Ubiquitous Multicore (UM) Methodology for Multimedia

Ashley Chonka, *Member, IEEE,* Wanlei Zhou, *Member, IEEE and* Leanne Ngo, *Member*, *IEEE*
School of Engineering and Information Technology,
Deakin University, Melbourne
{ashley, wanlei,mln}@deakin.edu.au

Yang Xiang, Member, *IEEE*
School of Management and Information Systems
Centre for Intelligent and Networked Systems
Central Queensland University
y.xiang@cqu.edu.au

Abstract

For at least a decade or more, multimedia developers have taken for granted, that each generation of microprocessors would lead them to modify their application, in order make them run substantially faster. This so-called 'free' ride seems to be coming to an end, with results in increased clock speeds, the widening of the gap in processor and memory performance, and the tradeoffs that are needed to meet the former two points. In this paper, we propose a ubiquitous multicore (UM) design, in order to speed up computations and allow real-time multimedia. To accomplish this objective, we separate out the different multimedia and place them on their own separate core processors. For example, a manager trains his/her staff on security, by utilizing different multimedia. For example, showing a visual documentary on security which asks staff members questions, records their answers and updates the manager in real-time. As our experiments show, our UM system increases performance speeds at an average of 100%, with the average execution cost of 1.4ms, which shows multimedia resources are being used more efficiently and effectively.

Index Terms — Multicore, Multimedia, Ubiquitous Multicore framework

1. Introduction

As systems become more complex, multimedia designers are demanding more computational speed, in order to complete multimedia tasks within a "reasonable" time. This demand by multimedia designers is not the only area in Computer Science requiring greater processing speed. The information industry, in general, is pushing forward towards providing more computer systems that contain multi-processors. For example Networks, Security, Web Services etc [1][2][3] are among these areas. Multicore can be defined as two or more core processors that are connected to a single CPU. These core processors incorporate into their design, microprocessors, which in turn share computer resources. For example, L2 cache and front-side bus are shared resources[4][5].

With the push of these multicore systems, software developers are confronted with increases in complexity, such as multicore systems will roughly double every 18 months, which is forcing programmers to adopt from serialized to parallelized programs. Further, communication efficiency will be more essential, since cache resources will go down as more cores are implemented. This means that cache fragmentation and 'stale' cache will tend to get worse [13].

In this paper, we build upon our previous framework on multicore architectures [6], called ubiquitous multicore (UM). Our UM framework was applied a security system that we designed and implemented on a multicore framework [6], in order to improve performance and also deal with the issues confronting security applications on multicore systems. In [6] we separated out applications to run on their own separate core processors and found that we had a +15% increase in performance. We have found that this type of framework can be expanded into other areas like SPAM filtering [7] (currently under review) and can be

978-0-7695-3428-2/08 $25.00 © 2008 IEEE
DOI 10.1109/CSA.2008.75

applied to many other areas that require parallel processing, like multimedia. Hence, the reason of calling this framework ubiquitous multicore, since our framework can be applied as long as there is a multi-core system. In [6][7] a number of advantages were pointed out, in which could be applied to multimedia, such as: different media applications running on isolated environments, different media application running in real-time with each other, multimedia activities could be monitored and visualized in real-time. Lastly, multimedia troubleshooting could be greatly enhanced. The rest of this paper is organized as follows. Section Two briefly covers the related work done in multi-core and the UM framework. The application of UM framework being applied to multimedia design is discussed in Section Three. Section Four presents the experiments and performance evaluation that were conducted. Lastly, Section Five covers the conclusion and future work.

2. Related Work

In this section, we discuss briefly our UM framework, the use of multicore on multimedia applications, and two other areas where our multicore framework has been applied.

2.1 Ubiquitous Multicore (UM) Framework

The Ubiquitous Multicore Framework is built from a divide-and-conquer approach [9], by dividing our applications and placing them on separate core processors (Figure 1). For example, multi-media applications are grouped together and ran on their own separate core-processors along side group security applications, which in turn run on their own core processors. The application core assigner (ACA), assigns the application either on behalf of the user, or the user can

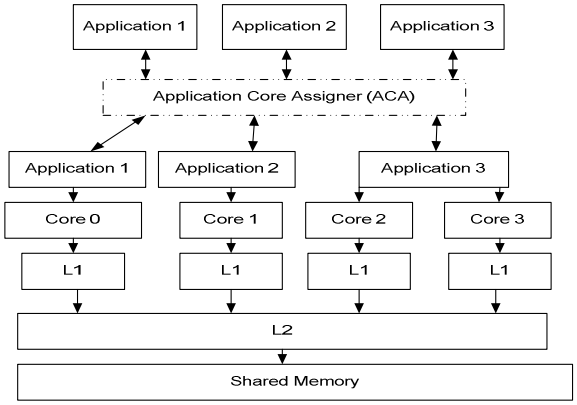

Figure 1. Ubiquitous Multicore Framework

select from the core(s) that are available. Once an 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, see figure 2.

2.2 Multicore systems used for multimedia applications.

Multicore systems have two or more processing cores integrated into a single chip [1][2][3]. In such a design, processing cores have 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 [8], 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. The UM framework is very different from Ou et al., in which UM is more abstract, by applying applications (not separate sections of a file) to separate core processors.

2.3 UM Framework for a Security Application

In our first paper [6] on multicore systems, we presented a multicore defence framework called bodyguard. Using this framework, we developed a bodyguard called Farmer (named after the Kevin Costner character in the movie, bodyguard). The basic hypothesis of the bodyguard framework, was to separate all security processes from other processes (email, browser, etc), and assign them to a set of cores. The remaining cores within the system were assigned to the applications that require security. The bodyguard framework is made up of a Forward Bodyguard (FB) and Side Bodyguard (SB). For example, in our Farmer bodyguard, the SB is responsible for providing a fast decision on whether to filter out any attack traffic. From this paper, we then were able to see that this type framework could be applied to other areas, like multimedia and Spam filtering, which lead us to the development of the Ubiquitous Multicore Framework.

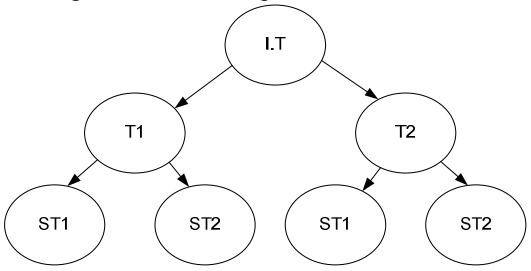

Figure 2. Example of Thread Processes on Core 0

2.4 UM framework for Multi-classifier Spam Filtering

To follow up on [6], we then applied our multicore framework to a multi-classifier SPAM filter. We found that if each classifier process is run 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, advantages that our multicore framework provided, are 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.

2.4 UM framework for Bio-Inspired Multimedia

In our most recent paper [19] we applied the UM framework to Telemedicine. Our experiment show, our UM system increases performance speeds at an average of 30%, with the average execution cost of 1.4ms, which shows multimedia resources are being used more efficiently and effectively. The benefits are as follows:

- By partitioning each application and its sub-tasks to separate cores, it will result in reducing the computational burden of the overall multicore system.
- Memory storage requirements will be reduced, since each application is assigned its own L1 cache.
- If one of the applications is idle, then its core processor can be assigned to assist the other applications, this leads to a fully optimized usage of resources.
- Lastly, if one of the applications fails, then the rest of the doctor's applications are still able to function, while maintenance is completed.

3. Applied UM Framework to multimedia applications

In this paper, we used the case study from [14], with the overall aim to help company staff members

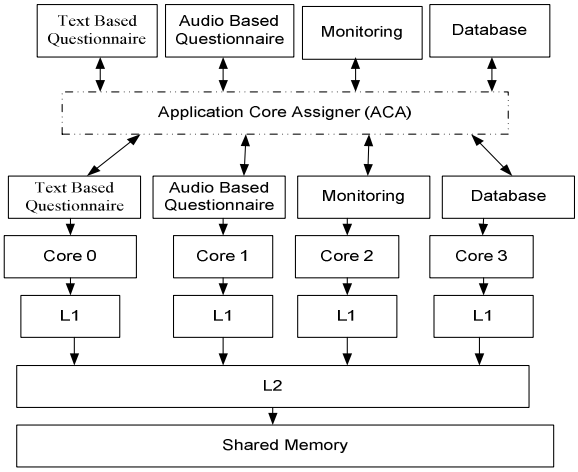

Figure 3 E-Learning IT Security Multimedia Application

their IT security culture and awareness based on our IT Security Culture Transition Model [15]. To accomplish this task a case study was conducted, in which a questionnaire was setup for staff members to answer. In conjunction as to answering the questions, the administrator was monitoring in real-time. Figure 3, displays an example of how to use the UM framework to develop and build our E-learning IT multimedia application

3.1 E-Learning IT Security Multimedia Multicore Algorithm (ELITE MA)

From figure 3, we develop Figure 4 ELITE MA (E-Learning IT Security Multimedia Algorithm). In order to partition the application takes into consideration the following the developmental algorithm equations of [17][18]. Note: Our algorithm calculations are a little different from Fosters and Wilkinson et. al., since they deal with multiple processors on different machines, our algorithm equations are based on multi-core systems. The total communication time for the 4 partition applications in a multi-core environment is as follows:

$$t_{com1} = \frac{n((cp * tcp) - 1)}{(cp * tcp)} t_{md} \quad (1)$$

where t_{md} is the transmission time for a data message sent over broadcasting, n is the power of 2, cp is each core processor being used, and tcp is the number of processor to be used. Computational time is represented by counting the number of computational

$$t_{comp} = f(n, cp) * tcp \quad (2)$$

1. Ask User if they want to assign partition manually or automatically
2. Partition Application
3. Assign Partitions to core processors.
4. Each Application performs communications and computations.
5. If application is finished go back to 1.
6. If all partitions are finished, notify user of the performance (if required), otherwise End

Figure 4. E-Learning IT Security multimedia Algorithm (ELITE MA)

steps, usually if all processors are being used then just one process computation is necessary: where n is the number of computation and cp is the number of cores. Communication Time is depended
upon the number of messages, size of the message, communication infrastructure (communication and network):

$$t_{com} = t_{beg\_startup} + wt_{md} \quad (3)$$

$t_{beg\_startup}$ is the message latency, which is the time it takes for a message to be sent with no data. The data messages sent via each partition is found in the formula:

$$t_{com2} = (\log(cp * tcp))t_{md} \quad (4)$$

For the total communication time is as follows:

$$t_{ttc} = t_{com1} + t_{com2} = \frac{n((cp*tcp)-1)}{(cp*tcp)}t_{md} \quad (5)$$

$$T_{tc} = T_{ttc} + (\log(cp*tcp))t_{md} \quad (6)$$

The computation formula for the 4 partition applications at the end of the partition phase (7) is as follows:

$$t_{comp} = \frac{n}{(cp * tcp)} \quad (7)$$

This gives us the Overall Execution Time for the 4 partition applications in the following formula:

$$t_p = \left[\frac{n((cp*tcp)-1)}{(cp*tcp)} + \log(cp*tcp)\right]t_{md} \quad (8)$$

$$t_{p1} = t_p + \frac{n}{(cp*tcp)} + \log(cp*tcp) \quad (9)$$

The very best speedup we could expect, when the 4 partitioned applications have completed their computations, is as follows:

$$\frac{t_s}{t_p} = \frac{n-1}{((n/cp*tcp)((cp*tcp)-1)+\log(cp*tcp)t_{md} + n/(cp*tcp+\log(cp*tcp)} \quad (10)$$

The actually speedup will be less than this due to partition phase; computation/communication (c/c) ratio is as follows:

$$\frac{t_{tcom}}{t_{comp}} = \frac{n/(cp*tcp)+\log(cp*tcp)}{((n/(cp*tcp))((cp*tcp)-1)+\log(cp*tcp))t_{md}} \quad (11)$$

For load balancing we use the Mandelbrot computation [16], in which if the maximum performance (mp) is reached for the processor, it will then search for another core processor to continue the work.

$$T_s \le mp * m \quad (12)$$

To partition the application correctly we use three phases communication, computation and communication.

Phase 1:

$$t_{comm1} = (p-1)(t_{stup} + t_d) \quad (13)$$

Phase 2:

$$t_{comp} \le \frac{mp*n}{p-1} \quad (14)$$

Phase 3:

$$t_{comm2} = u(t_{stup} + vt_d) \quad (15)$$

In order to maintain the highest speedup and computation/communication ratio we use the Overall Execution Time(16), Speedup factor (17), C/C ration (18):

$$t_p \le \frac{mp*n}{p-1} + (p-1)(t_{stup} + t_d) + k \quad (16)$$

$$\frac{t_s}{t_p} = \frac{mp*n}{\frac{mp*n}{p-1} + (p-1)(t_{stup} + t_d) + k} \quad (17)$$

$$\frac{mp*n}{(p-1)((p-1)(t_{stup} + t_d) + k)} \quad (18)$$

4. Performance Evaluation

We evaluate ELITE MA by simulating Figure 3, in which we assigned 4 applications on a multicore system.

4.1 ELITE MA 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 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 10ms, for each message sent by Text Based and Audio Based Questionnaire, followed by 20ms being applied to Database and Monitoring.

4.2. Simulation Setup

4.2.1 Benchmark factors

Once we have the execution times t_s, computational time t_{com}, and communication time t_{com}, we can establish what the speedup factor (19) and computation/communication ratio (20) from a single core to multicore system. The speedup is as follows:

$$\frac{t_s}{t_{cp}} = \frac{t_s}{t_{comp} + t_{com}} \tag{19}$$

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

$$\frac{t_{comp}}{t_{com}} \tag{20}$$

	Core 1	Core 2	Core 3	Core 4
Exe Time	1.5ms	1.4ms	1.3ms	1.4ms
Comp Time	.5 ms	.9ms	.3ms	.9ms
Comm Time	1ms	.5ms	1ms	.5ms
Speed Ratio	100%	100%	100%	100%
C/C	0.5	1.8	0.3	1.8
Time Complex	7.5	7.5	7.5	7.5
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 100% at the average cost of 1.4ms

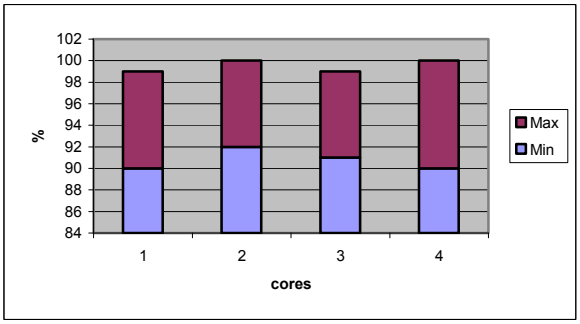

Figure 5 Min(90%)-Max(100%) CPU usage that was archived during our simulation

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

$$f(x) = O(g(x))$$
$$\left[0 \le f(x) \le cg(x)\right] \text{ for all } x \ge 0 \tag{21}$$

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. To evaluate Time complexity, we use the total sum of computation and communication (11)

$$t_{cp} + t_{com} = (n/cp+1) + (2t_{stup} + (n/cp+1)t_{md} \tag{22}$$

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)

4.2.2 Simulated Program

To measure and evaluate the performance, we wrote 4 simple programs to simulate the media applications, and assigned them to 4 cores within our multicore system by using affinity methods. The multimedia functions are simulated, by the 4 programs just to demonstrate the model, though 4 actual multimedia programs are planned in the future.

4.2. Evaluation

Based on our evaluations, displayed in table 1 and figure 5, we see that a speed average of 100% was archived at the average cost of 1.4ms. This is achieved by separating out each application and allowing them to run on their own separate cores. The 100% ratio, we think is a bit optimistic; thereby computation time and

communication time do need to be tested in real-time, to give more accurate account. The time complexity results also show that the efficiency of our algorithm is at 7.5. This means that for 13 computational steps (estimate) we achieved 7.5 data items. So, the more computations that are done the more data items we complete. For example, 15 computational steps will give us 8.5 data items. One of the results, the Computation/Communication Ratio shows that it was less then Time Complexity. This means, it will not improve speedup or efficiency beyond the figures we already have. Lastly, we see that the cost of running our program was below the cost-optimal, and at the same achieving an average of 95% CPU (see figure 5). This means that our model/program was quite cost efficient to run and resource usage (CPU) was almost fully optimized. Also, due Time Complexity is higher then Computation/Communication Ratio it would not be worthwhile trying to send our costs up to reach the optimal, since we would gain no performance benefit

5. Conclusion and Future Work

In this paper, we introduced a ubiquitous multicore framework, which was generated from our previous work on multicore. From this framework, we designed and built a multimedia multicore system called ELITE MA. The goal of such a system is to use the new multicore machines that are coming out, in order to fully utilise the power of the multicore system. We show with our experimental results that a speedup average of 100% with an average cost of 1.4ms, and a CPU efficiency of +90% for multimedia programs. In the future, we are plan to move our new multicore system on to the enterprise grid system (a number of machines with 4 cores each), at Deakin University, and to improve upon our results.

6. Acknowledgements

This research was supported by the ARC Linkage grant (Project number LP0562156).

7. References

[1] Multi-Core from Intel – Products and Platforms. http: //www.intel.com/multi- core/products.htm, 2006.
[2]AMD, (2008), http://multicore.amd.com/en/Products/, 2006.
[3] Gorder, P.M, (2007), *'Multicore processors for science and engineering*', IEEE CS and the AIP, 1521-9615/07/,March/April 2007
[4] Calandrino, J.M, Anderson, J.H., and Baumberger, D. P, 2007, '*A Hybrid Real-Time Scheduling Approach for Large-Scale Multicore Platforms*', 19th Euromicro Conference on Real-Time Systems (ECRTS'07), IEEE, 2007
[5] Bader, D.A, Kanade, V and Madduri, K, (2007), 'SWARM: A Parallel Programming Framework for Multicore Processors', Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International 26-30 March 2007 Page(s):1 – 8
[6] 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.
[7] 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
[8] 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
[9] JaJa, J. (1992), '*An Introduction to Parallel Algorithms*", Addison Wesley, Reading, MA
[10] Pierucci, L, and Del Re, E, (200), "An Interactive Multimedia Satellite Telemedicine Service," *IEEE MultiMedia*, vol. 07, no. 2, pp. 76-83, Apr-Jun, 2000
[11] Sicurello, F, (2001), ***"Some aspects on telemedicine and health network***", Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium on Volume , Issue , 2001 Page(s):651 – 654
[12] Knuth, D.E, (1976), "Big Omicron, Big Omega and Big Theta", SIGACT News (April-June), pp18-24
[13] Dongarra, J., Gannon, D., Fox, G., Kennedy, K. "The Impact of Multicore on Computational Science Software ," *CTWatch Quarterly*, Volume 3, Number 1, February 2007. http://www.ctwatch.org/quarterly/articles/2007/02/the-impact-of-multicore-on-computational-science-software/
[14] Ngo, L, Lanham E., Zhou, W., Warren, M., (2007), 'Using E-learning to improve to Improve IT Security in a corporate environment: A Case Study'
[15] Ngo, L, (2008), 'IT Security Culture Transition Process' IGI Global encyclopedia, Encyclopedia of Information Ethics and Security, edited by Dr. Quigley
[16] Wilson, G.V, (1995), '*Practical Parallel Programming*', MIT Press, Cambridge, MA
[17] Foster, I, (1994), "*Designing and Building Parallel Programs: concepts and tools for parallel software engineering*", Addison-Wesley Publishing Company, (1994)
[18] Wilkson, B & Allen, M, (2005), "*Parallel Programming: Techniques and Applications using network workstations and parallel computers*", Pearson Education, Pearson Prentice Hall, (2005)
[19] Chonka, A., Zhou, W., and Xiang, Y., (2008), "Bio-Inspired Multimedia using Ubiquitous Multicore Methodolgy"