A Performance Analysis for Microprocessor Architectures

Nakhoon Baek*
School of EECS
Kyungpook National University
Daegu 702-701
Korea

Hwanyong Lee
Solution Division
HUONE Inc.
Daegu 702-205
Korea

Abstract: In this paper, we selected three different CPU architectures for performance analysis: single-core, dual-core and hyper-threading CPU's. Four kinds of operations are executed on these architectures. After analyzing all the data, we found that the single-core and dual-core act as usually expected: the execution times of combined operations are very close to the sum of that of compounding operations. In contrast, the hyper-threading CPU shows better performance when each thread performs specific operations, rather than mixed operations.

Key–Words: CPU architectures, performance analysis, multi-threading

1 Introduction

Nowadays, the performance of microprocessors is approaching their physical limits. In the case of large-scale computers including super computers and mainframes, they already met this kind of technical limits in their CPU powers. Thus, they developed various parallel processing techniques including multi-threading, super-threading, hyper-threading, and so on[1].

In these days, microprocessors used in conventional PC's and even in high-end embedded systems have improved their ability to effectively support parallel processing techniques. At this time, we already have some commercial multi-core CPU's including Intel Core2 Duo, Intel Core2 Quad, Intel Xeon, customized triple-core CPU's for Xbox 360, etc[2, 3].

It is clear that conventional programming models based on the sequential processing paradigm is not exactly suitable for these multi-core CPU's. We need computer programs based on the parallel processing paradigm such as multi-processing and/or multi-threading.

In this paper, we represent the experimental results on the execution time of some CPU-intensive operations for an amount of integer operations and/or floating-point operations, to finally analyze which programming architecture is more suitable for newly appeared CPU architectures.

*Corresponding Author.

2 Background Works

In computer programming, a thread means a light process, which executes a given area of programming codes, with a dedicated stack area[4]. In contrast to usual processes, threads can share their memory each other, which can act as a strong point. Comparing the conventional sequential programming paradigm to the multi-threaded programming, one of the most strong point for the multi-threading is that multiple threads can be simultaneously executed in a parallelized manner[5].

Nowadays, multiple threads can be simultaneously executed on many computer systems. On single processor systems, the time sharing method is used to execute several threads, in turns. Through alternating the executing thread very frequently, this system can make the illusion of simultaneous executions. However, in fact, the single processor systems only alternate multiple threads, rather than physically executing the threads in a parallelized way.

Multi-processor systems or multi-core processors are capable of physically executing multiple threads. In other words, multi-processing is now possible. Thus, we can use the multi-threading in more wide areas of programming, although they recommend it only for I/O intensive works, in past.

Multi-threading does not always make overall speed ups in all situations[6]. First of all, parallel programs based on the multi-threading needs much more steps to start something useful, in comparison with previous sequential programming techniques. Thus, in worst case, the preparing and arranging times for the multi-threading requires somewhat significant

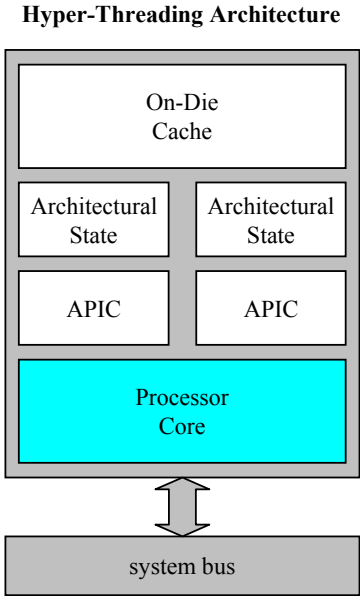

Figure 1: Hyper-threading architecture.

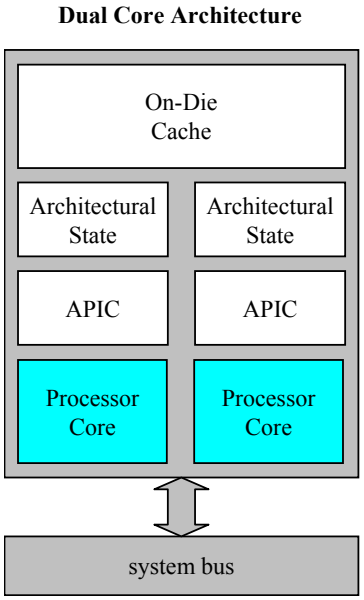

Figure 2: Dual-core architecture.

portions in its overall execution time. Of course, programmers should avoid this situation.

Hyper-threading is a recently developed technology for more efficient multi-threading, on the Pentium4 microprocessor architectures, delivered by Intel. It is also officially known as HTT(hyper-threading technology). In this technology, when a processing core is active, the other CPU pipelines not in use may be used by other threads, to finally simulate two logical processors in a single physical processor. So, we can expect two logical processors in a hyper-threading possible CPU's[7]. Figure 1 shows the conceptual diagram for the hyper-threading environment.

In spite of its strong points, hyper-threading also has drawbacks. Since the logical cores share level-1 and level-2 caches, there is some security holes and some slow-downs in real world applications[8].

Multi-core microprocessors have two or more processing cores in a single physical processor package, as shown in Figure 2. In this case, each processing core has its own resources such as caches, registers, execution units, etc. Thus, there is no resource sharing in multi-core architectures, while hyper-threading invokes some kind of resource sharing. Some multi-core processors are designed to cooperate with hyper-threading technology.

Although multi-core CPU's are one of cost-effective way of implementing parallel programming paradigm, it also has some drawbacks. At this time, multi-core CPU's have slower clock than conventional single-core CPU's. Thus, current multi-core CPU's show bad scores for sequentially designed computer programs, in comparison with single-core CPU's[9].

At this time, we have dual-core and quad-core CPU's commercially available. A customized CPU for Xbox 360 has triple cores, while some CPU's for workstation computers also have multi-core architectures.

3 Performance Analysis

In this paper, we will compare several CPU architectures: single-core, dual-core and hyper-threading CPU's. For this purpose, we select four kinds of operations. Basically, we focused on the arithmetic operations, to fully utilize the internal computing power of CPU's. To test integer operation units and floating-point units separately, we prepare the following operations:

- **integer operations:** consist of 1,000 integer additions, which are repeated 5,000,000 times.
- **floating-point operations:** consist of 1,000 double precision floating-point additions, repeated 5,000,000 times.
- **mixed operations:** consist of 1,000 integer additions and 1,000 floating-point additions. When multiple threads are used, each thread is allotted to the same amount of integer and floating-point operations.
- **separated operations:** consist of 1,000 integer additions and 1,000 floating-point additions. When multiple threads are used, each thread is wholly served for integer operations or for

Table 1: Execution times on the single-core CPU.

(unit: sec)

num. threads	operations			
	integer	double	mixed	separated
1	1.294	2.145	3.491	3.453
2	1.306	2.157	3.459	3.435
3	1.316	2.173	3.499	3.457
4	1.326	2.155	3.463	3.457
5	1.336	2.137	3.461	3.469
6	1.346	2.165	3.595	3.477
7	1.388	2.169	3.545	3.455
8	1.390	2.147	3.511	3.467

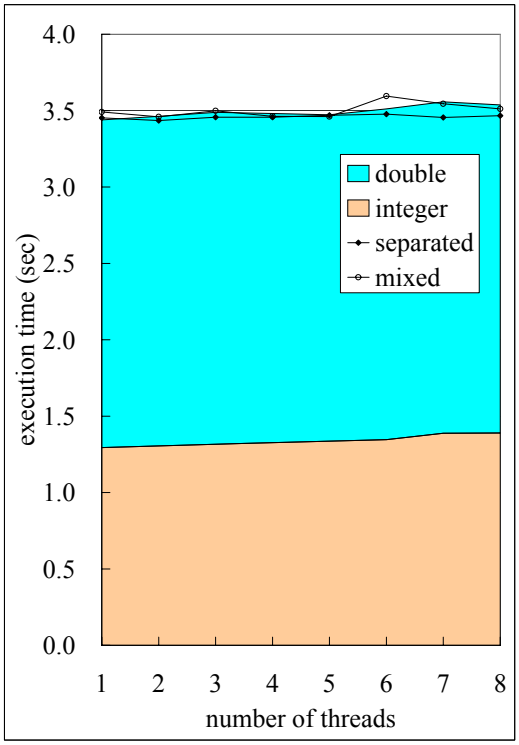

Figure 3: Single-core CPU performance.

floating-point operations. I.e., the integer and floating-point operations are separated into independent threads.

3.1 Single-core case

We use an Intel Pentium4 1.6 GHz processor with 1.5GB memory as a testing system for the single-core case. The measured execution times for the four kinds of operations are shown in Table 1. All the operations are measured for varying number of threads from 1 to 8. The graphical representation of these data is also shown in Figure 3. Since there is only one CPU core, the execution time is independent on the number of threads.

As we can trivially guess, the execution times for separated-operations and mixed-operations threads are very close to the sum of those of integer-operations and double-operations threads. Additionally, there is no noticeable difference between the execution time of mixed-operation and separated-operation threads.

Table 2: Execution times on the dual-core CPU.

(unit: sec)

num. threads	operations			
	integer	double	mixed	separated
1	0.691	1.772	2.716	2.459
2	0.366	0.894	1.256	1.775
3	0.403	0.941	1.294	1.334
4	0.400	0.919	1.297	1.331
5	0.400	0.931	1.306	1.306
6	0.409	0.928	1.319	1.303
7	0.412	0.925	1.319	1.281
8	0.416	0.903	1.316	1.294

3.2 Dula-core case

For dual-core cases, we use an Intel Core2 E6400 2.13GHz CPU system, with 1.0GB memory. The experiments are actually the same to the single-core case. The experimental results are summarized in Table 2 and Figure 4.

Since we use a dual-core CPU, the execution time with 2 or more threads are dropped to half of the execution time of single threaded case. Similar to the single-core case, the execution times for separated-operations and mixed-operations threads are very close to the sum of those of integer-operations and double-operations threads.

3.3 Hyper-threading case

To test the hyper-threading CPU case, an Intel Pentium4 2.8 GHz processor, with hyper-threading facility is used, with 1.0GB memory. Measured execution times are listed in Table 3, and its corresponding graphical representation is shown in Figure 5.

One remarkable thing on the graph is that the separated-operations threads show better performance with respect to the mixed-operations threads. We guess that the processor core is fully utilized when a

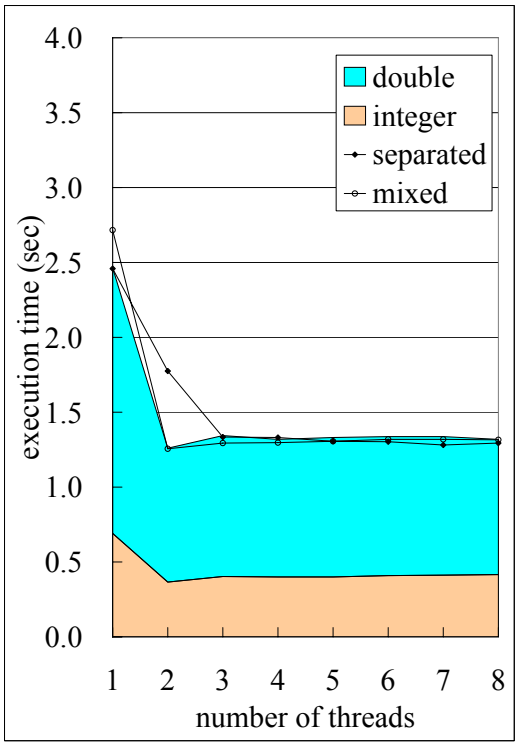

Figure 4: Dual-core CPU performance.

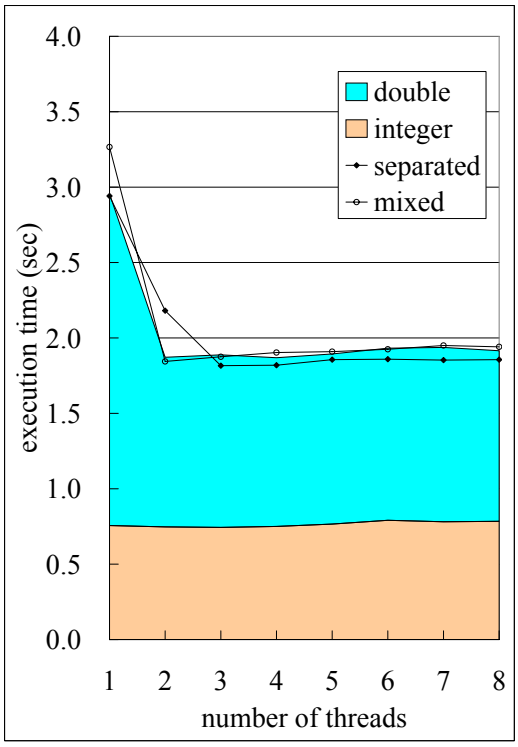

Figure 5: Hyper-threading CPU performance.

Table 3: Execution times on the hyper-threading CPU.

(unit: sec)

num. threads	operations			
	integer	double	mixed	separated
1	0.756	2.200	3.266	2.941
2	0.747	1.125	1.844	2.181
3	0.744	1.144	1.875	1.816
4	0.750	1.119	1.903	1.819
5	0.766	1.128	1.909	1.856
6	0.791	1.141	1.925	1.859
7	0.781	1.156	1.950	1.853
8	0.784	1.131	1.941	1.856

thread uses integer-operation unit and another thread uses floating-point unit.

4 Conclusion

In this paper, we selected three different CPU architectures for performance analysis: single-core, dual-core and hyper-threading CPU's. Four kinds of operations are executed on these architectures. We measured the execution times of these all operations, for different number of threads from 1 to 8.

After analyzing all the data, we found that the single-core and dual-core act as usually expected, i.e. the execution time of combined operations are very close to the sum of that of compounding operations. In contrast, the hyper-threading CPU shows better performance when each thread performs specific operations, rather than mixed operations.

Conclusively, in the case of hyper-threading CPU's, we had better design the multi-threading software to avoid a thread with mixed operations. We need more experiments and analysis for more precise inferences.

Acknowledgements: This work is financially supported by the Ministry of Education and Human Resources Development(MOE), the Ministry of Commerce, Industry and Energy(MOCIE) and the Ministry of Labor(MOLAB) through the fostering project of the Industrial-Academic Cooperation Centered University.

References:

[1] J. Stokes. Introduction to multithreading, superthreading and hyperthreading, 2005. http://arstechnica.com/articles/paedia/cpu/hyperthreading.ars.

[2] J. Stokes. Inside the Xbox 360, part I: procedural synthesis and dynamic worlds, 2005. http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars.

[3] J. Stokes. Inside the Xbox 360, part II: the Xenon CPU, 2005. http://arstechnica.com/articles/paedia/cpu/xbox360-2.ars.

[4] K. Wackowski and P. Gepner. Hyper-threading technology speeds clusters. In *Proc. 5th Int'l Conf. on Parallel Proc. and Appl. Math.*, pages 17–26, 2003.

[5] G. Keren. Multi-threaded programming with POSIX threads, 2002. http://users.actcom.coil/-choo/lupg/index.html.

[6] D. Sarkar. Cost and time-cost effectiveness of multiprocessing. *IEEE Trans. Parallel Distrib. Syst.*, 4(6):704–712, 1993.

[7] T. Martinez and Sunish Parikh. Understanding dual processors, hyper-threading technology, and multi-core systems. *Intel Optimizing Center*, 2005. http://www.devx.com/Intel/Article/27399.

[8] C. Percival. Cache missing for fun and profit. In *BSDCan '05*, 2005.

[9] J. Handy. *The Cache Memory Book*. Academic Press, 1998.