and computation/communication ratio (formula 8).

$$\frac{t_s}{t_{cp}} = \frac{t_s}{t_{comp} + t_{com}} \qquad (7)$$

Where t_s will stand for execution time on a single core processor (t_{cp}), this includes computation time and communication time.

$$\frac{t_{comp}}{t_{com}} \qquad (8)$$

Apart from speedup and the Computation and Communication ratios, we also evaluate the UM algorithm, through the use of Time Complexity or "big-oh", also referred to as "order of magnitude" [12]

$$f(x) = O(g(x))$$
$$[0 \le f(x) \le cg(x)] \text{ for all } x \ge 0 \qquad (9)$$

Where f(x) and g(x) are functions of x. A positive constant, c, has to exist for all $x \ge x_0$ otherwise it is zero.

	Core 1	Core 2	Core 3	Core 4
Exe Time	1.5ms	1.4ms	1.3ms	1.4ms
Comp Time	.3ms	.3ms	.2ms	.3ms
Comm Time	1ms	1ms	1ms	1ms
Speed Ratio	115%	108%	108%	108%
C/C	0.3	0.3	0.2	0.3
Time Complex	3.5	3.5	3.5	3.5
Cost	1.5	1.4	1.3	1.4
Cost-Optimal	3.7	3.7	3.7	3.7

Table 1. Results of speedup and the costs, which show an average increase of 110% at the average cost of 1.4ms

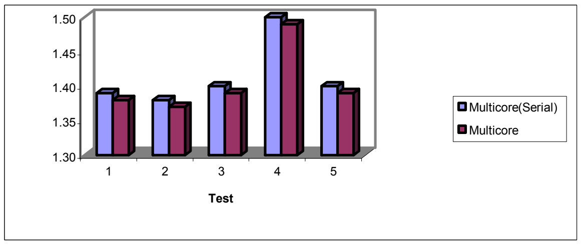

Figure 5. Resource Sharing delay time for shared memory (latency)

$$t_{cp}+t_{com}=(n/cp+1)+(2t_{stup}+(n/cp+1)t_{md} \qquad (10)$$

Where n is the number of threads on each core processor. The last benchmark we will use is the cost and cost-optimal.

Cost = (execution time) * (total number of processor used)

Cost Optimal = time complexity * number of processor = (n log n

MPI	**Core 1**	**Core 2**	**Core 3**	**Core 4**
Exe Time	1.5ms	1.4ms	1.3ms	1.4ms
Speed Ratio	115%	108%	108%	108%
C/C	0.5	1.8	0.3	1.8
Cost	1.5	1.4	1.3	1.4
MultiC	**Core 1**	**Core 2**	**Core 3**	**Core 4**
Exe Time	1.35ms	1.36ms	1.35ms	1.35ms
Speed Ratio	101%	101%	101%	101%
C/C	0.3	0.3	0.2	0.3
Cost	1.5	1.4	1.3	1.4

Table 2. Results of speedup and the costs, which show an average increase of 105% at the average cost of 1.4ms for the MPI over our previous Multicore result.

4.3. Experimentation

To give us a baseline of comparison, we wrote a program using MPI (19) [See Figure 4], in which the program gives us a "perfect" example of parallel programming. The results show [table 1] that we achieved a speedup result of 110% across of bodyguard application. From table 1 we then do a comparison of previous results from paper [8], in which we selected the best of our results (Figure 5, Test 2), and show them along side table 1 (See Table 2). To make the comparison fair, we used the same computation and communication time from the MPI program, for our multicore program. What the results from Table 2 shows is that our MPI program use's multicore technology with a greater efficiency then our multi-core program.

MPI (MC)	**Core 1**	**Core 2**	**Core 3**	**Core 4**
Exe Time	1.10ms	1.15ms	1.11ms	1.11ms
Comp Time	0ms	.04ms	.01ms	.01ms
Comm Time	1ms	1ms	1ms	1ms
Speed Ratio	110%	111%	110%	110%
C/C	0.3	0.3	0.2	0.3
Cost	1.5	1.4	1.3	1.4

Table 3. Results of speedup and the costs, which show an average increase of 110% at the average cost of 1.4ms for the MPI over our previous Multicore result.

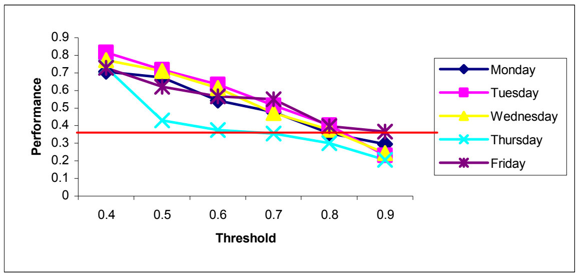

Figure 6. Training Results of our Neural Network archived a better than average (94%) result, with an average of 6 false positives per test (5 days of tests were conducted from the MIT Dataset).

But the reason for this greater efficiency was due to the fact that we wrote our multicore program in C++ only. Thereby, we used MPI in our multi-core program to get the following result in table 3. As we can see, our speed up ratio increase from 101% average to 110% with the use of MPI. As we see from our previous work, we only achieved a 20% increase, but the experiments we conducted were quite different (see table 4). In S-Core results, we just allowed the program to be assigned by the Linux Kernal, in M-Core we assigned the programs using affinity methods in C.

In our second experiment we trained up Farmers side bodyguard, which contains our Back Propagation Neural Network Filter (placed on core 2) to detect and filter DDoS attack traffic. In order to train up our Neural Network we used dataset from the week 2, 1998 DARPA intrusion detection evaluation set at Lincoln

System	T1	T2	T3	T4	T5	Total
S-Core	150	153	150	151	151	150
M-Core	130	133	129	133	132	130
Speedup	20	20	21	18	19	20

Table 4. Speedup Comparison between Serial Multicore and Multicore

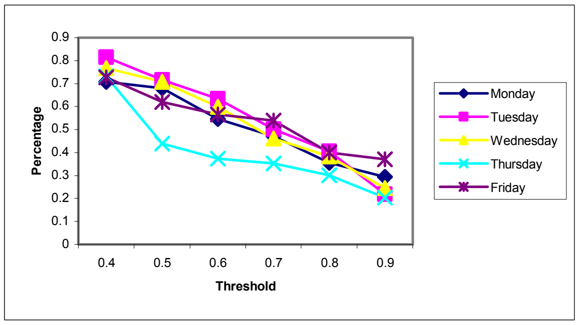

Figure 7. Average of 75% was achieved from our training results for detecting Legitimate Traffic.

	Legitimate Traffic [Best]	Attack Traffic [Best]	Legitimate Traffic [Worse]	Attack Traffic [Worse
Mon	90.0%	91.25%	75.20%	86.87%
Tues	92.35%	93.37%	74.37%	85.90%
Wed	95.62%	94.09%	71.02%	84.90%
Thur	95.40%	94.01%	72.25%	85.52%
Fri	95.43%	94.10%	71.17%	83.65%

Table 5. Test Results from our Neural Network, showing the best and worse achieved results.

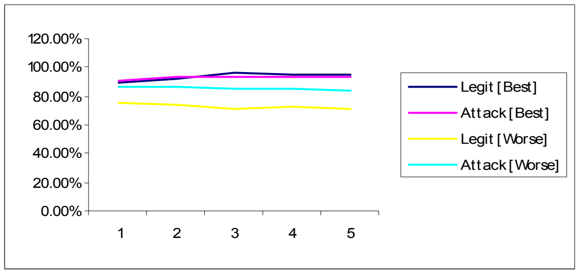

Figure 8. Test Results from our Neural Network

Laboratory, MIT [17]. The data sets from MIT come in TCP dump format, so we extracted the features we needed and insert them into a MySQL database. These features included SrcIP, DestIP, SrcPort, DestPort and. the length of time. We added an extra field to the table for the decision, 0 for legitimate and 1 for illegitimate.

Our results shown in Figures 6 and 7, that we were able to achieve a +90% of the known attack traffic, while allowing 75% of legitimate traffic, with an average of 6 false positives per test. This means that our security detection is quite sensitive in detecting and filtering out DDoS attack traffic. To confirm this result we then further our experiment by testing our Neural Network against the test data provide by [17]. In Table 5 and Figure 8 shows, we achieved with our Neural Network an average 93.76% for our best result for detecting legitimate traffic, while maintaining average 93.36% in detecting attack traffic. So these results shows that our Neural Network is fairly sensitive and effective with these types of attacks. This claim is further backed up by the “worst” results, that even if

our Neural Network is having a "bad" (so to speak), it can detect an average of 72.80% legitimate and 85.37% attack, which is still fairly good against DDoS attack.
Further analysis of why we achieved different results with our Neural Network is the way that you have to 'tune' the training of the Neural Network. You do this by changing a number of characteristics such as Learning Rate, Momentum and Threshold. By changing the Learning Rate, for example, you alter how the Neural Network learns. This then affects the results that are outputted, we selected for the 'best' results a Learning Rate of 0.2, Momentum 0.6, and Threshold of
0.4. For the worse results we set the Learning Rate at 0.2, Momentum 0.3, and Threshold 0.7.

5. Conclusion

In this paper, we further extended upon our previous work within multicore defense system, by applying the UM Framework to our Bodyguard Defense System. The goal of such a security system is to use the new multicore machines that are coming out, but also, with these machines they can be used to solve some of the many problems of computer security. Based on the results we have showed our defense system is improved from 110% speedup, through the use of MPI [19]. We, also, showed our test results of Farmer's side bodyguard (Back Propagation Neural Network), which would than tell the forward bodyguard to filter the attack traffic detected. The results show, based on 10 tests that we conducted over 4 hours of training the system, we got an average of 94% of attack traffic detected with an average of 6 false positives per test that we conducted.

6. References

[1] Savage, S., Wetherall, D., Karlin, A., and Anderson, T., (2001), '*Practical Network Support for IP Traceback*'**,** SIGCOMM'00, Stockholm, Sweden, 2000

[2] Belenky, A.,and Ansari, N., '*Tracing Multiple Attackers with Deterministic Packet Marking (DPM)'*, Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing

[3] Snoeren, A.C., et al., (2002), "Single-Packet IP Traceback," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, 2002, pp. 721–734.

[4] Baba, T., and Matsuda, S., (2002). "Tracing Network Attacks to Their Sources," *IEEE Internet Computing*, vol. 6, no. 3, 2002

[5] Bellovin, S., Leech, M., and Taylor, T., (2003), 'ICMP Traceback Messages,' Internet Draft, Internet Eng. Task Force, 2003; work in progress.

[6] Mankin, A., Massey, D., Wu, C.L., Wu S.F and Zhang, L., (2001), "On Design and Evaluation of 'Intention-Driven' ICMP Traceback," *Proc. IEEE Int'l Conf. Computer Comm. and Networks*, IEEE CS Press, 2001. pp. 159–165.

[7] Aljifri, M., (2003), '*IP Traceback: A NewDenial-of-Service Deterrent?*' Published By The IEEE Computer Society 1540-7993/03 2003

[8] Chonka, A Zhou, W Knapp, K and Xiang, Y, (2008), "Protecting Information Systems from DDoS Attack Using Multicore Methodology", *IEEE 8th International Conference on Computer and Information Technology*, IEEE, 2008.

[9] Islam, R. M.D, Singh, J, Zhou, W., and Chonka, A., (2008) , "Multi-Classifier Classification of Spam Email on a Multicore Architecture", Proceedings of IFIP International Conference on Network and Parallel Computing, 2008 [Accepted]

[10] Chonka, A, Zhou, W, and Ngo, L, (2008), "Ubiquitous Multicore (UM) Methodology for Multimedia, Proceeding of International Symposium on Computer Science and its Applications

[11] Multi-Core from Intel – Products and Platforms. http: //www.intel.com/multi- core/products.htm, 2006.

[12]AMD Multi-Core Products. http://multicore.amd.com/en/Products/, 2006.

[13] Gorder, P.M, (2007), *'Multicore processors for science and engineering*', IEEE CS and the AIP, 1521-9615/07/,March/April 2007

[14] Ou, S.H., Lin, T.J., Deng, X.S., Zhuo, Z.H., Liu, C.W., (2008), "Multithreaded coprocessor interface for multi-core multimedia SoC', Proceedings of the 2008 conference on Asia and South Pacific design automation, Seoul, Korea SESSION: University LSI design contest, Pages 115-116, ISBN:978-1-4244-1922-7, 2008

[15] JaJa, J. (1992), '*An Introduction to Parallel Algorithms*", Addison Wesley, Reading, MA

[16]] MIT 1998 DARPA Intrusion Detection Evaluation Data Set, http://www.ll.mit.edu/mission/communications/ist/index.htm l[17] Xiang, Y., and Zhou, W., (2004), 'Trace IP packets by flexible deterministic packet marking (FDPM)**'**, IP Operations and Management, 2004. Proceedings IEEE Workshop on 11-13 Oct. 2004

[18] Gropp, W., Lusk, E, Skjellum, A, (1996), "*Using MPI: Portable Parallel Programming with the Message-Passing Interface*", Massachusetts Institute of Technology, 1994.

[19] Foster, I, (1994), "*Designing and Building Parallel Programs: concepts and tools for parallel software engineering*", Addison-Wesley Publishing Company, (1994)

[20] Wilkson, B & Allen, M, (2005), "*Parallel Programming: Techniques and Applications using network workstations and parallel computers*", Pearson Education, Pearson Prentice Hall, (2005)