Dependability, Power, and Performance Trade-off on a Multicore Processor

Sato, Toshinori
System LSI Research Center, Kyushu University

Funaki, Toshimasa
Department of Artificial Intelligence, Kyushu Institute of Technology

出版情報：Asia and South Pacific Design Automation Conference. 13, pp.714-719, 2008-01-24
バージョン：
権利関係：

Dependability, Power, and Performance Trade-off on a Multicore Processor

Toshinori Sato
System LSI Research Center
Kyushu University
toshinori.sato@computer.org

Toshimasa Funaki
Department of Artificial Intelligence
Kyushu Institute of Technology
t-funaki@klab.ai.kyutech.ac.jp

Abstract - As deep submicron technologies are advanced, we face new challenges, such as power consumption and soft errors. A naïve technique, which utilizes emerging multicore processors and relies upon thread-level redundancy to detect soft errors, is power hungry. It consumes at least two times larger power than the conventional single-threaded processor does. This paper investigates a trade-off between dependability and power on a multicore processor, which is named multiple clustered core processor (MCCP). It is proposed to adapt processor resources according to the requested performance. A new metric to evaluate a trade-off between dependability, power, and performance is proposed. It is the product of soft error rate and the popular energy-delay product. We name it *energy, delay, and upset rate product* (EDUP). Detailed simulations show that the MCCP exploiting the adaptable technique improves the EDUP by up to 21% when it is compared with the one exploiting the naïve technique.

I Introduction

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 have been already shipped. Since processor performance is proportional to the square root of its area while its power consumption is proportional to the area, multicore processors are a good solution for power efficiency.

On the other hand, advanced semiconductor technologies increase soft error rate (SER) [4, 10]. With the reduction in transistor size, the area per bit scales down. In order to prevent breakdown caused by high electric field, the supply voltage also scales down. Hence, the node charge reduces and the bit cell is easy to flip by cosmic ray and alpha particles. Since each bit cell becomes small so that probability that some particles such as neutrons hit the cell will also become small, resulting in the net effect of almost constant SER per bit. Since the number of transistors per chip has been tremendously increased, SER per chip is also exponentially increasing.

In order to detect (and if possible to correct) faults due to single event upsets (SEU), redundant execution of a single program is proposed [7, 14, 17, 18]. The increase in the popularity of multicore processors is favorable to the redundant execution. A single program is duplicated and its two redundant copies are executed simultaneously in the different cores on a multicore processor. When two outcomes for the single program do not match, an SEU is detected. This redundant threading (RT) technique is a very simple and effective way to provide dependability. However, unfortunately, it consumes at least two times larger power than the conventional single-threaded processor does.

In order to solve the power consumption in the RT, we propose an adaptable RT technique. We are currently studying an adaptable multicore processor, which we call multiple clustered core processor (MCCP) [16]. It is based on the clustered microarchitecture [11] and makes a good trade-off between power and performance. We exploit the characteristic of the MCCP to improve power efficiency of the RT. In other words, it also makes a trade-off between dependability and power.

The rest of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces the MCCP, and proposes the adaptable RT technique that considers the trade-off between dependability and power. Section 4 presents evaluation results. And Section 5 provides conclusions.

II. Related Work

One of the simple implementations for providing dependability is redundant executions. Time redundancy or space redundancy can be utilized. The conventional multicore processors are very suitable for exploiting space redundancy in thread level [7, 14, 17, 18]. In order to check errorless, a single program thread is duplicated and the two redundant copies of the single thread are executed simultaneously on a multicore processor. When two outcomes for the single thread do not match, a fault is detected. Slipstream processor [18] is a multicore processor, and two redundant threads are executed on separate processor cores. It can not detect all single transient faults because it does not duplicate all instructions from a single thread. On the other hand, CRT [14], realized as a multicore processor, achieves lockstepping and CRTR [7] enhances CRT with recovery mechanism. Shimamura et al. [17] developed a fail-safe multicore processor with memory data comparison feature.

The clustered microarchitecture is a solution to solve the wire delay problem [11]. A large processor core is divided into multiple clusters. Each cluster is small so 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 [3] 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. The reduction in issue width eliminates useless power consumption.

III. Multiple Clustered Core Processors

MCCP [16] is shown in Fig. 1. It is a homogeneous multicore processor. The difference from the conventional homogeneous multicore processors is that it consists of multiple clustered cores rather than monolithic ones. Each core is based on the clustered microarchitecture [11]. Figure 1 shows an MCCP with two homogeneous clustered cores, each of which has two identical clusters. In the figure, each cluster consists of instruction scheduling queue (IQ), register files (RF), and functional units (FU). Instruction and data caches (I$ and D$), branch predictor (BrPred) and decoder (Decode) are shared by all clusters in a core. We exploit the clustered microarchitecture combined with multicore architecture in order to make a trade-off between power and performance [16].

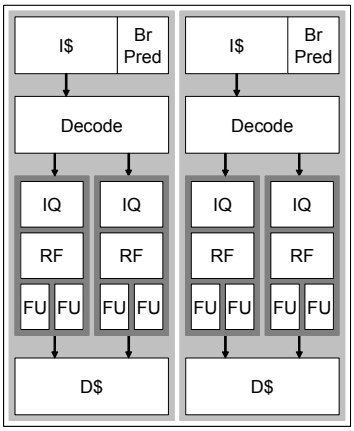

Fig. 1. Multiple Clustered Core Processor

A. Power-Performance Trade-off Issue

Multicore processors are a promising solution that achieves high performance with low power consumption. Figure 2 shows different types of multicore processors. Figure 2a is a uniprocessor. Figures 2b and 2d are homogeneous multicore processors, while Fig. 2c is a heterogeneous one. As you can see, the heterogeneous multicore processor consists of several cores with different scales in area and in performance. When a thread requires high performance but it does not have large parallelism in it, a large core serves. When the other thread 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 thread, 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 [1, 12].

Unfortunately, heterogeneous multicore processors are complex to design and are difficult to program. The MCCP exploits the clustered microarchitecture to realize heterogeneity on the homogeneous multicore processor [16].

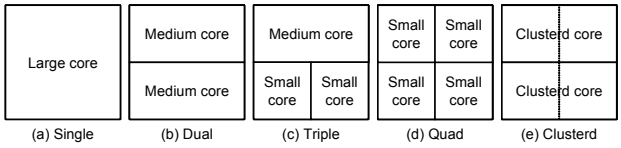

Fig. 2. Different Types of Multicore Processors

In order to attain the goal, we propose cluster gating [16]. Figure 3 explains how it works. This is a dual core MCCP consisting of two dual cluster cores. Figure 3a shows a homogeneous dual core processor consisting of large cores. When high performance is not required, some clusters are turned off, as shown in Fig. 3b. 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 thread is satisfied. Now, we have a heterogeneous dual core processor. Figure 3c shows a dual core processor consisting of small cores, in both of which one cluster is turned off.

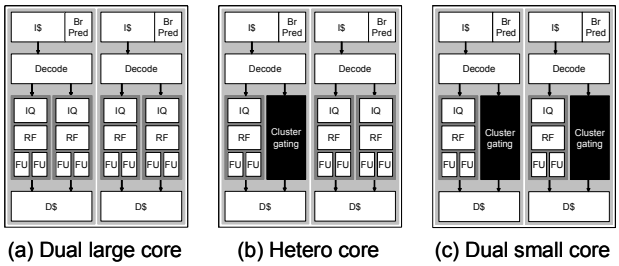

Fig. 3. Cluster Gating

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 thread allocation. We can always get a desirable scaled core. Providing several execution mode enables to consume power just enough for the required performance. The concept of the MCCP is extended into multi-performance processors for low-power embedded applications [15].

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 thread, which is allocated to the core, and determines how many clusters are turned on in order to match performance required by the thread. The other is software-based. Special instructions that turn on or off clusters are prepared. Programmers or compilers insert the instruction in each thread. Practically, it is better that programmers do not have to determine how many clusters are allocated to the thread. They only have to declare performance the thread requires. 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 clusters. Compatibility and transparency between different multicore processors are 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 are provided in binaries. This issue remains for the future study.

B. Dependability-Power Trade-off Issue

The MCCP has a good characteristic in its dependability, as shown in Fig. 4. It can utilize the RT technique since it is a multicore processor. A single thread is duplicated and is redundantly executed across multiple cores. As mentioned above, the naïve RT technique consumes two times larger power than the conventional single-threaded processor does. A simple way to reduce power consumed by the RT technique is to use a small processor core. However, it is easily expected that such a technique degrades processor performance, and hence it might increase energy consumption due to long execution time.

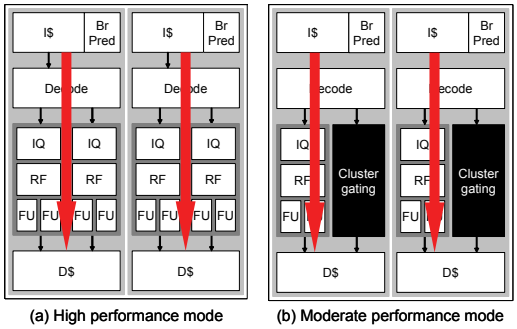

Fig. 4. Dependability Modes

In order to achieve both low power consumption and high performance on a dependable processor, we propose to utilize the adaptability of the MCCP. We propose to switch the mode of the MCCP between the dual large core mode and the dual small core mode according to required performance. When high performance is required, the dual large core mode is selected. Here, we call it *high performance mode*. Otherwise, we switch into the dual small core mode, which we call *moderate performance mode*. The two modes are depicted in Fig. 4. Since both modes redundantly execute a single program, they equally provide dependability. We do not consider the heterogeneous core mode shown in Fig. 3 (b). This is because the small core determines the execution time of the program and thus the large core will waste power consumption. Improving the small core's performance by utilizing execution results of the large core is an interesting research topic and it remains for the future study.

The main topic of the present paper is how to choose the dependability mode. One of the strategies relies upon programmers. A programmer marks how important every thread is and tells it to hardware (processor) using annotations. Another strategy is OS-based. OS marks the importance of every thread using some metric; for example, deadline time. In this paper, we propose a fully transparent hardware-based strategy.

The amount of instruction level parallelism (ILP) varies between application programs. A processor in the high performance mode wastes power consumption when ILP in the application program is small. On the contrary, the processor in the moderate performance mode diminishes its performance when ILP is large. Furthermore, the amount of ILP even within a single application program varies by more than a factor of two [3]. Figure 5 shows an example of the issue rate for SPEC2000 CINT benchmark gcc running on a dual-cluster core. The details of the core can be found in Section IV.A. The horizontal axe indicates the execution cycles and the vertical one represents the average number of instructions issued per cycles (issue IPC) over a window of 10,000 execution cycles. The issue IPC varies by more than a factor of two over a million cycles of execution. If a processor is in the high performance mode, it wastes power during low issue IPC. On the contrary, if a processor is in the moderate performance mode, performance is severely degraded during high issue IPC. These variations can be exploited to determine the dependability mode.

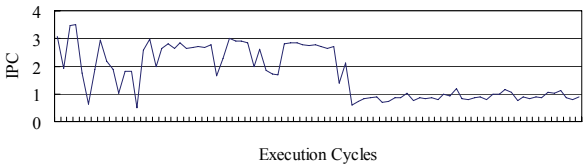

Fig. 5. Issue IPC Variation for gcc

As explained above, the high performance mode wastes power consumption during low issue IPC and the moderate performance mode degrades performance during high issue IPC. The observations lead us to switch between two modes according to requested performance. The MCCP utilizes the high performance mode only when issue IPC is high and similarly to utilize the moderate performance mode only when issue IPC is low, as shown in Figure 6. When issue IPC is low, there are idle execution resources and thus the moderate mode provides dependability without serious performance loss. In addition, since some clusters are occasionally turned off, the wasted power consumption is eliminated.

We assume that past program behavior indicates future behavior. Hence, based on past issue IPC, future issue IPC could be predicted. In order not to use a floating-point divider, we measure the number of instructions issued over a fixed sampling window. We predict future issue IPC based on the past number of issued instructions rather than on past issue IPC. We use predicted issue IPC for the mode selection. If it is smaller than a predetermined threshold value (Th2m) in the high performance mode, the MCCP switches into the moderate performance mode. Similarly, if predicted issue IPC is larger than another predefined threshold value (Tm2h) in the moderate performance mode, the MCCP switches into the high performance mode.

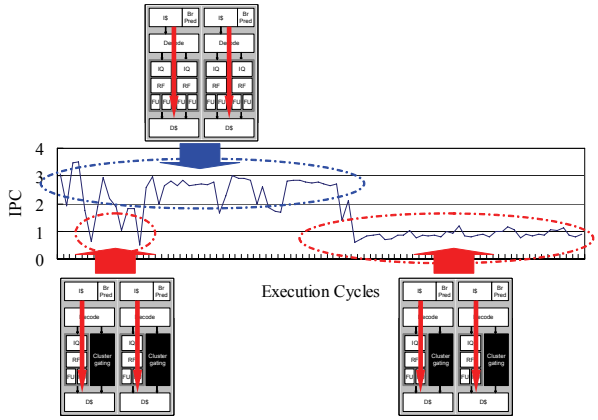

Fig. 6. IPC-directed Mode Switching

C. Energy, Delay, and Upset rate Product

In order to evaluate the trade-off between dependability, power, and performance, we need a metric. The energy-delay product (EDP) [9] is a popular metric for evaluating the trade-off between power and performance. We extend it to consider dependability as well as power and performance. SER or upset rate is a popular metric for evaluating dependability. Hence, we propose to product the EDP and the upset rate. We name it *energy, delay, and upset rate product* (EDUP).

Another possible metric for evaluating the trade-off between dependability and performance is MITF (mean instructions to failure) [19]. We are currently studying to extend the MITF to evaluate the trade-off on multicore processor [6]. The EDUP will support the MITF when power consumption is considered.

IV. Evaluation

A. Methodology

SimpleScalar/PISA tool set [2] is used for architectural-level simulation. We use the MCCP consisting of two cores, each of which consists of two clusters. Based on the study in [5], the configurations of one processor core are determined as shown in TABLE I. The front-end and L1 caches are shared by two clusters in a core. L2 cache is shared by two cores. The difference from [5] is that this study uses smaller L1 instruction and data caches (16KB each) than [5] does (64KB each). We use the threshold values of 2.0 and 1.6 for Th2m and Tm2h, respectively. These values were determined based on preliminary simulations, where Th2m was varied between 1.6 and 2.0, Tm2h was varied between 1.0 and 1.8, and all combinations of Th2m and Tm2h were considered. The overhead of synchronizing two cores to compare results from them is not included in the evaluations.

We estimate upset rate as the product of area and soft error rate per bit. Also based on [5], the areas of the core are estimated as shown in TABLE II [16]. The difference can be seen in L1 caches. Using the values in TABLE II, we can estimate the upset rates of the high performance and moderate performance modes. Since the mode switching does not affect on the outside of the cores, we ignore the L2 cache, the miscellaneous, the coherence unit, and the I/O from estimating the upset rate. Based on the considerations, we can see the upset rate of the moderate performance mode is 72% of the high performance mode.

TABLE I
Processor Core Configurations

Fetch width	8 instructions
L1 instruction cache	16K, 2 way, 1 cycle
Branch predictor	1K-gshare + 512- BTB
Dispatch width	4 instructions
Instruction window size	16 entries / cluster
Issue width	2 instructions / cluster
Commit width	4 instructions / cluster
Integer ALUs	2 units / cluster
Integer multiplires	2 units / cluster
Floating ALUs	2 unit / cluster
Floating multiplires	2 unit / cluster
L1 data cache ports	1 ports / cluster
L1 data cache	16K, 2 way, 1 cycle
Unified L2 cache	512K, 2 way, 10 cycles
Memory	Infinite, 100 cycles

TABLE II
Area Estimation (mm^2)

16K L1 data cache	2.6
16K L1 instruction cache	2.6
TLB	4.4
Fetch unit	1.3
Branch predictor	3.2
Decoder	1.7
OOO execution unit	10.1 / cluster
Register files	2.9 / cluster
Functional units	6.5 / cluster
Misc	2.4
Routing	26.4
512 L2 cache	110.0
Misc	6.1
Coherence unit	6.3
I/O	13.7

Based on [8], we estimate power consumed by each component. We found that power consumed by the moderate performance mode is 81% of that consumed by the high performance mode.

Six programs from SPEC2000 CINT and eight programs from MediaBench [13] are used. For each SPEC program, 1B instructions are skipped before actual simulation begins. After that each program is executed for 2B instructions. For MediaBench, each program is executed from beginning to end. We do not count NOP instructions. We vary the sampling window among 100, 1,000, and 10,000 cycles.

B. Results

Figure 7 presents how frequently two modes are selected in SPEC benchmark programs. For each group of three bars,

the left one indicates the breakdown of the execution cycles for the 100-cycle window, the center one is for the 1,000-cycle window, and the right one is for the 10,000-cycle window. Each bar is divided into two parts. The bottom one indicates the percentage of cycles where the high performance mode is selected, and the top one indicates the percentage of cycles where the moderate performance mode is selected.

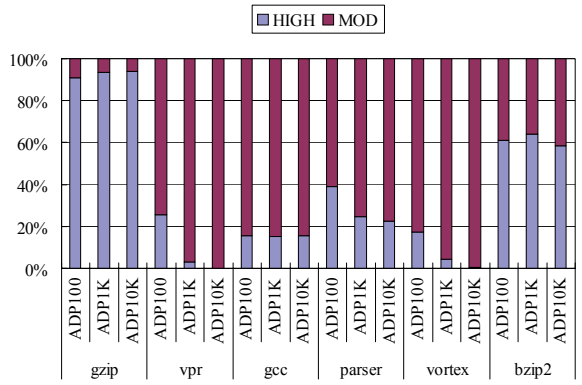

Fig. 7. Breakdown of Dependability Mode (SPEC2000)

As can be easily seen, SPEC benchmark programs evaluated in this study are classified into two groups. One consists of gzip and bzip2, and the other consists of vpr, gcc, parser, and vortex. The first group prefers the high performance mode. In contrast, the second one prefers the moderate performance mode. The results explain that the MCCP efficiently captures the characteristics of each program and adopt itself. It is also observed that the short sampling window selects the high performance mode more frequently than the long sampling window does.

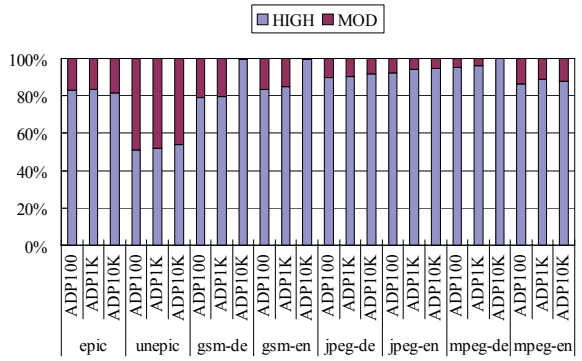

Fig. 8. Breakdown of Dependability Mode (MediaBench)

Figure 8 presents how frequently two modes are selected in MediaBench programs. The layout is the same to that of Fig. 7. Different from SPEC benchmark programs, almost all programs in MediaBench prefer the high performance mode. Only unepic selects both modes equally. It is also different from SPEC benchmark that the short sampling window selects the moderate performance mode more frequently than the long sampling window. From these observations, we found that SPEC benchmark and MediaBench has the absolutely different characteristics in their performance. This observation is obtained since the mode selection policy is based on the program's issue IPC in a fixed sampling window.

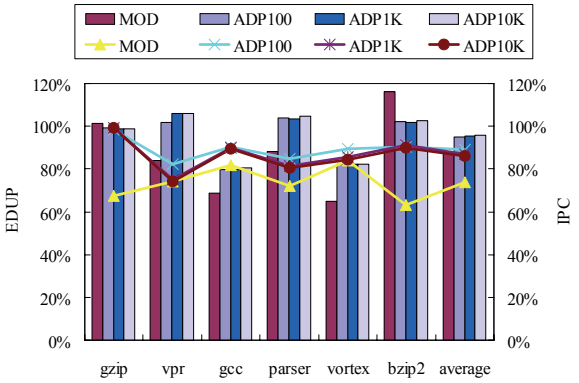

Fig. 9. Relative IPC and EDUP (SPEC2000)

Figure 9 shows the commit IPC and the EDUP for SPEC benchmark programs. Line graphs present the commit IPC and bar graphs present the EDUP. The bottom line graph indicates the commit IPC of the processor that is always in the moderate performance mode (hereafter we call the processor *the always moderate-performance*). The remaining line graphs are for the processor that utilizes the adaptable RT technique with different sampling window size. For each group of four bars, the first bar (see from left to right) indicates the EDUP of the always moderate-performance. The remaining bars are for the processor that utilizes the adaptable RT technique with different sampling window sizes. Every value is normalized by the corresponding value of the processor that is always in the high performance mode (hereafter we call the processor *the always high-performance*).

First, it is observed that processor performance is significantly degraded if the moderate performance mode is only utilized. The performance loss is as much as 38% and an average of 27%. In contrast, the adaptable RT technique mitigates the performance loss. It is approximately 10% for all sampling window sizes.

Second, the adaptable RT technique improves the EDUP as much as 21% and an average of 5%. We can not see considerable differences between the evaluated sampling window sizes. In four of six programs, the always moderate-performance shows better EDUP than the adaptable RT technique does. However, it should be noted that it suffers serious EDUP degradation in bzip2. In contrast, the adaptable RT technique maintains the almost same EDUP to that of the always high-performance.

Figure 10 shows the commit IPC and the EDUP for MediaBench. The layout is the same to that of Fig. 9. The always moderate-performance seriously degrades both

performance and the EDUP. In contrast, the adaptable RT technique keeps comparable performance and the EDUP to the always high-performance, while it can not improve the EDUP in the case of MediaBench.

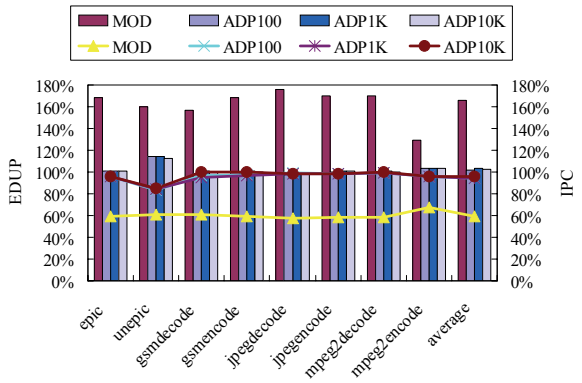

Fig. 10. Relative IPC and EDUP (MediaBench)

VI. Conclusions

The aggressively advanced semiconductor technologies unveil the problems caused by soft errors. While emerging multicore processors are suitable for soft error tolerance, they consume large power when the redundant threading (RT) technique is utilized. This paper proposed an adaptable RT technique for making a good trade-off between dependability and power. It exploits an adaptable characteristic of the multiple clustered core processor (MCCP). This paper also proposed a metric for evaluating the trade-off, energy, delay, and upset rate product (EDUP). Detailed simulations showed that the MCCP with the adaptable RT technique improves the EDUP by up to 21%.

Acknowledgement

This work is partially supported by Grant-in-Aid for Scientific Research (KAKENHI) (A) # 19200004 from Japan Society for the Promotion of Science, and by the CREST programs of Japan Science and Technology Agency.

References

[1] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's Law through EPI Throttling," 32nd International Symposium on Computer Architecture (2005)

[2] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an Infrastructure for Computer System Modeling," IEEE Computer, Vol.35, No.2 (2002)

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

[4] S. Borker, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," IEEE Micro, Vol. 25, No. 6 (2005)

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

[6] T. Funaki and T. Sato, "Dependability-Performance Trade-off on Multiple Clustered Core Processors," 4th International Workshop on Dependable Embedded Systems (2007)

[7] M Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, "Transient-Fault Recovery for Chip Multiprocessors," 30th International Symposium on Computer Architecture (2003)

[8] M. K. Gowan et al., "Power Considerations in the Design of the Alpha 21264 Microprocessor," 35th Design Automation Conference (1998)

[9] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design," Symposium on Low Power Electronics (1994)

[10] T. Karnik, P. Hazucha, and J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes," IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 2 (2004)

[11] R. E. Kessler, "The Alpha 21264 Microprocessor," IEEE Micro, Vol. 19, No. 2 (1999)

[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. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a Tool for Evaluating and Synthesizing Multimedia and Communications Systems," 30th International Symposium on Microarchitecture (1997)

[14] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed Design and Evaluation of Redundant Multithreading Alternatives," 29th International Symposium on Computer Architecture (2002)

[15] Y. Oyama, T. Ishihara, T. Sato, and H. Yasuura, "A Multi-Performance Processor for Low Power Embedded Applications," 10th Symposium on Low-Power and High-Speed Chips (2007)

[16] T. Sato and A. Chiyonobu, "Multiple Clustered Core Processors," 13th Workshop on Synthesis and System Integration of Mixed Information Technologies (2006)

[17] K. Shimamura, T. Takehara, Y. Shima, and K. Tsunedomi, "A Single-Chip Fail-Safe Microprocessor with Memory Data Comparison Feature," 12th Pacific Rim International Symposium on Dependable Computing (2006)

[18] K. Sundaramoorthy, Z. Purser, and E. Rotenberg, "Slipstream Processors: Improving Both Performance and Fault Tolerance," 9th International Conference on Architectural Support for Programming Languages and Operating Systems (2000)

[19] C. T. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Reducing the Soft-Error Rate of a High-Performance Microprocessor," IEEE Micro, Vol. 24, No. 6 (2004)