A Case for Exploiting Complex Arithmetic Circuits towards Performance Yield Enhancement

Shingo Watanabe[1], Masanori Hashimoto[2], Toshinori Sato[3]
[1] Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, 820-8502 Japan
[2] Osaka University, 1-5 Yamadaoka, Suita, Osaka, 565-0871 Japan
[3]Fukuoka University, 8-19-1 Nanakuma, Jonan-ku, Fukuoka, 814-0180 Japan
[3]E-mail: toshinori.sato@computer.org

Abstract

As semiconductor technologies are aggressively advanced, the problem of parameter variations is emerging. Process variations in transistors affect circuit delay, resulting in serious yield loss. Considering the situations, variation-aware designs for yield enhancement interest researchers. This paper investigates to exploit the statistical features in circuit delay and to cascade dependent instructions for reducing variations. From statistical static timing analysis in circuit level and performance evaluation in processor level, this paper tries to unveil how efficiently instruction cascading improves performance yield of processors. Cascading instructions increases logic depth and decreases the standard deviation of the circuit delay. That might improve performance yield of microprocessors. Unfortunately, however, it is found that variability reduction in the circuit level does not always mean yield enhancement in the microarchitecture level.

Keywords

Process variations, yield enhancement, microarchitecture

1. Introduction

Popularly known as Moore's law, advanced semiconductor technologies have increased the number of transistors on a single chip and contributed to improve processor performance. Unfortunately, however, aggressive integration technologies recently unveil a serious problem of parameter variations [3, 4]. Variations on a single chip are classified into die-to-die (D2D) and within-die (WID) variations. Recently, the latter ones, especially random WID variations, have become serious. Randomdopant fluctuations and line-edge roughness (Process variations), uneven supply voltage distribution (Voltage variations), and temperature fluctuations (Temperature variations) cause parameter variations. Process variations are essential in semiconductor technologies and they affect each transistor's threshold voltage, resulting in performance variability. This paper focuses on process variations. Process variations influence circuit delay. Even though chips have identical design and environment, some of them may violate timing specifications. It is easily expected that the number of bad chips increases, resulting in yield loss, in the near future. The goal of this study is to improve performance yield via variability reduction techniques in the microarchitecture level.

In order to reduce the variability in circuit delay and hence to enhance performance yield, the well-known statistical features on circuit delay [5, 8] should be exploited. Every circuit delay is strongly dependent upon the number of critical paths and the logic depth. As the number of critical paths increases, the mean delay increases and the standard deviation decreases. Similarly, as the logic depth increases, the mean delay increases and the standard deviation decreases. This paper investigates the instruction cascading (or ALU collapsing [21]) to exploit the statistical features in the microarchitecture level. The instruction cascading is a technique to collapse dependent instructions. Several instructions are cascaded and executed in a single cycle using cascaded units. Cascading multiple execution units increases the logic depth and thus the standard deviation may decrease. The negative impact on performance due to the decline in clock frequency may be compensated by the increase in instructions per cycle (IPC). This is because the multiple numbers of dependent instructions are executed in a single cycle. This paper statistically analyzes the delay of the cascaded adder, evaluates IPC gain obtained by the instruction cascading, and estimates performance yield.

This paper is organized as follows. The next section summarizes related works. Section 3 introduces the statistical features of circuit delay. Section 4 describes instruction cascading. Section 5 introduces our evaluation methodology and Section 6 presents the results. Finally, Section 7 concludes.

2. Related Works

Variation-aware designs for yield enhancement are a hot topic [9, 11-16, 18-20, 22]. These techniques take the approach of post-silicon compensation. Liang et al. [12] and Mohapatra et al. [13] avoid timing errors in register files and arithmetic units by adaptively switching to a longer latency. Ozdemir et al. [14] address the issue of scheduling complexity for variable-access L1 cache by using additional load-bypass buffers. Tiwari et al. [19] extend the variable latency technique into the pipeline. In the cases of unacceptable operating frequency, deeper pipeline depth is selected. K et al. [9] utilize the observation that every entry in the instruction queue is not fully affected by variations and

[1] The author is currently with Fujitsu Limited.

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

store instructions with one or two ready input operands in partially affected entries. This avoids multi-cycle accesses. Romanescu et al. [15] and Sato et al. [16] exploit instruction criticality to reduce the impact of long latency units. Priority for fast units is given to critical instructions. Romanescu et al. [15] also utilize prefetching to mitigate the impact of slow cache accesses. SAVS [11] and X-Pipe [22], which are extensions of Razor [6], are techniques to detect every timing violation caused by variability and to revert processor state to a safe point where it is detected. Adaptive body bias (ABB) [20] controls the substrate bias to improve transistor speed by lowering the threshold voltage. Teodorescu et al. [18] show that ABB is effective when it is applied at the block level.

As mentioned above, these techniques only compensate the impact of variations. To the best of our knowledge, it has not been considered yet to mitigate variability itself by some techniques in the microarchitecture level. In contrast to the previous works, this paper tries to reduce timing variability in the microarchitecture level by exploiting the statistical features of circuit delay

3. Statistical Features of Path Delay

This section explains the statistical features of circuit delay. Figure 1 illustrates the dependence of the WID maximum critical path delay density function on the number of critical paths [5, 8]. The horizontal axis indicates the delay and the vertical axis indicates the density. It is assumed that the paths are completely independent and are modeled as normal distributions $N(6, 1^2)$, where the mean delay (μ) is 6 and the standard deviation (σ) is 1. 'n' in Figure 1 presents the number of completely independent critical paths. Hence, the bold line denoted by 'n=1' shows the normal distributions $N(6, 1^2)$. As the number of critical paths increases, the mean delay increases. This is because the slowest critical path limits the circuit's overall performance. In contrast, the standard deviation decreases.

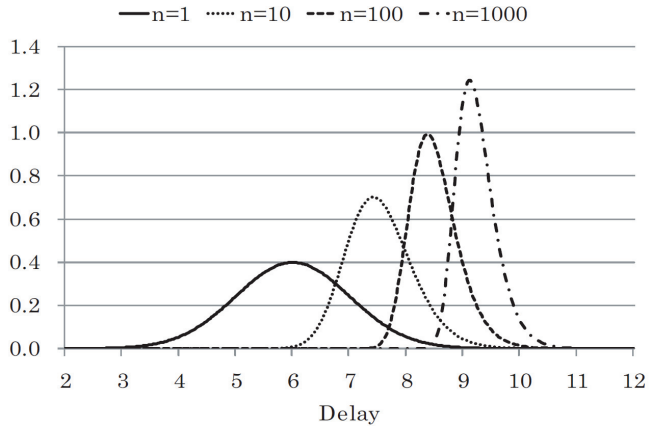

Figure 1: Dependence of circuit delay on number of critical paths.

Figure 2 illustrates the dependence of the delay density function on the logic depth of the critical paths, when the number of critical paths is 100. As same to Figure 1, the horizontal axis indicates the delay and the vertical axis indicates the density. 'm' in the figure presents the relative logic depth of one critical path. As the logic depth of a path increases, its standard deviation decreases. This is because the total delay of a path is averaged as the number of gates in the path increases. The impact of each gate's variability is reduced. As the logic depth (m) increases, the standard deviation relatively decreases by a factor of $1/\sqrt{m}$. The dotted line denoted by 'm' is identical with the broken line denoted by 'n=100' in Figure 1. As the logic depth of all paths increases, the mean delay increases and the standard deviation decreases.

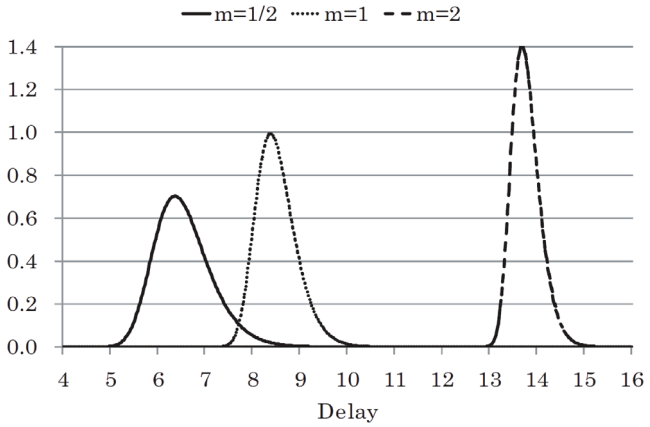

Figure 2: Dependence of circuit delay on logic depth.

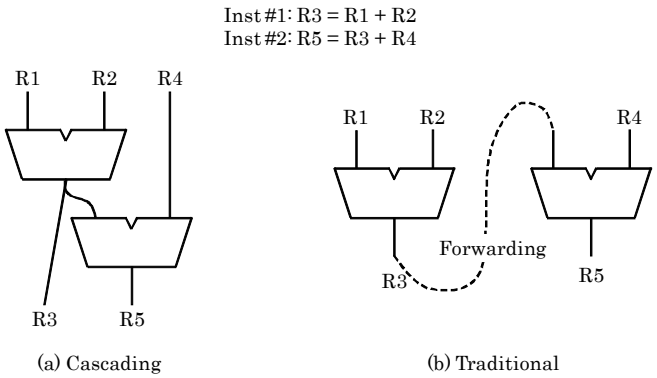

Figure 3: Instruction cascading.

4. Variability Reduction Microarchitecture

This section investigates a microarchitecture technique to reduce delay variability. It focuses on the execution stage in the pipeline. In other words, this paper assumes that the execution stage determines the clock frequency of the microprocessor. This is similar to an assumption used in [11]. An important thing to consider is that influences to the other stages due to microarchitecture modifications in the execution stage should be as small as possible.

In order to mitigate the variability in circuit delay, the use of instruction cascading is proposed by exploiting the statistical feature of circuit delay. As shown in Figure 3(a), the result of the producer instruction is directly fed into the consumer one. The cascaded unit executes multiple dependent instructions in a single cycle [21]. Conventional

processors cannot execute multiple dependent instructions in a single cycle, as shown in Figure 3(b). In contrast, the cascaded unit executes the two instructions in a single cycle and thus it improves IPC. Cascading multiple execution units increases the logic depth and thus mitigates delay variability.

On the other hand, cascading multiple units increases circuit delay. This has an advantage and a disadvantage. The advantage is that timing margins in the other pipeline stages rather than the execution stage increase. Since the clock frequency decreases, the timing constraints in the other stages are mitigated. The disadvantage is that net processor performance might be degraded. Later, this paper evaluates processor performance considering both the increase in IPC and the decline in clock frequency.

In this paper, to cascade dependent instructions is called grouping. It should be noted that grouping does not mean instruction fusion [7]. Those instructions are still two instructions and the total number of instructions does not change. Grouping is operated as follows.

1. Grouping is performed on two instructions. A pair of instructions are denoted as a group. Grouping three and more instructions is prohibited. Every instruction is included in at most one group. Every consumer instruction that has only one unready operand is a candidate for grouping.
2. Grouping is performed at the instruction decode stage. If two consecutive instructions form a pair of producer and consumer instructions, they are grouped. Later in this paper, we adopt the instruction cascading to a 2-way inorder processor. Hence, determining whether grouping is possible or not is easily implemented at the decode stage as a part of the mechanism detecting a dependence between the two instructions.
3. Every load instruction is not considered as a producer instruction for grouping. This does not mean memory operations are outside of candidates for cascading. Every load or store instruction is divided into an address calculation operation and a memory access operation, and hence an address calculation operation can be a consumer in a group.

5. Evaluation Methodology

Our evaluation consists of statistical static timing analysis and IPC performance evaluation. Processor performance is obtained from clock frequency and IPC. Combining SSTA results with IPC evaluation results gives us net processor performance.

5.1. Statistical Static Timing Analysis

We built an SSTA flow shown in Figure 4. It consists of Synopsys Design Compiler and two in-house tools.

1. From an RT level HDL design, a netlist and its associated SDF (standard delay format) file are generated. Synopsys Design Compiler is used.
2. A lot of SDF files, where there are variations in gate delay, are generated from the original SDF file. In an SDF file, timing specifications associate rising and falling delay values with input-to-output paths. The delay values are modified randomly assuming variations. This process is repeated just like Monte-Carlo simulation and a lot of sampled SDF files are obtained. The delay values are modeled as normal distributions. The deviance is provided according to an assumed ratio of the standard deviation to the mean delay (σ). For example, the typical gate σ/μ value of 0.064 for 65nm technology [3] is given. An in-house tool is used.
3. For each SDF file, an STA is performed on the netlist. Design Compiler is used.
4. All STA results are statistically processed and an SSTA result is obtained. An in-house tool is used.

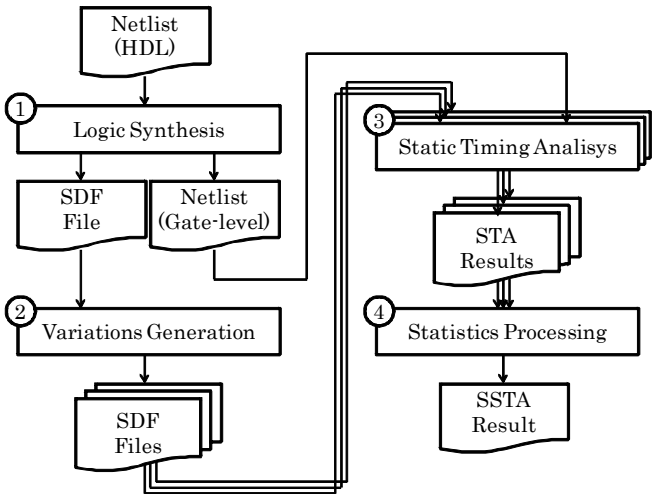

Figure 4: SSTA flow.

As you guess, you can easily build the additional tools. Only Design Compiler is required for generating a netlist and its corresponding SDF file and for performing a number of STAs. An advantage of this flow is that it considers the entire circuit rather than a part of the circuit. It analyzes not only most critical paths but also every possible path, and thus unexpected effects of variations on circuit delay can be identified. Another advantage is its short processing time due to the static nature of the analysis. A limitation is that it does not consider correlations in delay values. Since the variability of the rising and falling delay is randomly generated, the effect of circuit topology on the delay values is not considered. This might reduce accuracy. It is expected that increasing the number of SDF samples improves the accuracy.

As mentioned above, this paper bases on the assumption that the execution stage determines the clock frequency. This methodology follows that used in [11]. One of the performance critical units in the execution stage is ALUs. This paper focuses on ALUs, especially on addition. Hence, our evaluation methodology uses adders as a representative that determines the clock frequency. It should be noted there are other performance critical units in a microprocessor. Focusing on adders is a first step towards variability reduction microarchitectures that considers whole processor.

Five kinds of 32b adders are analyzed by the SSTA tool. They are carry select adder (CSA), carry look-ahead adder (CLA), ripple-block CLA, block CLA, and Lander-Fischer adder. A CSA is designed by hand and the other adders are generated by the Arithmetic Module Generator [23]. They are described by Verilog-HDL language. For every adder, two types of adders are analyzed. One is the conventional 2-input adder. The other is a 3-input 2-output adder, which consists of two conventional 2-input adders. They are cascaded and form the 3-input 2-output adder. Synopsys Design Compiler generates their netlists. Hitachi 0.18μm cell library is used. The 3-input 2-output adders are synthesized after its hierarchy is flattened.

In the following analysis, σ/μ is used as a metric to evaluate how delay variability is mitigated. The typical σ/μ value in gate delay is 0.064 for 65nm technology [3] and it is used to estimate the impact on circuit delay of variations in the current technologies. 10,000 SDF samples are generated for every SSTA.

5.2. IPC Performance Evaluation

SimpleScalar tool set [1] is used for evaluating IPC. Alpha instruction set is used. Six programs from SPEC2000 CINT and eight programs from MediaBench [10] are used as benchmark programs. For SPEC programs, first 1 billion instructions are skipped and the following 2 billion instructions are simulated in detail. For MediaBench, each program is executed from beginning to end. The configuration of the baseline processor is shown in Table 1. It is a 2-way inorder processor and is based on Intel Atom 200 series 1.6GHz/FSB 533MHz [17].

Hereafter, the processor utilizing the cascaded units is called CASCADE processor. In CASCADE processor, the total number of component units in Table 1 does not change.

Table 1: Baseline processor configuration.

Fetch width	2 instructions
L1 I-cache	32KB, 2way, 1 cycle
Branch predictor	gshare, 4K entries, 12 histories
Instruction buffer	16 entries
Issue width	2 instructions
Integer ALUs	2 units, 1 cycle
Integer multipliers	2 units
	MULT 3 cycles, DIV 20 cycles
Floating ALUs	2 units, 2 cycles
Floating multipliers	2 unit
	MULT 4 cycles, DIV 12 cycles,
	SQRT 24 cycles
L1 D-cache port	2 ports
L1 D-cache	32KB, 2way, 2 cycles
Unified L2 cache	512KB, 8way, 16 cycles
Memory	34 cycles
Commit width	2 instructions

6. Results

6.1. SSTA Results

Table 2 presents μ, σ, and σ/μ values of each adder's path delay. The first column shows the types of adders. The next three columns are μ, σ, and σ/μ values of 2-input adders and the next three are those of 3-input 2-output ones. The last column explains how the σ/μ value is reduced from the 2-input adders to the 3-input 2-output ones. In other words, it presents the mitigation in delay variability. Since each 3-input 2-output adder has larger path delay than its corresponding 2-input one does, both μ and σ are larger in the former than in the latter. An interesting observation is that σ/μ is reduced. That means the cascading succeeds in mitigating delay variability.

As can be seen in Table 2, five adders show the same tendency. Therefore, the following analysis focuses on the CSAs. In order to consider optimistic and pessimistic cases as well as the typical one, gate σ/μ values of 0.040, 0.080, and 0.100 are used as delay variability. In the rest of this paper, the 2-input CSA and the 3-input 2-output one are called *2CSA* and *3CSA*, respectively.

Table 2: Path delay variability of five adders.

	2-input			3-input 2-output			Imp.(%)
Adders	μ	σ	σ/μ	μ	σ	σ/μ	
CSA	4.82	0.088	0.0183	5.96	0.093	0.0155	15.1
CLA	4.84	0.087	0.0179	6.00	0.088	0.0146	18.4
Ripple-block CLA	4.92	0.088	0.0179	6.16	0.092	0.0149	16.7
Block CLA	4.90	0.089	0.0181	6.00	0.090	0.0149	17.4
Lander-Fischer	4.84	0.087	0.0181	6.04	0.093	0.0153	15.1

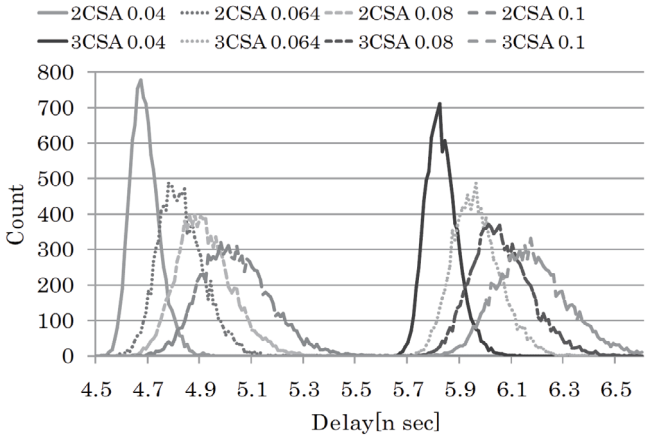

Figure 5: Dependence of path delay on gate delay variability.

Figure 5 presents frequency distributions of maximum critical path delay for both *2CSA* and *3CSA*. The horizontal axis indicates the delay and the vertical axis indicates the counts of the number of adders equal to the corresponding value. Graphs shown left are for *2CSA* and those shown right are for *3CSA*. Table 3 summarizes μ, σ, and σ/μ of path delay. The fifth column in the table explains how the σ/μ value of *3CSA* is smaller than that of *2CSA*. As the σ/μ value of gate delay increases, the mean delay, the standard deviation, and the σ/μ value of path delay increase.

Table 3: Influence of gate delay variability on μ, σ, and σ/μ of path delay.

	Gate σ/μ	μ	σ	σ/μ	Imp.(\%)
2CSA	0.040	4.68	0.056	0.0120	-
	0.064	4.82	0.088	0.0183	-
	0.080	4.91	0.108	0.0219	-
	0.100	5.03	0.134	0.0267	-
3CSA	0.040	5.82	0.062	0.0107	10.8
	0.064	5.96	0.093	0.0155	15.1
	0.080	6.05	0.117	0.0192	12.5
	0.100	6.18	0.143	0.0231	13.4

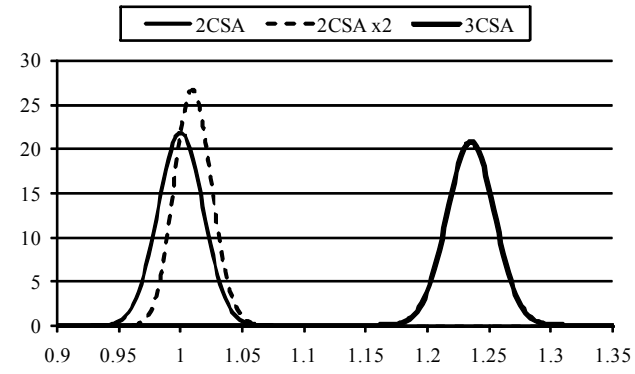

Figure 6: Influence of number of adders on circuit delay.

As the number of adders increases, the number of critical path also increases. Hence, the path delay is dependent upon the number of units. Hereafter, the path delay distributions analyzed above are modeled as normal distributions. Based on the statistical MAX operation [2], the dependence of path delay on the number of adders is estimated. Figure 6 presents frequency distributions of maximum path delay, when the number of units is considered. '2CSA', '2CSAx2', and '3CSA' in the figure denote the distribution in the cases of one *2CSA*, two *2CSA*s, and one *3CSA*. All values are normalized by the mean delay of the *2CSA*. As we already know in Figure 1, the increase in the number of adders means larger mean delay.

6.2. IPC Results

Figure 7 shows IPC improvement for CASCADE processor over the baseline processor. The horizontal axis indicates benchmark programs and the vertical axis indicates the IPC improvement. The instruction cascading increases IPC by 11.2% on average. Figure 6 explains the clock frequency of CASCADE processor is more than 20% slower than that of the baseline. The IPC improvement of 11.2% cannot compensate the slower clock frequency. Hence, it is afraid that CASCADE processor is inferior in performance in comparison with the baseline.

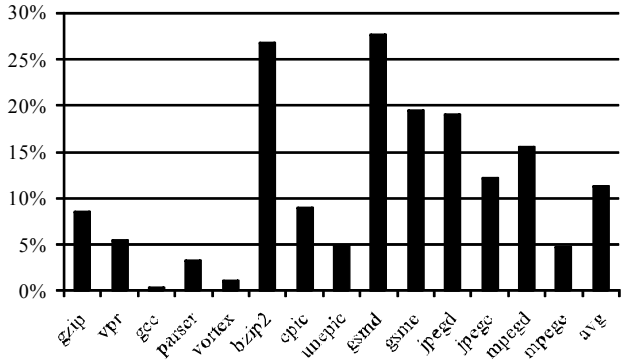

Figure 7: IPC Improvement.

6.3. Performance Yield Results

Combining the SSTA results with the IPC results gives us estimations on performance yield of processors. Processor performance is obtained from IPC and clock frequency. Frequency distributions are obtained from the SSTA results. Therefore, performance distributions are obtained from the SSTA and IPC results. Figure 8 shows performance distributions. The horizontal axis indicates the normalized performance and the vertical axis indicates the performance density. Performance is normalized by that of the baseline processor consisting of the *2CSA*s with the mean delay value.

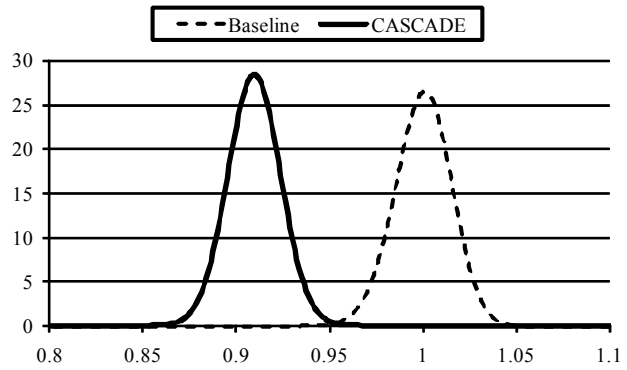

Figure 8: Performance distributions.

The effectiveness of cascading on performance yield is investigated. The baseline processor provides better performance than CASCADE processor does. Even though IPC is larger in CASCADE processor than in the baseline one, net performance is better in the baseline processor than in CASCADE one. This is because the gain in IPC does not compensate the decline in clock frequency. A surprising result is that σ/μ is almost the same between CASCADE and the baseline processors. In other words, timing variability in the processor level is not improved, even though that in the component level (i.e. adders) is improved. This is due to the dependence of delay on the number of critical paths. The maximum delay in the baseline processor is determined by two *2CSA*s. On the other hand, that in CASCADE processor is determined one *3CSA*. It has been seen in Section 3 that σ/μ decreases as the number of critical paths increases. On the other hand, it has been also seen that σ/μ decreases as the logic depth increases. Hence, it is found that the effect of the increase in logic depth and that of the increase in the numbers of critical paths are comparable.

From the observations above, the following findings are obtained. Even though increasing logic depth of an adder improves its timing variability, performance yield of processors is not improved. There are several reasons. First, the IPC gain does not compensate the decline in clock frequency. Second, the dependence of path delay on the logic depth and that on the number of critical path are comparable. Therefore, it is found that focusing on a part of a processor is not enough to improve its performance yield. Instead, global and aggressive refinements on microarchitecture will be required. In order to keep clock frequency, functional units with small delay should be used. A special 3-input 1-output adder consisting of carry save adders is a good candidate. However, two issues should be considered. One is that a microarchitecture change is required. Intermediate results are lost and thus a mechanism to provide precise interruptus should be innovated. Its complexity must not increase. The other is that timing margins in the other stages are reduced. The delay variability in the other stages should be considered.

7. Conclusions

This paper considered to utilize the cascaded units to improve performance yield of processors. The SSTA results explained that timing variability of an adder is mitigated. However, unfortunately, microarchitecture level performance evaluation showed the IPC gain is not enough to compensate the decline in clock frequency caused by the increase in the logic depth. Consequently, processor's performance yield was not improved when the yield enhancement technique was applied only on a part of the processor. The contribution of this paper is to present the SSTA methodology for microarchitecture evaluations. The other contribution is to unveil that variability reduction in the circuit level does not always mean yield enhancement in the microarchitecture level.

8. Acknowledgments

Hitachi 0.18μm standard cell libraries are provided by VDEC (VLSI Design and Education Center) in the University of Tokyo. 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).

9. References

[1] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling", IEEE Computer, 35(2), 2002.

[2] M. Berkelaar, "Statistical delay calculation, a linear time method", 6th International Workshop on Logic Synthesis, 1997.

[3] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond", IBM Journal of Research and Development, 50(4/5), 2006.

[4] 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, 2003.

[5] K. A. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration", IEEE Journal of Solid-State Circuits, 37(2), 2002.

[6] D. Ernst, N. S. Kim, S. Das, S. Pant, R. R. Rao, T. Pham, C. H. Ziesler, D. Blaauw, T. M. Austin, K. Flautner, and T. N. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation", 36th ACM/IEEE International Symposium on Microarchitecture, 2003.

[7] P. Gepner and M. F. Kowalik, "Multi-core processors: new way to achieve high system performance", 5th

International Symposium on Parallel Computing in Electrical Engineering, 2006.

[8] M. Hashimoto and H. Onodera, "Increase in delay uncertainty by performance optimization", IEEE International Symposium on Circuits and Systems, 2001.

[9] R. K and M. Mutyam, "Process variation aware issue queue design", Design, Automation and Test in Europe, 2008.

[10] 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, December 1997.

[11] H. Li, Y. Chen, K. Roy, and C.-K. Koh, "SAVS: a self-adaptive variable supply-voltage technique for process-tolerant and power-efficient multi-issue superscalar processor design", 11th Asia and South Pacific Design Automation Conference, 2006.

[12] 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, 2006.

[13] 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, 2007.

[14] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou, "Yield-aware cache architectures", 39th ACM/IEEE International Symposium on Microarchitecture, 2006.

[15] 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, 2008.

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

[17] T. Thakkar, "Mobile internet devices enabling the best internet experience in your pocket", 11th IEEE Symposium on Low-Power and High-Speed Chips, 2008.

[18] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating parameter variation with dynamic fine-grain body biasing", 40th ACM/IEEE International Symposium on Microarchitecture, 2007.

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

[20] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage", IEEE Journal of Solid-State Circuits, 37(11), 2002.

[21] S. Vassiliadis, J. Phillips, and B. Blaner, "Interlock collapsing ALU's", IEEE Transactions on Computers, 42(7), 1993.

[22] X. Vera, O. Unsal, and A. Gonzalez, "X-Pipe: an adaptive resilient microarchitecture for parameter variations", Workshop on Architectural Support for Gigascale Integration, 2006.

[23] Y. Watanabe, N. Homma, T. Aoki, and T. Higuchi, "Arithmetic module generator based on arithmetic description language", 13th Workshop on Synthesis and System Integration of Mixed Information Technologies, 2006.