Multiple Clustered Core Processors

Sato, Toshinori
Kyushu University

Chiyonobu, Akihiro
Kyushu Institute of Technology

Multiple Clustered Core Processors

Toshinori Sato
Kyushu University
toshinori.sato@computer.org

Akihiro Chiyonobu
Kyushu Institute of Technology
chiyo@mickey.ai.kyutech.ac.jp

Abstract

This paper proposes multiple clustered core processors as a solution that attains both low power consumption and easy programming facility. Considering the current trend of increasing power consumption and temperature, a lot of CPU venders have shipped or announced to ship multiple core processors. Especially, recent studies on heterogeneous multiple core processors show that they are more efficient in energy utilization than homogeneous ones. However, they request programmers to consider complex task scheduling since the size of every task always has to match the performance of core where it is allocated. Multiple clustered core processors relieve them from such a tedious job. Simulation results show that a multiple clustered core processor consumes slightly more power than a heterogeneous multiple core processor. However, in a case, the heterogeneous multiple core processor cannot solve a severe task scheduling problem, while the multiple clustered core processor can.

1. Introduction

Multiple core (MultiCore) processors are a promising solution that achieves high performance with low power consumption. This is because processor performance is proportional to the square root of its area, while its power consumption is proportional to the area. Thus, from the power consumption view, a MultiCore processor with a lot of small cores is a good solution. However, due to the difficulty in aggregating parallelism from some kinds of programs, it is difficult to achieve requested performance using the homogeneous many-core processor. Based on the considerations, heterogeneous MultiCore processors are proposed [7, 12, 16]. The heterogeneous MultiCore processor consists of several cores with different scales in chip area and performance. When a task requires high performance but it does not have large parallelism in it, a large core serves. When the other task also requires high performance and it has large parallelism, multiple small cores serve. And high performance is not required, a small core is utilized. The efficient use of different kinds of cores satisfies requested performance with low power consumption.

One of the problems of the heterogeneous MultiCore processors is the difficulty in programming. Programmers always have to concern where every task should be allocated. Small tasks should be allocated to small cores, while large tasks have to be allocated to large ones. This is a tedious job. Hence, from the programming view, a homogeneous MultiCore processor is a good solution. In order to achieve the two requirements: low power consumption and easy programming, we propose multiple clustered core processors.

The multiple clustered core processor is a homogeneous MultiCore processor. However, it consists of multiple clustered cores. The core is based on the clustered microarchitecture [5]. We exploit the clustered microarchitecture to realize heterogeneity on the homogeneous MultiCore processor. In order to attain the goal, we propose cluster gating. When high performance is not required, some clusters are gated off. Using the cluster gating, only a small number of clusters in the core are active so that requested performance of the associated task is satisfied. If the cluster gating is efficiently managed, programs which are implemented considering homogeneous MultiCore processors benefit from the virtually heterogeneous MultiCore processor. That makes programming easy.

This paper is organized as follows. Section 2 reviews related work. Section 3 proposes multiple clustered core processors as a solution for high performance and low power. Section 4 presents evaluation results. Finally, Section 5 concludes.

2. Related Work

The current trend of increasing power consumption prefers MultiCore processors as a solution to achieve both high performance and low power, and actually some commercial MultiCore processors are emerging [8, 9, 11, 13]. Since processor performance is proportional to the square root of its area while its power consumption is proportional to the area, homogeneous many-core processors are a good

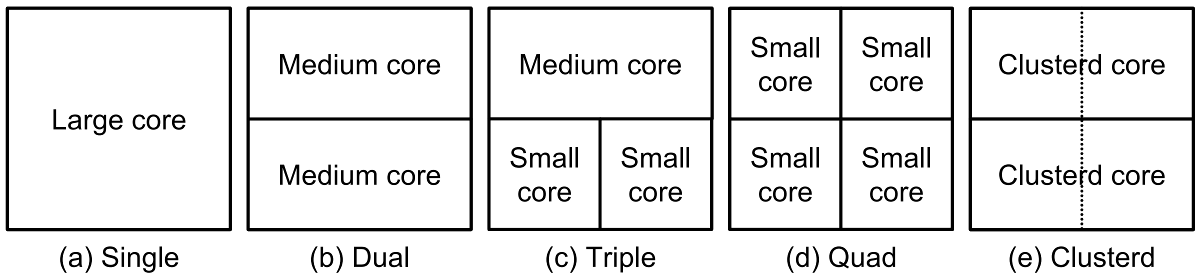

Figure 1. Different types of MultiCore processors

solution for power efficiency. However, due to the difficulty in aggregating parallelism from some kinds of programs, it is difficult to achieve requested performance using homogeneous MultiCore processors, and thus currently heterogeneous ones are proposed [7, 12, 16] as a solution of this problem.

The clustered microarchitecture is a solution to solve the wire delay problem, and there are a lot of studies [5, 10, 14]. A large processor core is divided into multiple clusters. Each cluster is so small that it mitigates wire delay problem. General purpose processors are designed to achieve the best performance on any kinds of application programs, and thus there are much more processor resources than most programs require. Thus, it is desirable that processor resources are turned on and off on demands of applications. Pipeline balancing [1] is such a technique, which reduces issue width when a program phase does not require the full issue width. This is possible by turning off some or all pipelines in one cluster, but it always keeps all register files turned on. Dynamic cluster resizing [6] is another technique that adapts the number of instruction queues in each cluster while maintains the total number of clusters. Dynamically tunable clustered design [2] enables to turn off entire clusters when communication overheads prefer smaller number of clusters.

3. Multiple Clustered Core Processors

MultiCore processors are a promising solution that achieves high performance with low power consumption. Figure 1 shows different types of MultiCore processors. Figure 1a is a uniprocessor. Figures 1b and 1d are homogeneous MultiCore processors, while Figure 1c is a heterogeneous one. As you can see, the heterogeneous MultiCore processor consists of several cores with different scales in area and performance. When a task requires high performance but it does not have large parallelism in it, a large core serves. When the other task also requires high performance but it has large parallelism in it, it is better in energy efficiency that multiple small cores serve. When high performance is not required by another task, a small core is utilized. The efficient use of different kinds of cores satisfies requested performance with low power consumption. From the view of energy efficiency, heterogeneous MultiCore processors consisting of cores with different scales are a good solution.

3.1. Issues in heterogeneous MultiCores

One of the problems of the heterogeneous MultiCore processors is the difficulty in programming. Programmers always have to concern where every task should be allocated. Small tasks should be allocated to small cores, while large tasks have to be allocated to large cores. This is a tedious job. For example, imagine that there are two large tasks. We have a large core and a lot of small cores. A single small core does not have enough performance. There are two possible solutions. One is allocating both tasks to the large core, and they are serially executed. The other is dividing one of the tasks into several small tasks so that each of them matches small core's performance. Then, all tasks, the large task and small ones, are executed in parallel. In order to determine which solution a programmer selects, he or she has to check which one is better in performance and in energy efficiency. This is very time consuming work. Imagine there are a lot of tasks with different sizes. There are too many possible solutions for a programmer to select one in a practical time.

Therefore, from the view of easy programming, a homogeneous MultiCore processor consisting of large cores is a good solution. However, as you can easily guess, they are less power efficient. In order to achieve the two requirements: low power consumption and easy programming, we propose multiple clustered core processors.

The multiple clustered core processor is a homogeneous MultiCore processor. The difference from the conventional homogeneous MultiCore processor is that it consists of multiple clustered cores

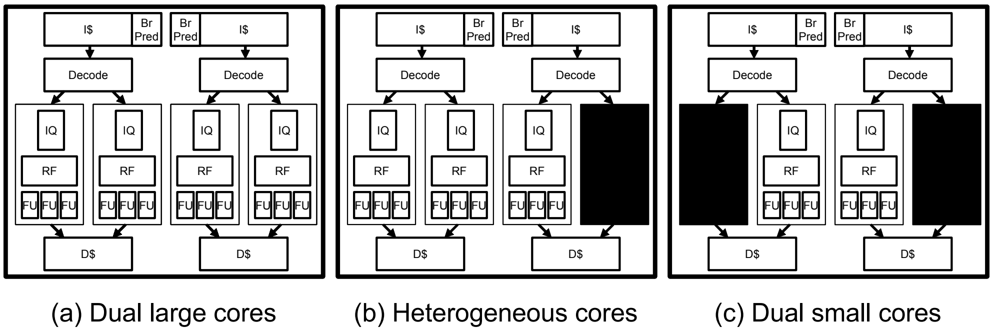

Figure 2. Cluster gating

rather than monolithic cores. The core is based on the clustered microarchitecture [5]. HP alpha 21264 [10] is an example of the clustered microprocessors. Figure 1e shows a multiple clustered core processor. It has two homogeneous clustered cores, each of which has two clusters.

3.2. Cluster gating

We exploit the clustered microarchitecture to realize heterogeneity on the homogeneous MultiCore processor. What we require is the heterogeneity in power and performance rather than that in structures. In order to attain the goal, we propose cluster gating. Figure 2 explains how the cluster gating works. This is a dual core processor consisting of two dual cluster cores. Figure 2a shows a homogeneous dual core processor consisting of large cores. When high performance is not required, some clusters are turned off, as shown in Figure 2b. The black box means that the cluster is turned off. Using the cluster gating, only a small number of clusters in the core are active so that requested performance of the allocated task is satisfied. Now, we have a heterogeneous dual core processor. Figure 2c shows a dual core processor consisting of small cores, in both of which one of the clusters is turned off. The difference between the cluster gating and dynamically tunable clustered design [2] is in the considerations of task size.

Considering the requested performance, one of the clusters becomes inactive. If the cluster gating is efficiently managed, programs which are implemented considering homogeneous MultiCore processors benefit from the virtually heterogeneous MultiCore processor. That makes programming easy, because we do not have to concern what kind of cores are available when we consider task allocation. We can always get a desirable scaled core. There are some options to realize the cluster gating. One is hardware-based. A dedicated hardware block in a core observes the characteristics of a task, which is allocated to the core, and determines how many clusters are turned on in order to match performance required by the task. The other is software-based. Special instructions that turn on or off clusters are prepared. Programmers or compilers insert the instruction in each task. Practically, it is better that programmers do not have to determine how many cores are allocated to the task. They only have to declare performance the task requires, in other words the task size and the deadline time. One method to realize this is using some kind of annotations or functions like API. Compilers translate them into the special instructions that denote the number of active cores. Compatibility and transparency between different MultiCore processors is provided in source codes. The other is that the special instructions denote only required performance and hardware determines the number of active clusters. In this case, the compatibility and transparency is provided in binaries. Considering the use in embedded applications, we think the software-based method is practical.

3.3. Other applications

Multiple clustered core processors have a good characteristic in temperature awareness. By alternatively gating some of the clusters of the core during certain periods of time, power is reduced to cool down the core. This is called cluster hopping [4]. Figure 3 shows how the cluster hopping works. The black box means that the cluster is turned off.

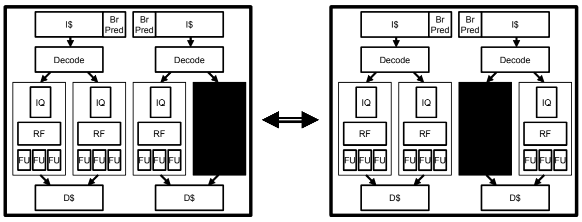

Figure 3. Cluster hopping

Similarly, by alternatively gating some of the cores of the processor during certain periods of time, power is reduced to cool down the chip. This is called core hopping. The core hopping resembles PE rotation [15] and Figure 4 shows how it works. The black box means that the core is turned off.

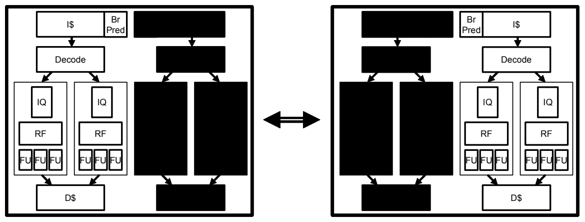

Figure 4. Core hopping

Another good characteristic of multiple clustered core processors is in dependability. One of the simple implementations for realizing dependability is to redundantly execute single program. Time redundancy or space redundancy can be utilized. MultiCore processors are very suitable for space redundancy. In order to check errorless, a single task is redundantly executed on multiple cores. When two outcomes for the single task (or every instruction in the task) do not match, an error is detected. Furthermore, multiple clustered core processors can change the dependability levels according to the importance of the current task. If the task is critical, it is duplicated and redundantly executed across multiple cores. If it is less critical, every instruction in the task is duplicated and redundantly executed across multiple clusters. In the latter case, some errors can not be detected, since some blocks in each core are not duplicated.

These are very interesting research topics, which we are currently investing.

4. Evaluations

4.1. Methodology

In order to evaluate the energy efficiency of the multiple clustered core processors, we perform a simulation study. We compare the energy consumed by the processors in Figure 1. First, we determine each core's configuration as shown in Table 1.

Table 1. Number of functional units

	IQ entries	Int units	FP units	Ld/St
Large core	36	5	5	3
Medium core	24	3	3	2
Small core	16	2	2	1
Clustered core	16 x 2	2 x 2	2 x 2	1 x 2

Second, based on the study in [3], the areas of the cores are estimated as shown in Table 2. Thus, the processors (a) to (e) in Figure 1 are 226.1mm^2, 237.8mm^2, 248.1mm^2, 258.4mm^2, and 201.6mm^2, respectively.

Table 2. Area estimations (mm^2)

	Large core	Medium core	Small core	Clustered (2 clusters)	Clustered (1 cluster)
D cache	26.0	13.0	5.2	13.0	13.0
I cache	20.8	10.4	5.2	10.4	10.4
TLB	10.1	4.4	1.9	4.4	4.4
Fetch, BrPred	6.7	4.5	2.9	4.5	4.5
Decode	2.5	1.7	1.1	1.7	1.7
OOO exec	54.6	24.1	10.1	20.2	10.1
RFs	12.8	5.9	2.9	5.8	2.9
Func units	30.8	12.9	6.5	13.0	6.5
Misc	2.4	2.4	2.4	2.4	2.4
Routing	59.3	39.5	26.4	26.4	26.4
Total	226.1	118.9	64.6	100.8	82.3

Surprisingly, the multiple clustered core processor is the smallest, while it has larger number of functional units than the dual core processor (Figure 1b) does. This is due to the complexity effective design in the clustered core. The complexity of instruction queue (IQ) is $O(N^2) – O(N^3)$ [14], where N is the issue width. The area in OOO exec of the clustered core is smaller than that of the medium core while the total number of the issue width is larger in the former than in the latter. This is because the issue width is limited inside each cluster in the clustered core. The complexity of operand bypassing is $O(N^2)$ [14], where N is the issue width, and thus the area in routing of the clustered core is smaller than that of the medium core.

Next, we estimate each core's performance. Following the study in [16], we determine that the large core's performance is 2400M instructions per second (IPS). The performance of the rest cores are determined based on the Pollack's rule [16]. Table 3 summarizes core performance. It is assumed that power consumed by each core is proportional to its area.

Table 3. Performance assumptions (IPS)

Large core	Medium core	Small core	Clustered (2 clusters)	Clustered (1 cluster)
2400[5]	1740	1283	1602	1448

And last, we assume tasks executed on the processors. Following the study in [16] again, we use a task mix shown in the left side of Table 4. We call it Task Mix #1. Each task is randomly generated every 1000 clock cycles. When 100 tasks are executed, a simulation is finished. As you can see later, some processors can not satisfy the worst case execution time (WCET) listed in Table 4. Hence, we use another task mix by relaxing some WCET. We call it Task Mix #2.

Table 4. Task mixes

	Mix #1[5]		Mix #2	
	Instructions	WCET	Instructions	WCET
Task 1	45M	0.030sec	45M	0.035sec
Task 2	150M	0.250sec	150M	0.250sec
Task 3	800M	1.000sec	800M	1.000sec
Task 4	2000M	1.500sec	2000M	1.600sec

4.2. Results

Simulation results are presented in Figure 5. We show total energy consumption. Energy consumed by each MultiCore processor is normalized by that consumed by the uniprocessor (a in Figure 5). In the case of Task Mix #1, the triple core (c in Figure 5) and quad core (d in Figure 5) processors can not satisfy the WCET. This means the small core does not have enough performance. When we compare the multiple clustered core processor (e in Figure 5) with the dual core processor (b in Figure 5), the multiple clustered core processor consumes less energy than the dual core processor. When some WCET is relaxed, all processors satisfy the constraints. In the case of Task Mix #2, the triple core and the quad core processors consume less power than the multiple clustered core processor. However, the difference is smaller than that between the dual core processor and the multiple clustered core processor. From these observations, the multiple clustered core processor is a good solution to attain high performance and low power, under the requirements of easy programmability.

5. Conclusions

In this paper, we proposed multiple clustered core processors. They solve the problem of programming difficulties. According to the task size that is allocated to a core, the core resizes the number of clusters. Thus, programmers do not have to concern task allocation problems. Preliminary evaluations showed that the dual clustered core processor is as energy efficient as the heterogeneous MultiCore processor. In addition, the dual clustered core processor solved the severe task scheduling problem that the heterogeneous MultiCore processor could not.

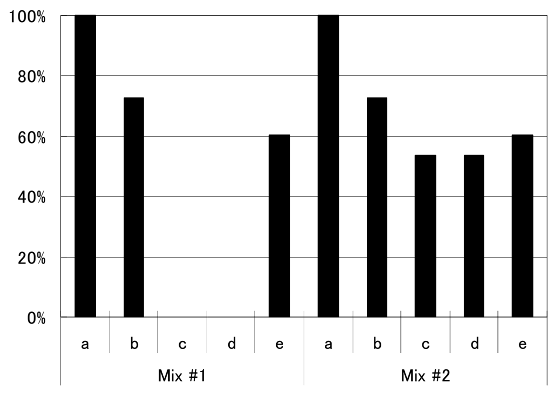

Figure 5: Normalized energy consumption

One of the future studies regarding multiple clustered core processors is more detailed evaluation using commercially distributed application programs. It might unveil some problems on implementing multiple clustered core processors. The other is investigating algorithms for activity migration both between clusters and between cores. It improves temperature-awareness of multiple clustered core processors. We are also interested in dependable issues. Exploiting redundancy found in MultiCore processors will be a complexity- and cost-effective solution for dependability.

Acknowledgements

This work is partially supported by Grants-in-Aid for Scientific Research #16300019 and #176549 from Japan Society for the Promotion of Science.

References

[1] R. I. Bahar and S. Manne, Power and Energy Reduction via Pipeline Balancing, 28th International Symposium on Computer Architecture, 2001.

[2] R. Balasubramonian, S. Dwarkadas, and D.H. Albonesi, Dynamically Managing the Communication-Parallelism Trade-off in Future Clustered Processors, 30th International Symposium on Computer Architecture, 2003.

[3] J. Burns and J.-L. Gaudiot, Area and System Clock Effects on SMT/CMP Throughput, IEEE Transactions on Computers, Vol.54, No.2, 2005.

[4] P. Chaparro, J. Gonzalez, and A. Gonzalez, Thermal-Aware Clustered Microarchitectures, 22nd International Conference on Computer Design, 2004.

[5] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, The Multicluster Architecture: Reducing Cycle Time Through Partitioning, 30th International Symposium on Microarchitecture, 1997.

[6] J. Gonzalez and A. Gonzalez, Dynamic Cluster Resizing, 21st International Conference on Computer Design, 2003.

[7] M. Hagiwara, I. Minematsu, T. Yamashita, Y. Komatsu, T. Fujimoto, T. Tsukada, and K. Ishibashi, A Low-Power Processor Based on Symmetric Multi-CPU Architecture for SoCs, 8th IEEE Symposium on Low-Power and High-Speed Chips, 2005.

[8] P. Hofstee, Power Efficient Processor Architecture and the Cell Processor, 11th International Symposium on High-Performance Computer Architecture, 2005.

[9] R. Kalla, B. Sinharoy, and J. M. Tendler, IBM POWER5 Chip: A Dual-Core Multithreded Processor, IEEE Micro, Vol.24, No.2, 2004.

[10] R. E. Kessler, The Alpha 21264 Microprocessor, IEEE Micro, Vol.19, No.2, 1999.

[11] P. Kongetira, K. Aingaran, and K. Olukotun, Niagara: A 32-Way Multithreded SPARC Processor, IEEE Micro, Vol.25, No.2, 2005.

[12] R. Kumar, K. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, Single-ISA Heterogeneous Multi-core Architectures: the Potential for Processor Power Reduction, 36th International Symposium on Microarchitecture, 2003.

[13] C. McNairy and R. Bhatia, Montecito: A Dual-Core, Dual-Threaded Itanium Processor, IEEE Micro, Vol.25, No.2, 2005.

[14] S. Palacharla, N. P. Jouppi, and J. E. Smith, Complexity-Effective Superscalar Processors, 24th International Symposium on Computer Architecture, 1997.

[15] H. Sato and T. Sato, A Preliminary Evaluation on Energy Efficiency of a Temperature-aware Multicore-processor, 2nd Workshop on Temperature Aware Computer Systems, 2005.

[16] Y. Takatsukasa, K. Kobayashi, and H. Onodera, Dynamic Voltage and Frequency Scaling Technologies for Heterogeneous Multi-Processor Architecture in Future Nanometer Technologies, 12th Workshop on Synthesis and System Integration of Mixed Information Technologies, 2004.