The Effect of Core Number and Core Diversity on Power and Performance in Multicore Processors

A. Zolfaghari Jooya and M. Soryani

Computer Science Dep., Iran University of Science and Technology,
Tehran, Iran
al_zolfaghari@comp.iust.ac.ir,
soryani@iust.ac.ir

Abstract. Today, multi-core processors dominate server, desktop and notebook computer's market. Such processors have been able to decrease power consumption and thermal challenges that designers face within single core processors. In order to improve multi-core processor's performance, designers should choose the best set of cores based on power consumption and execution delay. In this paper, we study several architectures that are composed of a configurable number of cores. We use three cores with different levels of performance and power consumption. Then, we implement different configurations of a multi-core processor. In each configuration, which has a different set of cores, we run benchmarks with various numbers of simultaneous threads, from 1 up to 32. Power consumption and execution delay of each configuration has been measured. It has been shown that the best configuration is a heterogeneous multi-core processor that is composed of 16 cores in our bounded area. Then, we examined various ways that threads can be assigned to different cores in the best configuration. It is shown that for serial workloads the best choice is to use high performance cores, but in parallel workloads that consist of multiple threads, a mixture of cores with different performance levels gives the best performance.

Keywords: Heterogeneous multi-core processor, power consumption, execution delay, multithreaded benchmark.

1 Introduction

The constant decrease in feature size leads to an increase on transistor count on chip area which enables processor architects to improve performance by designing more complex processors. This amount of transistors on the chip area increases power consumption and produces more heat. These two factors, i.e. power consumption and thermal management, are the most important design limitation factors today [7,8,9].

From computational point of view, we have already extracted the easy ILP (instruction-level parallelism) through techniques like superscalar processing, out-of-order processing, etc.. The ILP that is left is difficult to exploit. However, technology keeps making transistors available to us at the rate predicted by Moore's Law [11]. We have reached a point where we have more transistors available than we know how to make effective use of in a conventional monolithic processor environment.

There is diversity in workloads that a typical processor is expected to run. This diversity can be due to diversity among applications or different threads of the same application. It can also be due to diversity across varying program phases within an application or varying processor load. Instead of using all the transistors to construct a monolithic processor targeting high single-thread performance, we can use the transistors to construct multiple simpler cores where each core can execute a program (or a thread of execution) [1]. A multicore processor provides increased total computational capability on a single chip without requiring a complex microarchitecture.

As a result, simple multicore processors have better performance per watt and area characteristics than complex single-core processors [10]. Multicore processors have better adaptability whit workloads and tune the number of active cores according to different workloads or different phases of a single application. This adaptability leads to consume less power and reduce temperature of the chip.

In this paper we implement three cores, each with a different level of performance, power and area. We use different composition of these cores on our bounded area and show delay and power level of each configuration. After finding the best composition of cores, we study the effect of different ways that threads can be assigned to different cores.

Many works have appeared in the literature exploring the design space of multicore processors from the point of view of different metrics and application domains.

Huh et al. [2] evaluate the impact of several design factors on the performance. The authors discuss the interactions of core complexity, cache hierarchy, and available off-chip bandwidth. The paper focuses on a workload composed of single-threaded applications. It is shown that out-of-order cores are more effective than in-order ones.

In [3], Kumar et al. propose a single-ISA heterogeneous multi-core architecture as a mechanism to reduce processor power dissipation. They assume a single chip containing a set of diverse cores that target different performance levels and consume different levels of power. They describe an example architecture with five cores of varying performance and complexity.

In [4], Jouppi et al. demonstrate that single-ISA heterogeneous multi-core architecture can provide significantly higher performance in the same area than a conventional multiprocessor chip. It does so by matching the various jobs of a diverse workload to the various cores. Authors show that this type of architecture covers a spectrum of workloads particularly well, providing high single-thread performance when thread parallelism is low, and high throughput when thread parallelism is high.

In [5] Monchiero et al. target an architecture composed of a configurable number of cores, a memory hierarchy consisting of private L1 and L2, and a shared bus interconnect. They explore the design space varying the number of cores, L2 cache size and processor complexity, showing the behavior of the different configurations/applications with respect to performance, energy consumption and temperature.

The rest of paper is organized as follows. In section 2 we discuss our design methodology and define the details of microarchitecture, simulator and benchmarks. In section 3 we introduce simulation results. Finally, Section 4 concludes the paper.

2 Methodology

2.1 Microarchitecture

We have implemented three cores with different levels of performance, power and different chip areas. Our cores are similar to Alpha processors (alpha21064 [12], alpha21164 [13] and alpha21264 [14]). The core characteristics are similar to the ones used in [3]. Table 1 summarizes these characteristics. We have changed the issue-width of EV5 from 2 to 4 and its instruction and data cache size from 8 KB to 16 KB, in order to increase performance distance between EV4 and EV5. More than half of the chip area was considered for L2 cache and interconnections. In the remainder of chip area we can put four EV6 (alpha21264), or twenty EV5 (alpha21164), or forty EV4 (alpha21064), or a different mixture of these cores. We consider that all cores are implemented in 100 nm technology and run at 2.1 GHz.

A large L2 cache (4 MB) was used to be shared between cores. The L2 cache is a 4 way set associative with a block size of 32 bytes.

Table 1. Characteristics of the cores

core	EV4	EV5	EV6
Issue-width	2 (in-order)	4 (in-order)	6 (OOO)
I-cache	8 KB , DM	16 KB , 2 way	64 KB , 4way
D-cache	8 KB , DM	16 KB , 2 way	64 KB , 4way
B-predictor	static	hybrid	hybrid
Area (mm2)	2.5	5	25
Power (watt)	5	7.5	20

2.2 Simulator and Benchmarks

We used the SESC [15] simulator which is a cycle accurate architectural simulator. It models a very wide set of architectures such as single processors, CMPs and thread level speculation. For simulating heterogeneous multi-core processors, we modified the configuration file and added different core configurations.

We have used four scientific/technical parallel workloads from splash2 [6]. These workloads consist of two applications and two computational kernels. The kernels are FFT and LU decomposition. The two applications that we have used are Barnes and Ocean. Table 2 lists the benchmarks that we selected and the input parameters of each one.

3 Experimental Results

3.1 Different Compositions of Cores

In this section we present the simulation results for different core configurations of our processor. We show execution delay (msec), power and energy-delay of our simulation results. The first part of Fig. 1 shows the execution delay for some configurations of the processor for ocean benchmark.

Note that 2.6.8 means our processor has two EV6, six EV5, eight EV4 cores. EV6 and EV4 are the highest performance and the lowest performance cores respectively. Also there are some assumptions in our simulation:

Number of simultaneous threads is considered to be power of two.

Number of simultaneous threads that run on cores must be less than or equal to the number of cores.

If the number of simultaneous threads is less than the number of cores, we assume that extra cores are off and don't consume power.

The power that is consumed by cores is reported in the results, not total chip power.

We have used static scheduling. The cores with higher performance have priority for usage to cores with lower performance (when threads are assigned to cores), so threads are first assigned to high performance cores and then, if there are any left, they are assigned to other cores i.e. if we run 8 threads on 2.6.8 multi-core processor, 2 threads are assigned to two EV6 and 6 of them run on six EV5.

Table 2. Benchmarks and input parameters

Bench.	Description	input

FFT	Perform 1D fast Fourier transform using six-step FFT method	m = 16
		l = 5
		n = 1024

Ocean	This application studies the role of eddy and boundary currents in influencing large-scale ocean movements.	N = 130

LU	Parallel dense blocked LU factorization	n = 1000
		b = 64

Barnes	implements the Barnes-Hut method to simulate the interaction of a system of bodies	nbody = 64
		seed = 45
		fleaves = 5.0

There are three homogeneous multi-core processors in this figure (e.g. 4.0.0 that consists of four EV6 cores, 0.20.0 that consists of twenty EV5 cores and 0.0.40 that consists of forty EV4 cores). Other configurations implement heterogeneous multi-core processors with two or three different cores.

Note that our simulation is limited with this fact that the number of workload's threads must be power of 2 and less than the number of cores that we implement in our design. Therefore we can't show execution delay of 2.6.8 configuration, which has 16 cores, for the benchmark that has 32 simultaneous threads. But we can see that 1.0.31 configuration has the best execution delay for 32thread- benchmark in all figures. If we run a serial workload on a low performance core, we will get the worst delay.

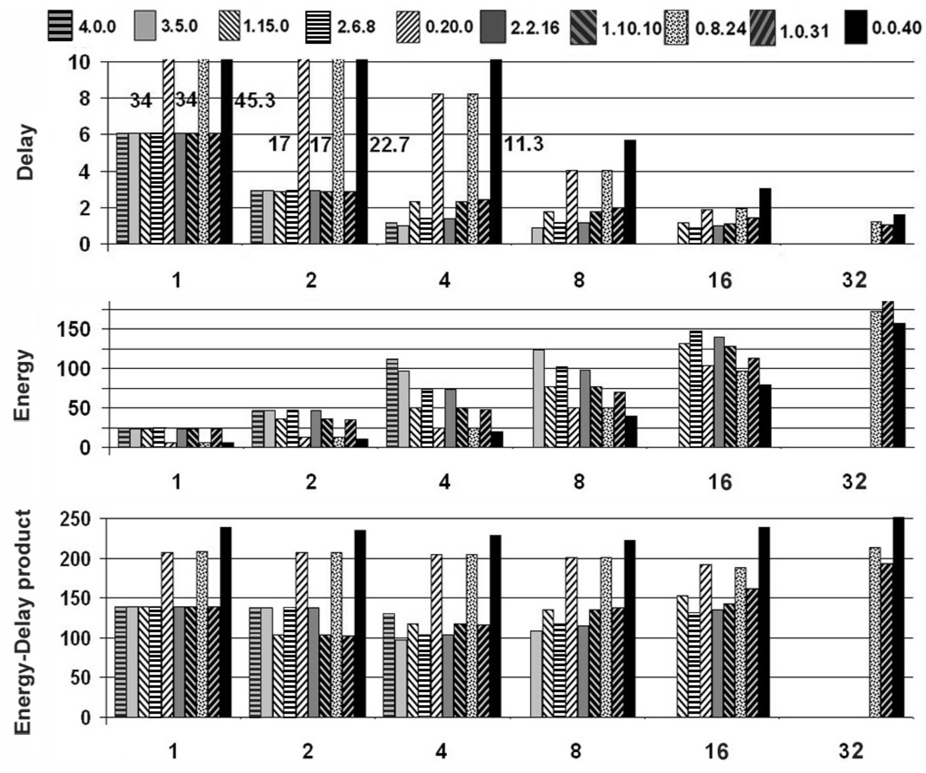

Fig. 1. Execution-Delay, energy and energy-delay product for ocean benchmark for different thread numbers

The middle part of Fig. 1 shows the amount of power that is consumed in different configurations for ocean benchmark. It is clear that if we run the application's threads on high performance cores, we must pay the penalty that is more power consumption. Note that the total power consumption is increased with the number of simultaneous threads, because more cores are used simultaneously.

The last part of this figure shows Energy-Delay product diagrams. It can be seen that 0.0.40 (homogeneous multi-core) always has the worst Energy-Delay curve. The best configurations are 2.6.8 and 1.0.31 for workloads that have 32 simultaneous threads.

Depending on the goal of the design; the scheduler can use selected cores to reach the best result. For example consider that we implement the 2.6.8 configuration for multi-core processor in an embedded system and the amount of power that is consumed is critical. Scheduler can run multithreaded workload on the low performance cores that consume less power. If we run 8 threads on 8 EV4 cores in 2.6.8 configuration, the power that is consumed is 1/3 of when we run 8 threads on 2 EV6 and 6 EV5 cores. If the goal of design is to reach the best execution delay, the threads must run on high performance cores.

Heterogeneous multi-core processors are suitable for systems that have variable goals in their life time. For example embedded systems can have multiple goals, depending on environmental or operational situations. Reducing power or execution delay or both of them can be such system's goals. In these systems heterogeneous multi-core processors have the best compatibility with these goals, and scheduler can decide that which cores be used to run the threads.

We implemented many other possible configurations. We found that 2.6.8 configuration still has the best performance and some other configurations have performance near to that. For example 2.4.12 and 2.5.10 have performances near 2.6.8 and 3.5.0 has the best Energy-Delay product. Other benchmarks shown in Table 2 have also been used in the experiments and similar results have been achieved.

3.2 Thread Assignment Policy

In this section we choose the configuration that had the best performance in previous section, and study the effect of different thread assignment policies. We ran different number of simultaneous threads (from 1 thread to 8 threads) on different cores and compared the execution delay, power and energy-delay of each simulation result. We found the best assignment for each number of threads from the execution delay, power and energy-delay aspects.

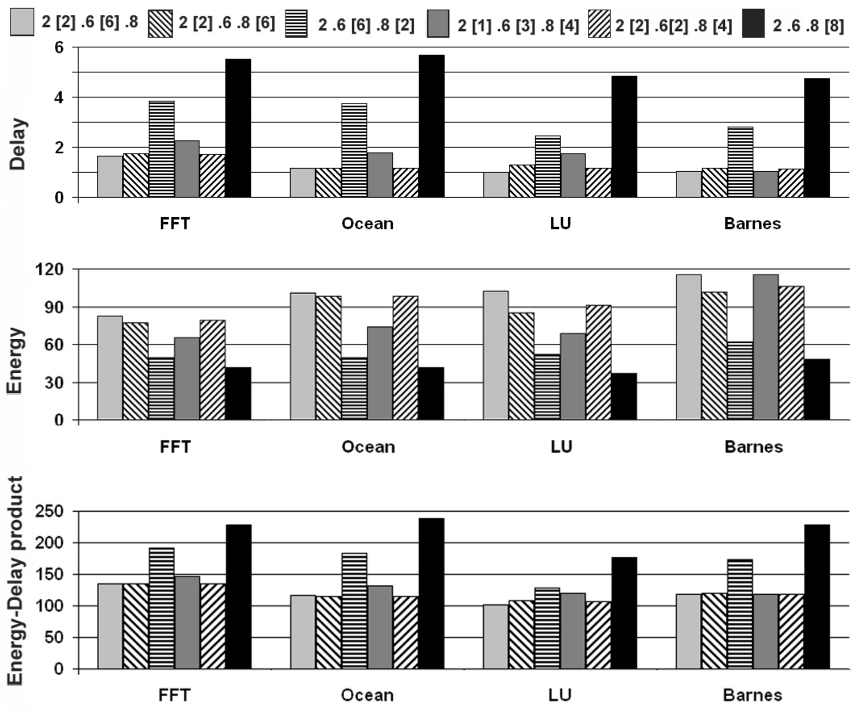

Fig. 2. The 2.6.8 configuration with 8 simultaneous threads

Fig. 2 shows the simulation results for the workload that consists of 8 threads. This figure is composed of three parts that represent power consumption, delay and energy-delay product. The threads that run in each group of cores are shown in brackets. For example (2 . 6 [6] . 8 [2]) configuration means that our workload has eight threads and six of them run on six EV5 and other two threads run on two EV4 cores.

In this experiment we assume 8 threads are available to be run on cores. A smart scheduler that uses the best cores to execute the workload can save both power and time. For example, consider two configurations. The first one is 2 [2] .6 [6] .8 and the second is 2 .6 [6] .8 [2]. The first configuration's energy-delay is 29.5% better than second configuration, but its power consumption is two times worse. Also in second configuration, we can keep high performance cores (two EV6 cores) for serial jobs.

In another example we compare the first configuration of previous example with 2 [2] .6 [2] .8 [4]. Both of them have nearly the same delay, power and energy-delay product. Note that in the first configuration eight EV4 cores remain idle in processor that has the worst delay, but in the second configuration we still have four EV5 and four EV4 cores, that have much better performance than eight EV4. It is concluded that for this situation 2 [1]. 6 [3]. 8[4] configuration is the best, because half of the number of each core are used, and other half remain idle, and scheduler has a better choice for other loads.

Workloads that have high parallelism give more choice to scheduler to assign threads to different cores, and scheduler can choose the best configuration to reach to the goal of the design.

4 Conclusions

In this work we considered a bounded chip area and implemented a multi-core processor whit different set of cores. We used three cores with different levels of performance. Our results showed that a processor that consists of all three cores gives the best delay and energy-delay result. We also found that if energy-delay is the measure of performance, the best collection of cores to execute threads is a mixture of all kinds of cores.

Other design parameters that affect performance of CMPs are L2 cache configuration that is shared between cores, interconnection network, and a smart scheduler that can explore different phases of application and predicts the demand of next phase in order to choose the best set of cores to execute the workload.

References

1. Kumar, R.: Holistic Design for Multi-core Architectures. PhD thesis, university of California, san diego (2006)
2. Huh, J., Burger, D., Keckler, S.: Exploring the design space of future cmps. In: PACT 2001: Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques, pp. 199–210, Washington, DC, USA (2001)
3. Kumar, R., Farkas, K., Jouppi, N.P., Ranganathan, P., Tullsen, D.M.: Single-ISA Heterogeneous Multi-Core Architectures:The Potential for Processor Power Reduction. In: Proceedings of the 36th International Symposium on Microarchitecture (Decemeber 2003)

4. Jouppi, N.P., Tullsen, D.M., Kumar, R., Ranganathan, P., Farkas, K.I.: Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In: Proceedings of the 31st International Symposium on Computer Architecture (2004)
5. Monchiero, M., Canal, R., González, A.: Design Space Exploration for Multicore Architectures: A Power/Performance/Thermal View. In: ICS 2006 (2005)
6. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 Programs: Characterization and Methodological Considerations. In: Proceedings of the 22nd International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, pp. 24–36. ACM Press, New York (1995)
7. Brooks, D., Martonosi, M.: Dynamic Thermal Management for High-Performance Microprocessors. In: Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Monterrey, Mexico (January 2001)
8. Brooks, D., et al.: Power-aware Microarchitecture: Design and Modeling Challenges for the next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
9. Flynn, M.J., Hung, P., Rudd, K.: Deep-Submicron Microprocessor Design Issues. IEEE Micro 19(4), 11–22 (1999)
10. Balakrishnan, S., Rajwar, R.: The Impact of Performance Asymmetry in Emerging Multicore Architectures. In: Proceedings of the 32nd International Symposium on Computer Architecture ISCA 2005 (2005)
11. Moore, G.: Cramming more components onto integrated circuits 38 (1965)
12. Alpha 21064 and Alpha 21064A: Hardware reference Manual. Digital Equipment Corporation (1992)
13. Alpha 21164 Microprocessor: Hardware Reference Manual. Digital Equipment Corporation (1998)
14. Alpha 21264/EV6 Microprocessor: Hardware Reference Manual. Compaq Corporation (1998)
15. Renau, J., Fraguela, B., Tuck, J., Liu, W., Prvulovic, M., Ceze, L., Sarangi, S., Sack, P., Strauss, K., Montesinos, P.: SESC simulator (January 2005), `http://sesc.sourceforge.net`