Formulating MITF for a Multicore Processor with SEU Tolerance

Funaki, Toshimasa
Kyushu Institute of Technology

Sato, Toshinori
Fukuoka University | Kyushu University | JST, CREST

出版情報：Euromicro Conference on Digital System Design. 11, pp.234-241, 2008-09-03
バージョン：
権利関係：

Formulating MITF for a Multicore Processor with SEU Tolerance

Toshimasa Funaki
Kyushu Institute of Technology

Toshinori Sato
Fukuoka University
Kyushu University
JST, CREST
toshinori.sato@computer.org

Abstract

While shrinking geometries of embedded LSI devices is beneficial for portable intelligent systems, it is increasingly susceptible to influences from electrical noise, process variation, and natural radiation interference. Even in consumer applications, modern embedded devices should be protected by dependable technologies. The challenging issue is there is a severe constraint in power consumption. As a platform to investigate the dependability, power, and performance trade-off, multiple clustered core processor (MCCP) is being investigated. It is a homogeneous multicore processor and has configurability in scale, which is beneficial for considering the trade-off. This paper focuses on how to explore the trade-off, and proposes to use mean instructions to failure (MITF) as a metric. To the best of our knowledge, this is the first study that formulates MITF for multicore processors. We compare three redundancy modes; undependable, thread-level redundancy, and instruction-level redundancy modes based on detailed simulations. As expected, thread-level redundancy shows largest MITF.
Keywords: soft error problem, thread-level redundancy, instruction-level redundancy

1. Introduction

Advanced semiconductor technologies increase soft error rate (SER) [1, 2]. 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. These trends have increased SER per bit by 8% every generation [2]. In addition, the number of transistors per chip has been tremendously increased, and hence SER per chip is also exponentially increasing.

Exploiting redundancy is a conventional way to detect and then to correct faults caused by single event upsets (SEU). The redundant execution of a single program [3, 4, 5, 6] benefits from the current trend of multicore processors. A single program is replicated and its redundant copies are executed simultaneously in different cores on a multicore processor. When two outcomes for the single program do not match, an SEU is detected. A recovery mechanism is provided for recovering the processor state to a safe point where the SEU is detected. This utilizes space redundancy or thread-level redundancy.

We can also exploit instruction-level redundancy in a single processor [7, 8, 9]. Detecting SEU's is possible by redundantly executing every instruction in the program, and two results for the instruction are compared with each other. If they do not match, an SEU is detected. Also in this case, another recovery mechanism is provided for the processor state recovery. Two techniques are proposed for utilizing instruction-level redundancy; one is based on space redundancy [7, 9] and the other is based on time redundancy [8]. In the former case, every instruction is replicated and its redundant copies are executed simultaneously in the same processor core. In the latter case, every instruction is redundantly executed at least twice. This paper focuses on the former one; space redundancy.

There is a trade-off between dependability and performance. For example, up to two times performance gain is expected when we use two processor cores for parallel execution rather than for redundant execution. On the other hand, there is another trade-off between power and performance. Thread-level redundancy consumes large power since at least two cores serve for a single program, while it achieves high performance. In contrast, instruction-level redundancy diminishes performance since the number of instructions executed in a single core is increased, while it consumes less power than thread-level redundancy does. Given the above considerations, the ultimate goal of this ongoing study is to achieve high dependability and high performance with low power consumption. The trade-off among three features; dependability, performance, and power should be investigated. As a platform to study toward the goal, we proposed multiple clustered core processors (MCCP's) [10]. This paper focuses on the trade-off between the first two features on the MCCP, and tries to establish a metric

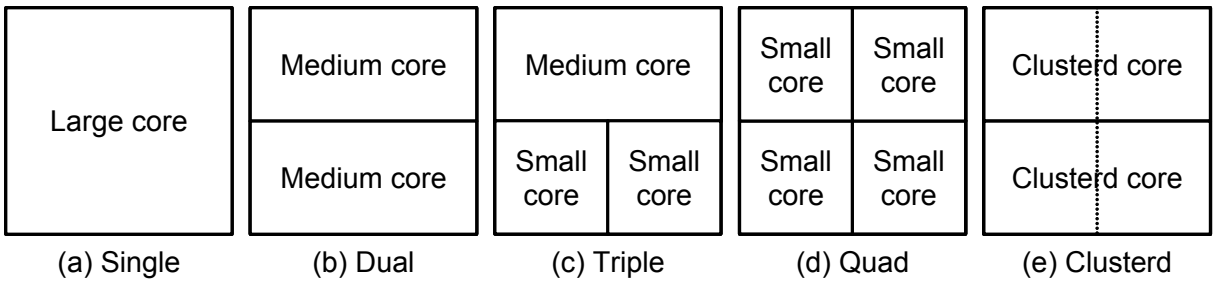

Figure 1. Different Types of Multicore Processors

for evaluating the trade-off. The trade-off including power consumption is also considered, while proposing a metric for evaluating the trade-off remains for a future study.

The rest of this paper is organized as follows. Section2 summarizes related work. Section 3 introduces the MCCP. Section 4 examines the trade-off between dependability and performance. Section 5 introduces evaluation methodology. Section 6 presents simulation results. And Section 7 provides conclusions.

2. Related Work

One of the simple implementations for providing dependability is redundant executions. Time redundancy or space redundancy can be utilized. Sohi et al. [7] and Ray et al. [9] duplicate and simultaneously issue every instruction. Similarly, Sato et al. [8] execute every instruction twice transparently in a dynamically scheduled processor. Their proposed techniques detect faults by comparing two results which originate from a single instruction. Hence, redundancy is exploited in instruction level. In contrast, the conventional multicore processors are very suitable for exploiting space redundancy in thread level [3, 4, 5, 6]. In order to detect the occurrence of errors, a single program thread is duplicated and its two redundant copies of the single thread are executed simultaneously on a multicore processor. When two outcomes for the single thread (or those for every instruction in the thread) do not match, a fault is detected. Slipstream processor [4] 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 [5], realized as a multicore processor, achieves lockstepping and CRTR [6] enhances CRT with recovery mechanism. The MCCP utilizes redundancy both in instruction level and in thread level.

Considering Amdahl's law, extracting instruction-level parallelism from parallel and multithreaded applications is still important even in multicore era. Heterogeneous multicore processors [11, 12, 13] consist of several cores different in size and performance. They could match a variety of requirements from sequential and parallel applications. Unfortunately, however, heterogeneity hurts scalability in parallel applications [13]. Flexibility in scale will solve this problem. Core fusion [14] logically synthesizes a powerful single-threaded core from multiple small cores. Flexible Heterogeneous MultiCore (FMC) processor [15] reconfigures itself to fit application requirements by building a large core of shared resources. Composable Lightweight Processor (CLP) [16] allows multiple simple cores to form a powerful single-threaded processor. The MCCP chooses a different concept to provide flexibility. In order to match application requirements in power and performance, the MCCP relies on scaling-down strategy rather than scaling-up one.

The clustered microarchitecture is a solution to solve the wire delay problem [17]. 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 [18] 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. The MCCP exploits the clustered microarchitecture to realize heterogeneity on homogeneous multicore processors.

3. Multiple Clustered Core Processor

Multicore processors are a promising solution that achieves high performance with low power consumption. Figure 1 shows different types of multicore processors. Figure 1(a) is a uniprocessor. Figures 1(b) and 1(d) are homogeneous multicore processors, while Figure 1(c) is a heterogeneous one. As you can see, the heterogeneous multicore processor consists of several cores with

different scales in size and performance. When an application requires high performance but it does not have large parallelism in it, a large core serves. When the other application 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 application, 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, it is already known heterogeneous multicore processors consisting of cores with different scales are a good solution [11, 12].

3.1. Issues in heterogeneity

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 executed in series. 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 a 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 proposed the MCCP. We can provide heterogeneity for a homogeneous multicore processor.

3.2. Overview of MCCP

The MCCP is proposed as a platform for embedded applications, which require high performance, high efficiency in power, and high dependability [10]. Figure 2 depicts an example of the MCCP, which is a homogeneous multicore processor. There is an important difference from the conventional homogeneous multicore processors. It adopts clustered datapath rather than monolithic ones. Each core is based on the clustered microarchitecture [17]. The MCCP shown in Figure 2 has two homogeneous clustered cores, each of which has two identical clusters. You can choose any numbers of cores and of clusters you like. In this example, each cluster consists of instruction queue (IQ) for dynamic scheduling, register files (RF) for keeping operands, and functional units (FU) for execution. Instruction and data caches (I$ and D$), branch predictor (BrPred) and decoder (Decode) are shared by the clusters. L2 cache (L2$) is shared by the cores.

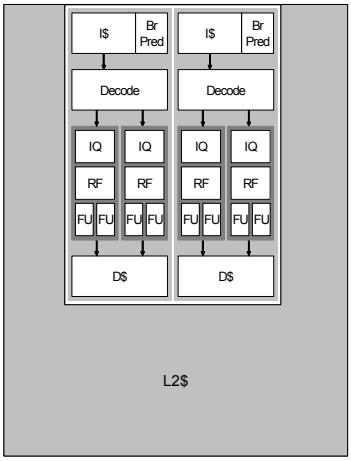

Figure 2. Multiple Clustered Core Processor

3.3. Power-performance trade-off

Adopting the clustered datapath provides the MCCP with configurability in the processor. We proposed cluster gating and core gating [10]. Figure 3 explains how the cluster gating and core gating work. L2 cache is not shown in the figure. This is a dual-core MCCP consisting of two dual-cluster cores. Figure 3(a) shows a homogeneous dual large-core processor. When high performance is not required, some clusters are turned off by cluster gating, as shown in Figure 3(b). The black box means that the cluster is turned off. This is a heterogeneous dual-core processor. Figure 3(c) shows a dual small-core processor, in both cores of which one cluster is turned off. When performance is still over that required, one core is turned off, as shown in Figure 3(d). The black box means that the core is turned off. Using the core gating, only a small number of cores are active so that just requested performance is satisfied. As shown in Figure 3(e), both cluster gating and core gating can be combined to configure a single small-core processor.

The MCCP benefits on power efficiency from the configurability. Providing several scaling in datapath enables to consume power just enough for the required performance. Our previous study [10] shows that the MCCP attains equivalent performance to the conventional homogeneous and heterogeneous multicore processors

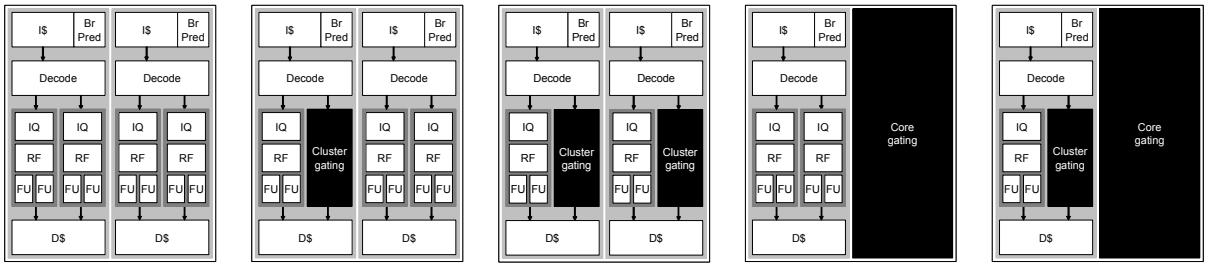

Figure 3. Multiple Configurations in MCCP

with less power consumption. The concept of the clustered core is extended into multi-performance processors for low-power embedded applications [19].

3.4. Redundancy modes

The MCCP has a good characteristic in its dependability. One of the simple implementations for providing dependability is to redundantly execute a single program. Time redundancy or space redundancy can be utilized. The conventional multicore processors are very suitable for exploiting space redundancy for dependability. In order to detect the occurrence of errors, a single thread is redundantly executed on multiple cores. When two outcomes for the single thread (or those for every instruction in the thread) do not match, an SEU is detected.

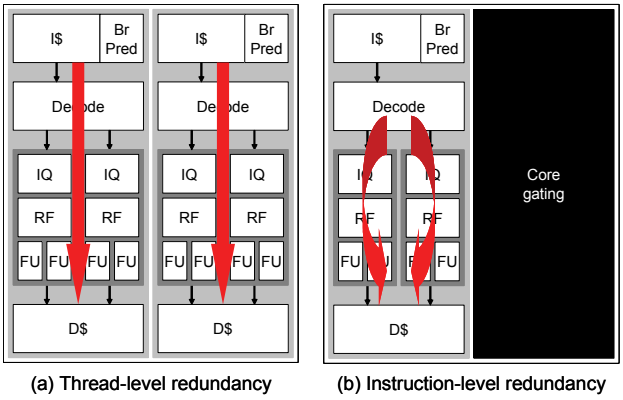

Figure 4. Two Redundancy Modes of MCCP

The configurability also provides the MCCP with multiple redundancy modes. Figure 4 shows some examples. Figure 4(a) depicts the thread-level redundancy mode, where a single program is replicated and they are redundantly executed in different cores. Figure 4(b) depicts the instruction-level redundancy mode, where every instruction in the program is replicated and they are executed in different clusters inside a single core. The thread-level redundancy mode achieves high performance but consumes large power. In contrast, the instruction-level redundancy mode consumes less power but achieves lower power than the thread-level redundancy mode does. Hence, a metric for evaluating dependability-performance trade-off is required and thus it is the main topic of this paper.

4. Formulating Metric for Dependability-Performance Trade-off

This section formulates a metric for dependability-performance trade-off. Mean instructions to failure (MITF) [20] is selected for the metric. It indicates how many instructions a processor executes between two errors. MITF is calculated as follows [20]:

$$MITF = \frac{\#instructions}{\#errors} = \frac{\#instructions}{\left(\frac{\#cycles}{f \times MTTF}\right)}$$

$$= IPC \times f \times MTTF = \frac{IPC \times f}{SER \times AVF}$$

where f is clock frequency, $MTTF$ is mean time to failure, IPC is instructions per cycle, SER is the SEU rate, and AVF is architectural vulnerability factor [20], which is the probability that a fault in a processor structure will result in a visible error in a program's final output. Using MITF, dependability-performance trade-off of the undependable mode, the thread-level redundancy mode, and the instruction-level redundancy mode are formulated as follows. MITF of the first one is

$$MITF_{undependable} = \frac{IPC_L \times f}{N_L \times E_b \times AVF_{core} + N_{Score} \times E_b \times AVF_{Score}}$$

where IPC_L is IPC of the large core, N_L is the total bit of the core, N_{Score} is the total bit of the components shared by the cores in the single chip, E_b is the SEU rate per bit, AVF_{core} is AVF of the single processor core without any redundancy, and AVF_{Score} is AVF of the shared portion in the chip. It is assumed that the undependable mode uses a single core.

MITF of the second model is

$$MITF_{thread} = \frac{IPC_L \times f}{N_{Score} \times E_b \times AVF_{Score}}$$

Here, any SEUs in the cores are detected and hence only SEUs occurred in the shared portion.

And last, MITF of the third one is

$$MITF_{instruction} = \frac{IPC_S \times f}{N_{Scluster} \times E_b \times AVF_{Scluster} + N_{Score} \times E_b \times AVF_{Score}}$$

where IPC_S is IPC of the small core, $N_{Scluster}$ is the total bit of the component shared by the clusters in the single core. $AVF_{Scluster}$ is AVF of the shared portion in the core.

An MITF is used to determine the dependability mode for each application. In other words, in the case of the MITF-directed mode selection, the dependability mode does not change during the programming execution. This is different from the ILP-directed mode selection, which switches the mode during the program execution. The reason why we do not use this adaptable technique in MITF-directed mode selection is that it is impossible to get AVF values dynamically in the program execution. Utilizing AVF prediction technique [21, 22, 23] for adaptation is a future study.

5. Methodology

Sim-SODA [24] and SimPoint [25] are used for architectural-level simulation and SPEC2000 is used for the evaluation. For each program, 10M instructions of a single simulation point that accurately representatives the complete execution of the program are found by SimPoint. A dual-cluster dual-core MCCP is selected for the platform. Based on the study in [26], the configurations of one processor core are determined as shown in Table 1. The front-end and L1 caches are shared by two clusters in a core. L2 cache is shared by two cores.

We simulate two single core processors and use the results to estimate simulation values of the dual core MCCP. One is LARGE processor where both two clusters are turned on. The other is SMALL processor where only one cluster is turned on. Using LARGE processor, we estimate the unpredictable and the thread-level redundancy modes. Using SMALL processor, we estimate the instruction-level redundancy mode.

E_b of 0.01 FIT/bit is used since SER is approximately from 0.001 to 0.01 FIT/bit [27]. It is assumed that the memory structures (caches and TLB) are protected by error correcting code (ECC). When ECC is used, E_b is 10^4 times reduced [28]. We assume N's are proportional to the area of the structure. The area estimates based on [26] are used, as shown in Table 2. The front-end and L1 caches are shared by two clusters in a core. L2 cache is shared by two cores. Using the values in the table, SEU rate for each mode is estimated.

Table 1. Processor Core Configurations

Fetch width	4 instructions
L1 instruction cache	32K, 2 way, 1 cycle
Branch predictor	4K global + 4K local
Dispatch width	4 instructions
Reorder buffer	128 entries
Instruction queue	16 entries / cluster
Issue width	2 instructions / cluster
Commit width	4 instructions / cluster
Integer ALUs	2 units / cluster
Integer multipliers	2 units / cluster
L1 data cache	32K, 2 way, 1 cycle
Unified L2 cache	512K, 8 way, 7 cycles

Table 2. Area Estimates (mm^2)

32K L1 data cache	13.0
32K L1 instruction cache	10.4
ITLB	2.2
DTLB	2.2
Fetch unit	1.3
Branch predictor	3.2
Decoder	1.7
Reorder buffer	16.6
Instruction queue	1.8 / 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

6. Results

IPC's of LARGE processor (IPC_L) and that of SMALL processor (IPC_S) are summarized in Figure 7.

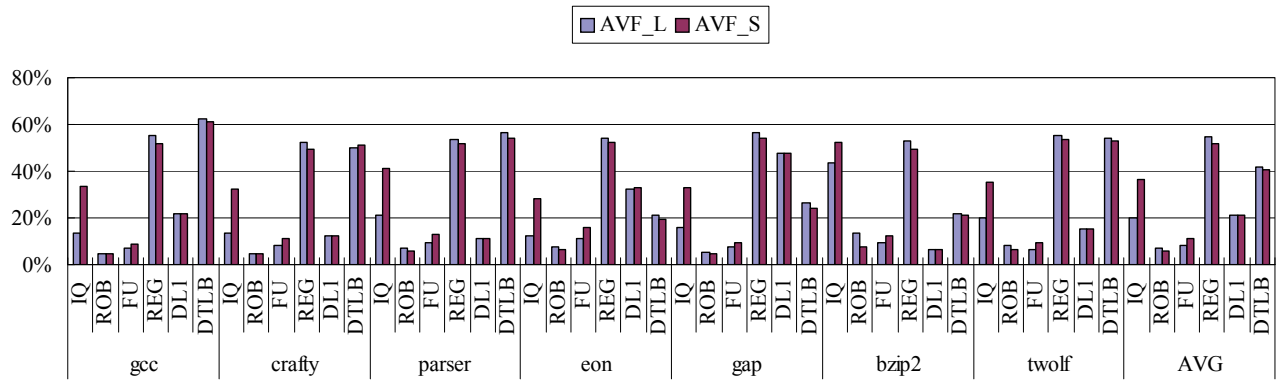

Figure 8. AVF

As can be easily observed, IPC_S's are significantly smaller than IPC_L's. In other words, the instruction-level redundancy diminishes performance very seriously. On average, IPC is reduced by 19.5%. Even if instruction-level redundancy provides with the equivalent dependability that thread-level redundancy does, MITF values are significantly smaller in instruction level than in thread level. In practice, instruction-level redundancy has less quality in dependability than thread-level one does. Hence, we are afraid that there is a large gap between MITF's for both modes.

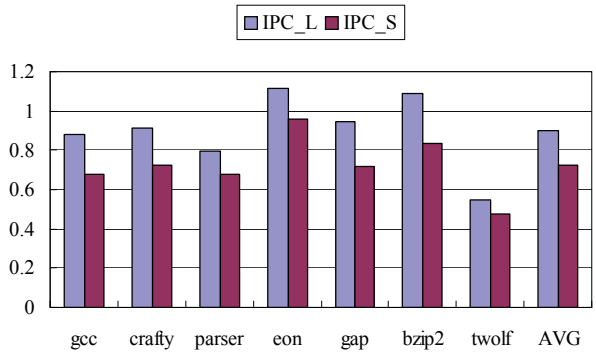

Figure 7. Instructions per Cycle

AVF's of LARGE processor (AVF_L) and that of SMALL processor (AVF_S) are summarized in Figure 8. Sim-SODA reports AVF values for instruction queue (IQ), reorder buffer (ROB), functional units (FUs), register files (REG), L1 data cache (DL1), and data TLB (DTLB). It is observed there are not any significant difference between AVF_L's and AVF_S's, except for the IQ. Hence, it is not expected that the AVF values contributes to MITF very much. For the other portions, we assume 10% for AVF. Memory structures are protected by ECC and almost all the vulnerable portions are combinational logic. Hence, we use the average AVF value of FU for the portions.

Figure 9 presents MITF for three redundancy modes. All values are normalized by the MITF for the undependable mode. It is surprising that the thread-redundancy mode has six times larger MITF than the undependable mode. This is because the thread-level redundancy mode protects almost all the portions in the processor. In contrast, MITF of the instruction-level redundancy mode is larger than that of the undependable mode by only 46%. From the trade-off between dependability and performance, the thread-level redundancy mode is attractive.

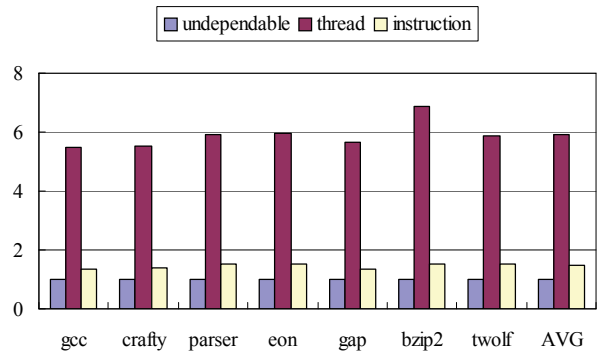

Figure 9. Relative MITF

Figure 10 presents MITF per energy consumption. The thread-level redundancy mode consumes larger power than the instruction-level redundancy mode does. Hence, this comparison is very interesting. Surprisingly, when we consider energy, MITF values are almost same between the undependable and the instruction-level redundancy modes. This is because the instruction-redundancy mode diminishes performance, as shown in Figure 7. Hence, the thread-level redundancy mode is still very attractive. The

instruction-level redundancy mode should be selected only when power budget is seriously small.

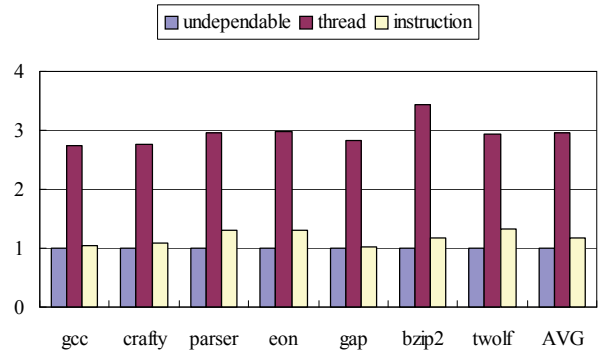

Figure 10. Relative MITF per Energy

7. Conclusions

This paper quantified dependability-performance trade-off on the MCCP. MITF was used as a metric for evaluating the trade-off. To the best of our knowledge, this was the first study that formulated MITF for multicore processors. When ECC is used for memory protection, thread-level redundancy mode has much more dependable than instruction-level redundancy mode at the cost of higher power consumption. The results will be used for the decision which redundancy model is selected for the present thread according to its importance.

For a future direction, studying a metric for dependability, power, and performance trade-off on the MCCP is also interesting.

Acknowledgements

This research has been supported by the Kayamori Foundation of Informational Science Advancement. It is also supported by Grant-in-Aid for Scientific Research (KAKENHI) (A)#19200004 and (B)#20300019 from Japan Society for the Promotion of Science (JSPS), and by the Core Research for Evolutional Science and Technology (CREST) program of Japan Science and Technology Agency (JST).

References

[1] 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)

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

[3] E. Rotenberg, "AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors," 29th International Symposium on Fault-Tolerant Computing (1999)

[4] 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)

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

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

[7] G. S. Sohi, M. Franklin, and K. K. Saluja, "A Study of Time-Redundant Fault Tolerance Techniques for High-Performance Pipelined Computers," 19th International Symposium on Fault-Tolerant Computing (1989)

[8] T. Sato and I. Arita, "In Search of Efficient Reliable Processor Design," 30th International Conference on Parallel Processing (2001)

[9] J. Ray, J. C. Hoe, and B. Falsafi, "Dual use of Superscalar Datapath for Transient-Fault Detection and Recovery," 34th International Symposium on Microarchitecture (2001)

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

[11] 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)

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

[13] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The Impact of Performance Asymmetry in Emerging Multicore Architectures," 32nd International Symposium on Computer Architecture (2005)

[14] E. Ipek, M. Kirman, N. Kirman, and J.F. Martinez, "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors," 34th International Symposium on Computer Architecture (2007)

[15] M. Pericas, R. Gonzalez, A. Cristal, F. Cazorla, D. A. Jimenez, and M. Valero, "A Flexible Heterogeneous Multi-Core Architecture," 16th International Conference on Parallel Architectures and Compilation Techniques (2007)

[16] C. Kim, S. Sethumadhavan, M.S. Govindan, N. Ranganathan, D. Gulati, D. Burger, and S. W. Keckler, "Composable Lightweight Processors," 40th International Symposium on Microarchitecture (2007)

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

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

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

[20] 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)

[21] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic Prediction of Architectural Vulnerability from Microarchitectural State," 34th International Symposium on Computer Architecture (2007)

[22] N. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for Bounding Vulnerabilities of Processor Structures," 34th International Symposium on Computer Architecture (2007)

[23] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online Estimation of Architectural Vulnerability Factor for Soft Errors," 35th International Symposium on Computer Architecture (2008)

[24] X. Fu, T. Li, and J. Fortes, "Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis," Workshop on Modeling, Benchmarking and Simulation (2006)

[25] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and More Flexible Program Analysis," Workshop on Modeling, Benchmarking and Simulation (2005)

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

[27] E. Normand, "Single Event Upset at Ground Level," IEEE Transactions on Nuclear Science, Vol. 43, No. 6 (1996)

[28] M. Sugihara, T. Ishihara, K. Hashimoto, and M. Muroyama, "A Simulation-Based Soft Error Estimation Methodology for Computer Systems," 7th International Symposium on Quality Electronic Design (2006)