Uncriticality-directed Scheduling for Tackling Variation and Power Challenges

Sato, Toshinori
Fukuoka University

Watanabe, Shingo
Kyushu Institute of Technology

出版情報：International Symposium on Quality Electronic Design. 10, pp.820-825, 2009-03-18
バージョン：
権利関係：

Uncriticality-directed Scheduling for Tackling Variation and Power Challenges

Toshinori Sato[1], Shingo Watanabe[2]
[1] Fukuoka University, 8-19-1 Nanakuma, Jonan-ku, Fukuoka, 814-0180 Japan
[2] Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, 820-8502 Japan
[1]E-mail: toshinori.sato@computer.org

Abstract

The advance in semiconductor technologies presents the serious problem of parameter variations. They affect threshold voltage of transistors and thus circuit delay has variability. Increasing the supply voltage to reduce the delay should not be a solution, since it increases power consumption, which is another serious problem in microprocessor designs. This paper proposes to combine recently-proposed configurable latency technique with an instruction scheduling technique considering instruction uncriticality. While relying only on the configurable latency technique degrades processor performance, the combination maintains it by executing only uncritical instructions in the long-latency units. The uncriticality-directed technique is extended for power reduction. This can be achieved by decreasing supply voltage for some variation-unaffected units. Detailed simulations show that the proposed scheduling technique improves processor performance by 7.0% on average over the conventional scheduling and that performance degradation from a variation-free processor is only 2.3% on average, when 2 of 4 integer ALUs are affected by variations. It also improves energy efficiency by 9.9% on average.

Keywords

Variability resilience, low power, microarchitecture

1. Introduction

While microprocessor performance has increased by the advance in semiconductor technology for more than three decades, it has to be further improved to satisfy emerging requirements. An example is security problem. Antivirus program wastes a large portion of processing power, which could be utilized by application programs if our computers were safe from the threat of computer viruses. It is predicted that even smart cell phones will require antivirus as well as PCs in the near future. Another example is mobile multimedia applications. Modern cell phones can play video games, digital still and video cameras, MP3 players, TV phones, and digital TVs. Hence, microprocessors, especially embedded processors, require more performance. It looks easy to improve processor performance, since the advanced semiconductor technologies will provide billions of transistors on a single chip, which enable a super-computer on the chip. However, this scaling is difficult due to new challenges of power consumption and parameter variations.

The advanced semiconductor technologies increase parameter variations [3, 8, 17]. Variations on an LSI chip are classified into die-to-die (D2D) and within-die (WID) variations. Recently, the latter ones, especially random WID variations, have become serious. Random dopant placement and line edge roughness (**P**rocess variations), supply voltage integrity (**V**oltage variations), and temperature fluctuations (**T**emperature variations) cause parameter variations. This paper focuses on process variations. They are essential in semiconductor technologies and they affect each transistor's threshold voltage, resulting in delay variability. Traditionally, microprocessor designs are based on the worst-case design methodology, which relies on guardband. Processor's maximum clock frequency is determined by considering the worst-case critical-path delay and the safety margin. The margin is required since delays are not constant due to the variations. In the worst-case design, the margins are summed up and thus PVT variations have a serious impact on supply voltage to satisfy required operating frequency and to improve performance yield of microprocessors. In other words, managing delay variability is a key to power reduction.

This paper first focuses on the parameter variations problem. Parameter variations affect threshold voltage of transistors and hence circuit delay has variability. This reduces performance yield of microprocessors and thus is serious for profitability of semiconductor companies. Fortunately, all circuits on a single chip are not affected by variations. Hence, by replacing the variation-affected circuits with variation-tolerant ones, the variation-affected chips can be shipped. This paper considers to replace variation-affected execution units on the chips with long-latency ones. Unfortunately, this causes severe performance loss. To attack the problem, this paper proposes an instruction scheduling technique that is aware of variation-originated long latency, which we call uncriticality-directed instruction scheduling. Second, this paper tries to reduce power consumed by variation-unaffected units. It utilizes the uncriticality-directed scheduling technique. The contributions of this paper are as follows:

1. Variability-aware instruction scheduling, which maintains processor performance under delay variability, is proposed.
2. Low-power instruction scheduling, which is an extension of the variability-aware scheduling, is proposed.

[1] The author is also with Kyushu University and CREST, JST.

[2] The author is currently with Fujitsu Limited.

This paper is organized as follows. The next section summarizes related works. Section 3 proposes an alternative scheduling directed by instruction uncriticality and describes a mechanism to identify uncritical instructions. Section 4 explains our evaluation methodology. Section 5 presents experimental results. Finally, Section 6 concludes.

2. Related Works

Variability-aware designs for yield enhancement are a hot topic. These techniques take the approach of post-silicon compensation. The simplest technique is to slow down clock frequency of variation-affected chips. This causes serious performance loss and thus is the least desirable choice. Another straightforward technique is to increase their supply voltage. Higher supply voltage improves transistor speed and thus tolerates large delay due to delay variability. Unfortunately, however, this cannot be chosen because power consumption problem is already serious.

Liang et al. [10] propose a configurable latency technique for floating-point unit (FPU). For every variation-affected chips, where timing specifications on FPU are not satisfied after fabrication, an extra pipeline stage is inserted into the FPU datapath and the latency of the datapath is extended. The latency does not change after the chip is shipped. Tiwari et al. [18] extend the configurable latency technique into the instruction pipeline. In the cases of unacceptable operating frequency, deeper pipeline depth is selected. Chen et al. [4] and Mohapatra et al. [12] respectively propose to adaptable latency techniques for arithmetic units. Each technique detects the situation, where a timing error occurs in an arithmetic units, on-the-fly and then adaptively switches to a longer latency. Different from the configurable latency technique, the latency changes according to operating situations. The adaptable latency technique might have a serious impact on the delay of arithmetic units because the decision to switch the latency must be done immediately before execution. Therefore, in the present paper, we propose to utilize the configurable latency technique. Unfortunately, as we will present later, it degrades processor performance severely.

Romanescu et al. [14] exploit instruction criticality to reduce the impact of the long-latency units due to delay variability. Using a critical path predictor (CPP) [6, 19], instructions that determine the number of cycles executing the program are identified. Such instructions are called critical instructions. Priority for short-latency units is given to the critical instructions. Unfortunately, while CPPs can be utilized for identifying critical path, none of them achieves both simplicity in circuit and accuracy in identification [5]. Complex circuit consumes additional power. Low accuracy seriously diminishes processor performance. In addition, CPPs has a serious problem to implement. They have a similar structure as caches and require a very large SRAM array. An SRAM cell is one of the most vulnerable circuit to process variations [1]. The conventional CPPs are not practical under the situation where PVT variations are severe. In the present paper, we will propose a practical technique to exploit instruction criticality.

3. Variability-aware Technique

3.1. Uncriticality-directed Scheduling

In order to tolerate timing errors in arithmetic units due to delay variability, this paper proposes a combination of a latency-aware instruction scheduling technique and the configurable latency technique. Without loss of generality, this paper focuses on the integer ALU with 1-cycle latency (iALU). For every variation-affected chips, where timing specifications on iALU are not satisfied, the latency of the iALU datapath is extended to 2 cycles. Different from the adaptable latency technique, the latency does not change after the chip is shipped. As we will see later, replacing each variation-affected iALU with the long-latency iALU significantly diminishes processor performance. In order to mitigate performance loss, this paper proposes to execute only uncritical instructions in the long-latency iALUs. While it is very similar to Romanescu et al.'s technique [14], which utilizes the conventional CPP, the two techniques were independently studied in parallel [16]. This section explains how uncritical instructions are identified.

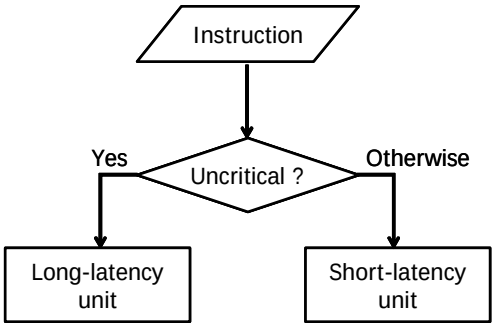

Figure 1: Uncriticality-directed Scheduling.

While we have studied several techniques to identify critical instructions for years, we have not yet found any technique that achieves both simplicity in circuit and high accuracy in identification [5]. To reduce the performance loss due to low identification accuracy, we propose to exploit instruction uncriticality rather than instruction criticality [16, 20]. Only uncritical instructions are executed in the long-latency units, as shown in Figure 1. Out-of-order execution processors have the instruction scheduling window, where instructions wait for their input operands. Each instruction can be issued to a functional unit where it is executed, only when its operands become available. Here, we call such an instruction ***ready*** instruction. In the instruction window, there are two types of ready instructions. One is instructions that have their dependent instructions in the instruction window. The other is instructions that do not have any dependent instructions. Here, we call the latter instructions ***solitary*** instructions. It is not necessary to

execute solitary instructions in hurry, since their execution results will not be immediately used. In other words, solitary instructions are uncritical. They can be executed in the long-latency units. It should be noted that even embedded processors currently execute instructions in an out-of-order fashion to attain high performance [13, 15] and then the proposed scheduling is adoptable.

Figure 2 shows the mechanism to identify solitary instructions. A small table is attached to the map table, which every modern microprocessor has for the purpose of dynamic register renaming. We call the table ***solitary*** table. The solitary table is 1-bit wide and its entry size is equal to the number of physical registers. Since conventional processors have only tens of registers, its hardware budget is very small. In a different view, every register file entry requires an additional 1-bit field. Hence, the solitary table does not require any large SRAMs, which are vulnerable to process variations. A large memory cell used in the register files is used to implement it. Under the situation where PVT variations are severe, the solitary table is more practical than the conventional CPPs.

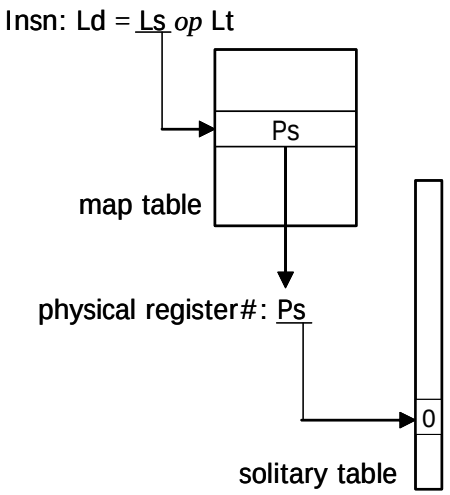

Figure 2: Solitary Table.

The solitary table works as follows. (1) When a new physical register is allocated as a destination, its associated entry in the solitary table is set. (2) When every instruction refers the map table by its logical source register number Ls and obtains the corresponding physical register number Ps, its associated entry in the solitary table is reset. (3) Whenever an instruction is issued, it refers the solitary table by its physical destination register number. If its associated entry is still set, it is a solitary instruction. This mechanism is 100% accurate in identifying solitary instructions, because all instructions in the instruction window have updated the solitary table when they are dispatched into the window. From these observations, we can see that the solitary table achieves both simplicity in circuit and accuracy in identification, when it is utilized for uncriticality-directed instruction scheduling.

3.2. Extension for Energy Reduction

Modern microprocessors usually have multiple iALUs. It would not be expected that all iALUs were affected by process variations and thus had long latency. A number of iALUs have short latency and consumes large power. This paper proposes to replace some of them with the long-latency iALUs. Because they are not affected by process variations, increasing their latency, in other words decreasing their delay per cycle, can reduce their supply voltage. Some of variation-unaffected iALUs becomes power-efficient with the cost of long latency. If the uncriticality-directed scheduling works well, considerable power reduction is expected. In summary, when an instruction is identified as uncritical and the long-latency unaffected iALU remains, it is executed on the power-efficient unaffected iALU as shown in Figure 3.

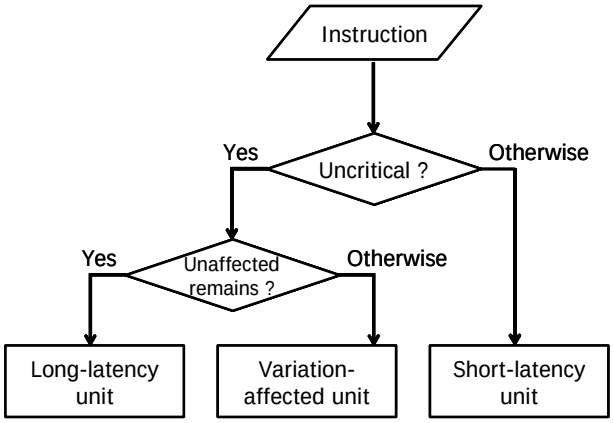

Figure 3: Extension for Power Reduction.

4. Evaluation Methodology

We implemented our simulator using SimpleScalar/ PISA tool set [2]. Table 1 summarizes processor configurations.

Six programs from SPEC2000 CINT and eight programs from MediaBench [9] are used. For SPEC programs, 200 million instructions are skipped before actual simulation begins. After that each program is executed for 100 million instructions. For MediaBench, each program is executed from beginning to end. We do not count NOP instructions.

We consider three cases, where 1, 2, and 3 iALUs are affected by variations, respectively, and thus they are replaced with the 2-latency iALUs. We compare the proposed scheduling with the conventional one.

For power evaluation, we assume the frequency-voltage specification (200MHz-1.0V and 100MHz-0.72V) of AMPLE processor [7]. The variation-affected ALUs and short-latency iALUs work at 1.0V and the long-latency

unaffected iALUs work at 0.72V. Only dynamic power consumed by iALUs are considered. While leakage power has become a serious problem, it is not considered in this evaluation. A reason is that the proposed technique never increases leakage current since it reduces supply voltage with maintaining threshold voltage of transistors.

Table 1: Processor Configuration.

Fetch width	8 instructions
L1 instruction cache	16KB, 2 way, 1 cycle
Branch predictor	gshare + bimodal
gshare predictor	4K entries, 12 histories
bimodal predictor	4K entries
Branch target buffer	1K set, 4way
Dispatch width	4 instructions
Instruction window size	32/64/128 instructions
Issue width	4 instructions
Integer ALUs	4 units
Integer multipliers	2 units
Floating-point ALU	1 unit
Floating-point multiplier	1 unit
L1 data cache ports	2 ports
L1 data cache	16KB, 4 way, 2 cycles
Unified L2 cache	8MB, 8 way, 10 cycles
Memory	Infinite, 100 cycles
Commit width	8 instructions

5. Results

5.1. Performance Improvement

Figure 4 explains how the uncriticality-directed instruction scheduling maintains processor performance under delay variability in the case where the instruction window size (IW) equals 32. The percentage increase in execution cycles is used as a metric. For each group of six bars, the left three bars (Conv:X) are for the conventional scheduling, and the right three bars (UCD:X) are for the uncriticality-directed scheduling. For each group of three bars, the left one (X=1) indicates the percentage reduction for the case where one iALU is affected by variations and is replaced by the 2-latency iALU. The middle (X=2) and the right (X=3) bars are for the cases where 2 and 3 variation-affected iALUs are replaced by the 2-latency iALUs, respectively.

As you can see, the conventional scheduling degrades processor performance seriously, especially for the case where the impact of variations is large. When 3 iALUs have to be replaced with the 2-latency iALUs, processor performance is decreased by as much as 52.1% with an average of 25.6%. In contrast, the uncriticality-directed scheduling efficiently maintains performance. It reduces performance only by 9.0% on average. Especially in the case of the replacement with one iALU, performance degradation is negligible and is 2.6% on average. This means the uncriticality-directed scheduling improves processor performance over the conventional one. It reduce the execution cycles by the averages of 12.0%, 14.7%, and 12.6% with the maximums of 25.5%, 28.1%, and 23.6% for one, two, and three 2-latency iALU situations, respectively.

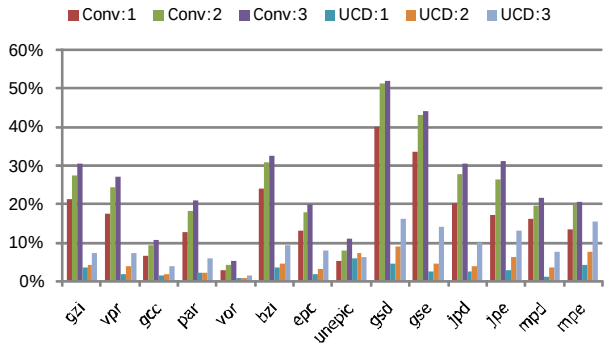

Figure 4: Increase in Execution Cycles (IW=32).

The increase in the instruction window size may obviate the uncriticality-directed scheduling. This is because larger instruction windows improve instruction scheduling capability and will tolerate long execution latency. In order to evaluate how the instruction window size affects the usefulness of the uncriticality-directed scheduling, we increase the size from 32 to 64 and 128. Figures 5 and 6 show the results. Actually, performance degradation in the conventional model decreases. When the window size is 64, it is 10.3%, 14.8%, and 16.7% for one, two, and three 2-latency iALU situations, respectively, on average.

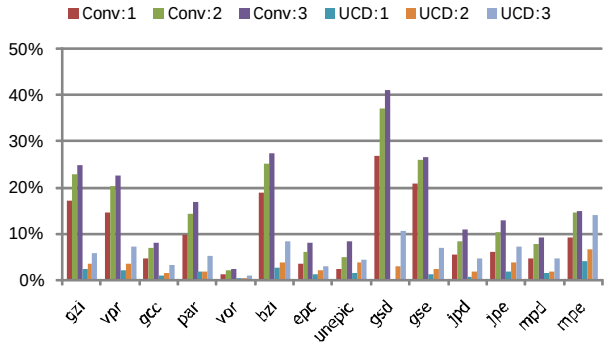

Figure 5: Increase in Execution Cycles (IW=64).

As the size increases, the degradation rate decreases. When the size is 128, it is 7.0%, 10.4%, and 12.5% for one, two, and three 2-latency iALU situations, respectively, on average. However, the increase in the window size does not obviate the uncriticality-directed scheduling. Even in the case where the size is 128, it reduce the execution cycles by the averages of 5.1%, 7.0%, and 6.6% with the maximums of 13.5%, 16.6%, and 14.2% for one, two, and three 2-latency iALU situations, respectively.

From these observations, we found that the uncriticality-directed instruction scheduling effectively exploit the configurable latency iALUs and thus has variability tolerance.

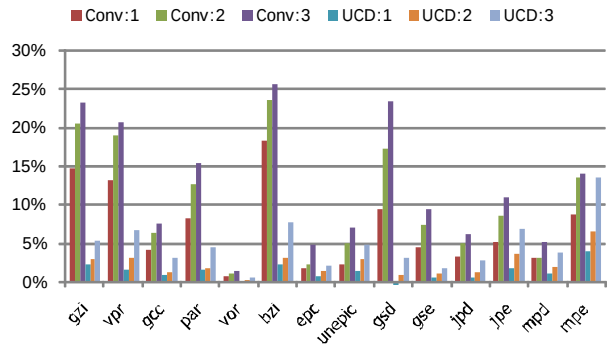

Figure 6: Increase in Execution Cycles (IW=128).

5.2. Energy Reduction

From the performance evaluation results, it is found that the uncriticality-directed scheduling maintains processor performance even when three iALUs are replaced by the 2-latency iALUs. Therefore, two situations are considered here. One is the case where one iALU is affected by process variations. In this case, two unaffected iALUs are replaced by the 2-latency iALUs. Totally, three iALUs have 2-cycle latency. The other is the case where two iALU are affected by process variations. In this case, one unaffected iALU is replaced by the 2-latency iALU. Similarly, three iALUs have 2-cycle latency in total. Remember that only 2-latency unaffected iALUs work at the lower supply voltage. Both the variation-affected iALUs and the short-latency iALUs work at the higher supply voltage.

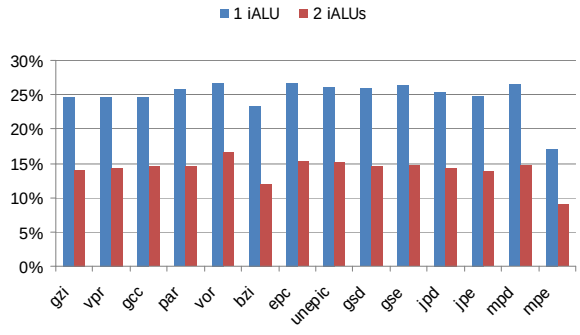

Figure 7: Energy Reduction (IW=128).

Figure 7 presents the results for the cases where the instruction window has 128 entries. It shows the percentage of energy reduction. Only energy consumed by iALUs is considered. For other window size, the tendency of the results is same and thus the worst effective case is only shown. Energy consumed by iALU is reduced by 24.9% and 14.1% when the number of the variation-affected iALUs is 1 and 2, respectively. The baseline is the case where only variation-affected iALUs are replaced by the 2-latency iALUs. These values are 25.5% and 15.8% for the 32-entry window case, and are 24.9% and 14.6% for the 64-entry window case.

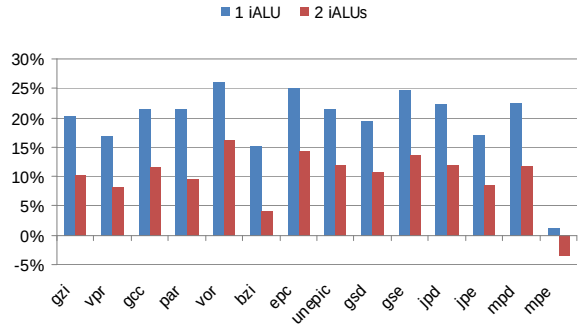

Figure 8: ED^{2}P Improvement (IW=128).

While the decrease in energy consumption is preferable, the increase in execution cycles is undesirable. Both energy and performance should be considered for evaluating energy efficiency. Figure 8 presents the percentage improvement in energy-delay-square product (ED^{2}P) when the instruction window size is 128. ED^{2}P is a good metric for evaluating energy efficiency when there are multiple supply voltages [11]. ED^{2}P is improved by 19.7% and 9.9% when the number of the variation-affected iALUs is 1 and 2, respectively. Only in the case of mpeg2encode (mpe), ED^{2}P is diminished by 3.5% when two iALUs are affected by variations. As can be seen in Figures 6 and 7, performance degradation is largest and energy reduction is smallest in mpeg2encode. The synergetic effect results in ED^{2}P degradation.

In general, the proposed technique effectively reduce energy consumption under the situation of severe PVT variations.

6. Conclusions

The advanced semiconductor technologies increase process variations, which seriously affect circuit delay. Recent proposals of the configurable latency technique is one solution for managing delay variability, however, the impact on processor performance is severe. This paper proposed the uncriticality-directed instruction scheduling. Only uncritical instructions are executed in the long-latency iALUs. When 2 of 4 iALUs are affected by variations, it improves processor performance by the average of 7.0% over the conventional scheduling, and performance degradation from a variation-free processor is only 2.3% on average. When the scheduling method is extended for power reduction, it improves ED^{2}P by 9.9% on average. The uncriticality-directed instruction scheduling effectively

exploit the configurable latency units and thus has variation tolerance.

7. Acknowledgments

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) programs of Japan Science and Technology Agency (JST).

8. References

[1] K. Agarwal and S. Nassif, "The impact of random device variation on SRAM cell stability in Sub-90-nm CMOS technologies", IEEE Transactions on Very Large Scale Integration Systems, Vol. 16, No. 1, pp.86-97, January 2008.

[2] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling", IEEE Computer, Vol. 35, No. 2, pp.59-67, February 2002.

[3] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture", 40th ACM/IEEE Design Automation Conference, pp.338-342, June 2003.

[4] Y. Chen, H. Li, J. Li, and C.-K. Koh, "Variable-latency adder (VL-adder): new arithmetic circuit design practice to overcome NBTI", ACM/IEEE International Symposium on Low Power Electronics and Design, pp.195-200, August 2007.

[5] A. Chiyonobu and T. Sato, "Evaluating the critical path predictors using critical path detection criteria", IPSJ SIG Technical Report, 2006-ARC-169, Vol. 2006, No. 88, pp.61-66, August 2006 (in Japanese).

[6] B. Fields, S. Rubin, and R. Bodik, "Focusing processor policies via critical-path prediction", 28th ACM/IEEE International Symposium on Computer Architecture, pp.74-85, July 2001.

[7] T. Ishihara, S. Yamaguchi, Y. Ishitobi, T. Matsumura, Y. Kunitake, Y. Oyama, Y. Kaneda, M. Muroyama, and T. Sato, "AMPLE: an adaptive multi-performance processor for low-energy embedded applications", 6th IEEE Symposium on Application Specific Processors, pp.83-88, June 2008.

[8] T. Karnik, S. Borkar, and V. De, "Sub-90nm technologies: challenges and opportunities for CAD", 20th ACM/IEEE International Conference on Computer Aided Design, pp.203-206, November 2002.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", 30th ACM/IEEE International Symposium on Microarchitecture, pp.330-335, December 1997.

[10] X. Liang and D. Brooks, "Mitigating the impact of process variations on processor register files and execution units", 39th ACM/IEEE International Symposium on Microarchitecture, pp.504-514, December 2006.

[11] M. Martonosi, D. Brooks, and P. Bose, "Modeling and analyzing CPU power and performance: metrics, methods, and abstractions", ACM/IFIP Joint International Conference on Measurement & Modeling of Computer Systems, Tutorial, June 2001.

[12] D. Mohapatra, G. Karakonstantis, and K. Roy, "Low-power process-variation tolerant arithmetic units using input-based elastic clocking", ACM/IEEE International Symposium on Low Power Electronics and Design, pp.74-79, August 2007.

[13] V. Rajagopalan, "New area and power-efficient MIPS processors achieve high performance", 4th Microprocessor Forum Japan, CD-ROM, June 2007.

[14] B. F. Romanescu, M. E. Bauer, S. Ozev, and D. J. Sorin, "Reducing the impact of intra-core process variability with criticality-based resource allocation and prefetching", ACM International Conference on Computing Frontiers, pp.129-138, May 2008.

[15] T. Sartorius, "The Scorpion mobile application microprocessor", 4th Microprocessor Forum Japan, CD-ROM, June 2007.

[16] T. Sato and S. Watanabe, "Instruction scheduling for variation-originated variable latencies", 9th IEEE International Symposium on Quality Electronic Design, pp.361-364, March 2008.

[17] X. Tang, V. K. De, and J. D. Meindl, "Intrinsic MOSFET parameter fluctuations due to random dopant placement", IEEE Transactions on VLSI Systems, Vol. 5, No. 4, pp.369-376, December 1997.

[18] A. Tiwari, S. R. Sarangi, and J. Torrellas, "ReCycle: pipeline adaptation to tolerate process variation", 34th ACM/IEEE International Symposium on Computer Architecture, pp.323-334, June 2007.

[19] E. Tune, D. Liang, D. M. Tullsen, and B. Calder, "Dynamic prediction of critical path instructions", 7th IEEE International Symposium on High Performance Computer Architecture, pp.185-196, January 2001.

[20] S. Watanabe and T. Sato, "Uncriticality-directed low-power instruction scheduling", IEEE Computer Society Annual Symposium on VLSI, pp.69-74, April 2008.