Cascading Dependent Operations for Mitigating Timing Variability

Watanabe, Shingo
Kyushu Institute of Technology

Hashimoto, Masanori
Osaka University

Sato, Toshinori
Fukuoka University, Kyushu University, JST(CREST)

出版情報：Workshop on Quality-Aware Design, 2008-06
バージョン：
権利関係：

Cascading Dependent Operations for Mitigating Timing Variability

Shingo Watanabe
Kyushu Institute of Technology

Masanori Hashimoto
Osaka University

Toshinori Sato
Fukuoka University
Kyushu University
JST, CREST

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. This paper investigates to exploit the statistical features in circuit delay and to cascade dependent ALU operations for reducing variations. From the statistical static timing analysis in circuit level and the performance evaluation in processor level, this paper tries to unveil how efficiently ALU cascading improves performance yield of processors. It is found that innovations are required for managing parameter variations in the microarchitecture level.

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[11]. 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 techniques in the microarchitecture level.

The following statistical features on circuit delay are well known[5][7]. 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. In order to exploit the statistical features in the microarchitecture level, the ALU cascading (or ALU collapsing[14]) is investigated. The ALU cascading is a technique to collapse dependent operations. Several operations are cascaded and executed in a single cycle using cascaded ALUs. Cascading ALUs 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 number 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 ALU 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 ALU cascading. Section 5 performs statistical static timing analysis (SSTA) of adders. Section 6 evaluates performance of a processor that utilize the ALU cascading. Section 7 estimates performance yield based on the results from the previous two sections. Finally, Section 8 concludes.

2. Related Works

Variation-aware designs for yield enhancement are a hot topic. However, to the best of our knowledge, it has not been considered yet to mitigate variability itself by some techniques in the microarchitecture level. The variable-latency adder[10], FPU[9], and register file[9] as well as ReCycle[13] are techniques to increase latency in order to tolerate large delay caused by variations. They do not control variability itself. Razor[6] and X-Pipe[15] are techniques to detect every timing violation caused by variability

and to revert processor state to a safe point where it is detected. They do not avoid timing violations. In contrast to the previously mentioned 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][7]. The horizontal axis indicates the delay and the vertical axis indicates the density. It is assumed the paths are completely independent and are modeled as normal distributions $N\left(6,1^2\right)$, 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 $N\left(6,1^2\right)$. 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.

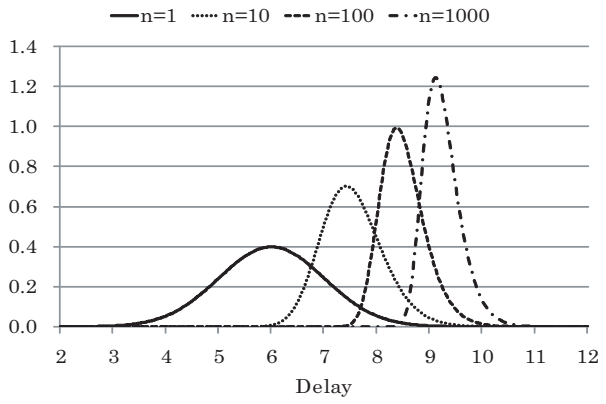

Figure 1. Dependence of circuit delay on the number of critical paths

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=1' 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.

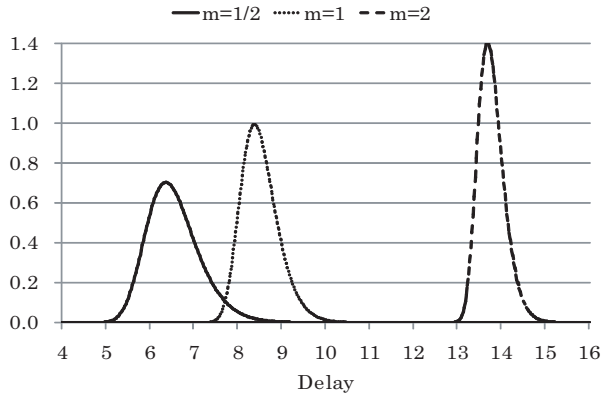

Figure 2. Dependence of circuit delay on logic depth

4. ALU cascading

In order to mitigate the variability in circuit delay, the use of ALU cascading is proposed by exploiting the statistical feature of circuit delay. As shown in Figure 3(a), the result of the producer operation is directly fed into the consumer operation. The cascaded ALU executes multiple dependent operations in a single cycle[14]. Cascading multiple ALUs increases the logic depth and thus mitigates delay variability.

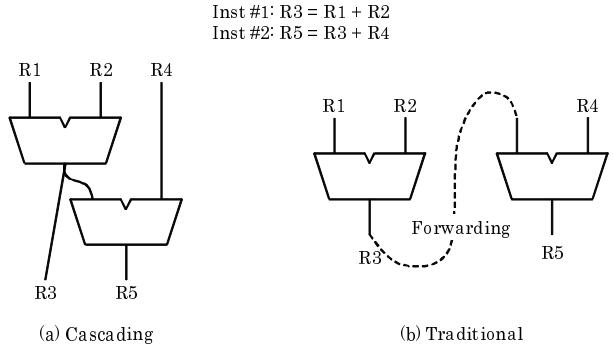

Figure 3. ALU cascading

Traditional superscalar processors can not execute multiple dependent operations in a single cycle, as shown in Figure 3(b). The ALU cascading improves IPC. However,

unfortunately, cascading ALUs increases circuit delay and that means clock frequency is declined. Hence, net processor performance might be degraded. Later, this paper evaluates processor performance considering both the increase in IPC and the decline in clock frequency.

5. Statistical Static Timing Analysis

A 32b carry select adder (CSLA) is selected for the SSTA. It is designed using Verilog-HDL and called CSLA2I in this paper. Two CSLA2Is are cascaded and form a 3-input 2-output adder. It is called CSLA3I in this paper. Synopsys DesignCompiler generates their netlists. Hitachi 0.18μm cell library is used. CSLA3I is synthesized after its hierarchy is flattened.

5.1. Methodology

The SSTA flow is shown in Figure 4. It consists of DesignCompiler and two in-house tools.

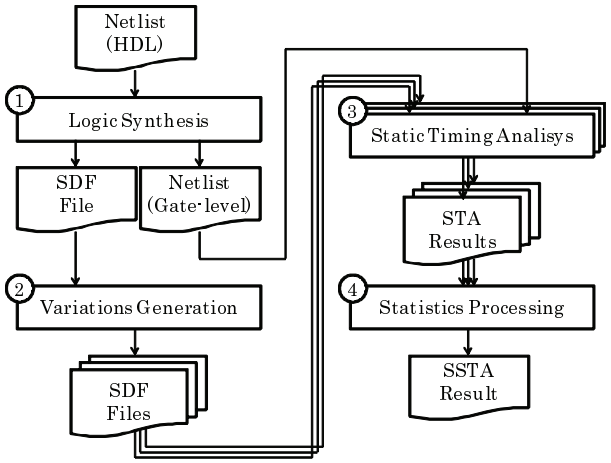

Figure 4. SSTA flow

1. A netlist and its associated SDF (standard delay format) file are generated. Synopsys DesignCompiler is used.
2. A lot of SDF files, where there are variations in gate delay, are generated from the original SDF file. An in-house tool, which generates variations, is used.
3. For each SDF file, a static timing analysis (STA) is performed on the netlist. DesignCompiler is used.
4. All STA results are statistically processed and an SSTA result is obtained. An in-house tool built on the top of MS Excel is used.

The followings explain how the SDF files are generated. 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/μ). The typical σ/μ value is 0.064 for 65nm technology[3]. Hence, σ/μ values of 0.040, 0.064, 0.080, and 0.100 are used in order to consider optimistic and pessimistic cases as well as the typical one. 10,000 samples are generated for every SSTA.

In the following analysis, σ/μ is used as a metric to evaluate how delay variability is mitigated.

5.2. Results

5.2.1 Dependence of path delay on gate delay variability

Figure 5 presents frequency distributions of maximum critical path delay for both CSLA2I and CSLA3I. 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 CSLA2I and those shown right are for CSLA3I. As the σ/μ value of gate delay increases, the mean delay, the standard deviation, and the σ/μ value of path delay increase.

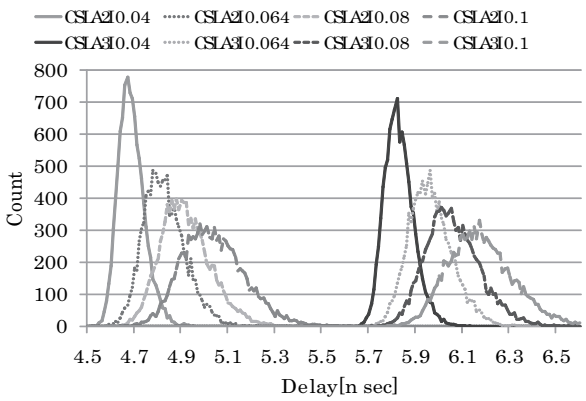

Figure 5. Dependence of path delay on gate delay variability

Table 1 summarizes μ, σ, and σ/μ of path delay. The fifth column in the figure explains how the σ/μ value of CSLA3I is smaller than that of CSLA2I. In other words, it presents the mitigation in delay variability. Since CSLA3I has larger path delay than CSLA2I does, both μ and σ are

larger in CSLA3I than in CSLA2I. An interesting observation is that σ/μ is reduced. That means cascading adders succeeds in mitigating delay variability.

Table 1. Influence of gate delay variability on μ, σ, and σ/μ of path delay

	Gate σ/μ	μ	σ	σ/μ	Imp.(%)
CSLA2I	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	–
CSLA3I	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

5.2.2 Dependence of path delay on the number of adders

As the number of adders increases, the number of critical path also increases. Hence, the path delay is dependent upon the number of adders. This section analyzes how it affects path delay variability. Hereafter, the path delay distributions analyzed in the previous section 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 adders are varied between 1 and 8. Since a CSLA3I consists of two CSLA2Is, the number of CSLA3I is varied between 1 and 4. The σ/μ value of 0.064 is used. The horizontal axis indicates the delay and the vertical axis indicates the delay density. Graphs shown left are for CSLA2I and those shown right are for CSLA3I. As the number of adders increases, the path delay variability, that is the σ/μ value of path delay, decreases.

Table 2 summarizes μ, σ, and σ/μ of path delay. In the case of CSLA2I, μ increases by 2.6% and σ/μ decreases by 40% when the number of CSLA2Is increases from 1 to 8. Similarly, in the case of CSLA3I, μ increases by 1.6% and σ/μ decreases by 31% when the number of CSLA3Is increases from 1 to 4. An interesting observation is that there are not any significant difference in σ/μ between CSLA2I and CSLA3I when the total number of component 2-input adders are same.

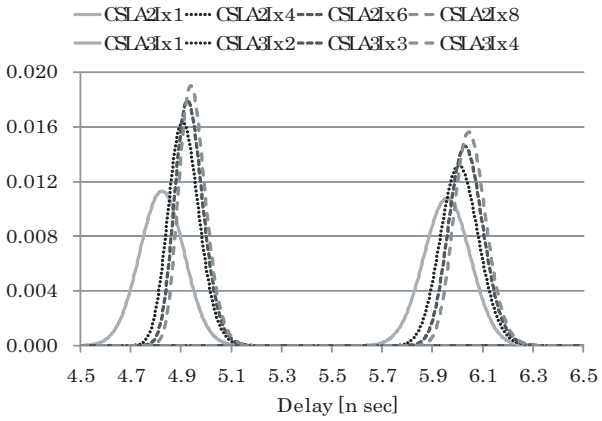

Figure 6. Influence of number of adders on circuit delay

Table 2. Influence of number of adders on mean and standard deviation of circuit delay

		μ	σ	σ/μ
CSLA2I	x1	4.82	0.088	0.0183
	x4	4.91	0.062	0.0126
	x6	4.93	0.057	0.0115
	x8	4.95	0.054	0.0109
CSLA3I	x1	5.96	0.093	0.0155
	x2	6.01	0.076	0.0127
	x3	6.04	0.069	0.0115
	x4	6.05	0.065	0.0107

6. IPC Performance Evaluation

This section evaluates how the cascaded ALUs improve processor performance in IPC. Hereafter, the processor utilizing the cascaded ALUs is called CASCADE processor.

6.1. Methodology

SimpleScalar tool set[1] is used for building the evaluation environment. Alpha instruction set is used. Twelve programs from SPEC2000 CINT are used as benchmark programs. First 500 million instructions are skipped and the following 100 instructions are simulated in detail. The configuration of the baseline processor is shown in Table 3. 4-, 6-, and 8-way instruction-issue processors are evaluated. The differences among these processors are the instruction issue width and the number of adders.

Table 3. Baseline processor configuration

Fetch width	8 instructions
L1 I-cache	32KB, 2way, 1 cycle
Branch predictor	gshare + bimodal gshare: 4K entries, 12 histories bimodal: 4K entries
RUU size	128 entries
Issue width	4/6/8 instructions
Integer ALUs	4/6/8 units, 1 cycle
Integer multipliers	1 units MULT 3 cycles, DIV 20 cycles
Floating ALUs	4 units, 2 cycles
Floating multipliers	1 unit MULT 4 cycles, DVI 12 cycles, SQRT 24 cycles
L1 D-cache port	2 ports
L1 D-cache	32KB, 4way, 2 cycles
Unified L2 cache	4MB, 8way, 8 cycles
Memory	150 cycles
Commit width	8 instructions

6.2. CASCADE Processor

Only integer ALU operations are candidates for cascading. The path delay distributions of adders, which are analyzed in Section 5 is used as those of ALUs. Hereafter, the conventional 2-input ALU is called ALU2I and the 3-input 2-output ALU, which consists of two ALU2Is, is called ALU3I. It is assumed that the execution stage determines clock frequency. This is similar to an assumption used in [8]. Since the typical delay of a CSLA3I is 25% larger than that of a CSLA2I, the clock frequency of CASCADE processor is 20% slower than that of the baseline processor. It should be noted that the timing constraints are relaxed for the pipeline stages other than the execution stage since the clock frequency is declined.

Cascading on ALU3I is operated as follows.

1. Only integer ALU instructions are candidates for cascading. Every load or store instruction is splitted into an address calculation operation and a memory access operation, and hence address calculation operations are included into the candidates.

2. Identifying candidate instructions is performed when a consumer instruction is dispatched into the instruction window. Its producer instruction is searched in the window. The term 'grouping' is used as identifying candidate instructions for cascading. If it has been already issued or finished, the consumer instruction is not cascaded and is called an ungroupable instruction.

3. Grouping is performed on two instructions. A pair of instructions are denoted as a group. Grouping 3 and more instructions is prohibited. Every instruction is included in at most one group.

4. Every consumer instruction that has only one unready operand is a candidate for grouping. If both operands are not ready, it is excluded from the candidates. This limitation is indispensable to avoid deadlock. Figure 7 shows two examples of deadlock. A circle, an arc, and a broken line indicate an instruction, a dependence between instructions, and a group, respectively. Since both instructions corresponding to the same group must be issued simultaneously, deadlock may occur. In the left case in Figure 7, instruction `I3` should be issued after `I2` is finished and `I2` should be issued after `I1` is finished. Hence, it is impossible to issue both `I1` and `i3` simultaneously. Similarly, in the right case in the figure, it is impossible to issue the groups (`I1`, `I3`) and (`I2`, `I4`).

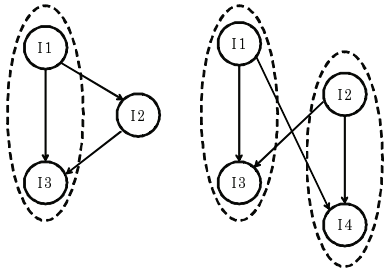

Figure 7. Deadlock

Table 4 summarizes the configuration of CASCADE processor. Only differences from Table 3 are shown. Considering the decrease in clock frequency, memory and L2 cache access latencies in clock cycle are reduced. There are two kinds of ALU configurations. One is homogeneous and has only ALU3Is. It is called Symmetry. The other is heterogeneous and has both ALU2Is and ALU3Is. It is called Asymmetry. When an ungrouped instruction is issued in Symmetry CASCADE processor, one 2-input component ALU in a ALU3I is wasted. On the other hand, if there are many groups, Asymmetry CASCADE processor degrades execution throughput due to the lack of ALU3Is. Next in this section, influence of these situations on IPC is evaluated.

6.3. Results

Figure 8 shows IPC. All values are normalized by IPC of the 4-way instruction-issue baseline processor. The horizontal axis indicates benchmark programs and the vertical

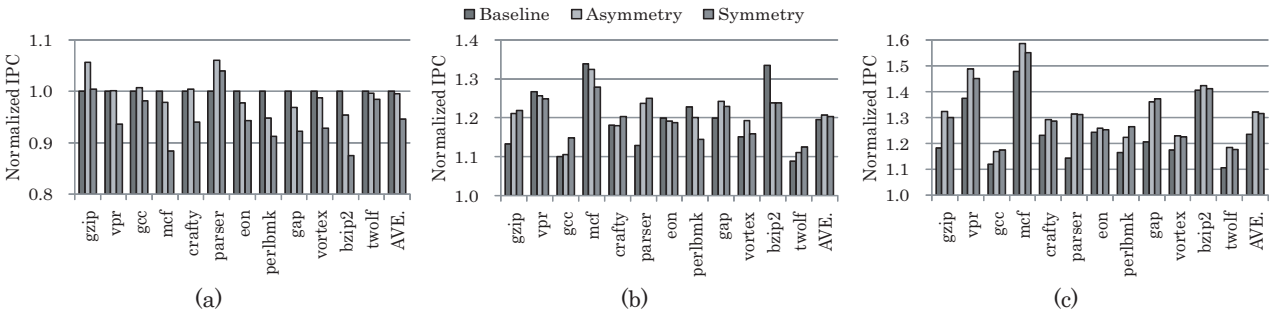

Figure 8. IPC (a) 4-way, (b) 6-way, (c) 8-way

Table 4. CASCADE processor configuration

		Symmetry	Asymmetry
4 issue	ALU2I	–	2 unit, 1 cycle
	ALU3I	2 units, 1 cycle	1 unit, 1 cycle
6 issue	ALU2I	–	4 units, 1 cycle
	ALU3I	3 units, 1 cycle	1 unit, 1 cycle
8 issue	ALU2I	–	4 units, 1 cycle
	ALU3I	4 units, 1 cycle	2 units, 1 cycle
Common	L2 cache	4MB, 8 way, 7 cycles	
	Memory	120 cycles	

axis indicates the relative IPC. In the figure, `Baseline`, `Asymmetry`, and `Symmetry` denote the baseline, Asymmetry CASCADE, and Symmetry CASCADE processors, respectively.

First, the effect of cascading on IPC is considered. According to the issue width, the effect is different. In the 4-way issue, IPC is smaller in CASCADE processors than in the baseline one for most programs. Asymmetry and Symmetry CASCADE processors diminishes IPC by 0.5% and 5.4% on average, respectively. A reason why IPC is diminished is that the 4-way baseline processor can efficiently exploit instruction level parallelism (ILP). The baseline processor can execute up to 4 instructions. In contrast, Asymmetry and Symmetry CASCADE processors can execute at most 3 and 2 instructions, respectively, if there are not any groups. This reduces execution throughput.

In the 6-way issue, a little improvement in IPC is observed. In Asymmetry and Symmetry CASCADE processors, IPC is increased by 1.0% and 0.6% on average, respectively. In the 8-way issue, a noticeable IPC gain is obtained. An average IPC improvements for Asymmetry and Symmetry CASCADE processors are 6.9% and 6.4%, respectively. A reason why IPC gain increases as the issue width increases is that the baseline processor can not exploit ILP when its issue width is large due to the lack of ILP. It wastes some ALUs. In the contrary, CASCADE processors can utilize ALUs even though ILP is small. In that situation, there are a lot of dependences between instructions. In other words, there are a lot of groups of cascaded instructions. Hence, CASCADE processors increase ILP by executing dependent instructions in a single cycle. From these observations, the ALU cascading is more efficient for wide issue processors than for narrow issue ones.

Next, the ALU configuration in CASCADE processors is considered. For all issue widths, Asymmetry CASCADE processor has larger IPC than Symmetry one does. Hence, ALU utilization may not good in Symmetry CASCADE processor. Figure 9 is the breakdown of instructions. The 4-way Asymmetry CASCADE processor is used. In the figure, `Prod` and `Cons` indicate the percentage of grouped instructions. `Prod` means producer instructions and `Cons` means consumers. `Ungrouped` indicates the percentage of ungroupable instructions. `Others` indicates instructions that are not candidates for grouping such as memory access operations. Grouped instructions occupy only 29% on average. In contrast, ungroupable instructions occupy 41% on average. For other processor configurations, the similar tendencies are observed. It is found that the percentage of grouped instructions is not so large. Since Asymmetry CASCADE processor has ALU2Is, which are efficiently utilized by ungroupable instructions, it has larger IPC than Symmetry one does. One of the reasons why the percentage of grouped instructions is small is that only instructions in the instruction window are checked for grouping. If the scope of grouping is increased by some technique such as the operand table[12], the percentage would be improved.

7. Performance Yield Estimation

Combining the SSTA results in Section 5 with the IPC results in Section 6 gives us estimations on performance yield of processors. Processor performance is obtained from IPC and clock frequency. Frequency distributions are

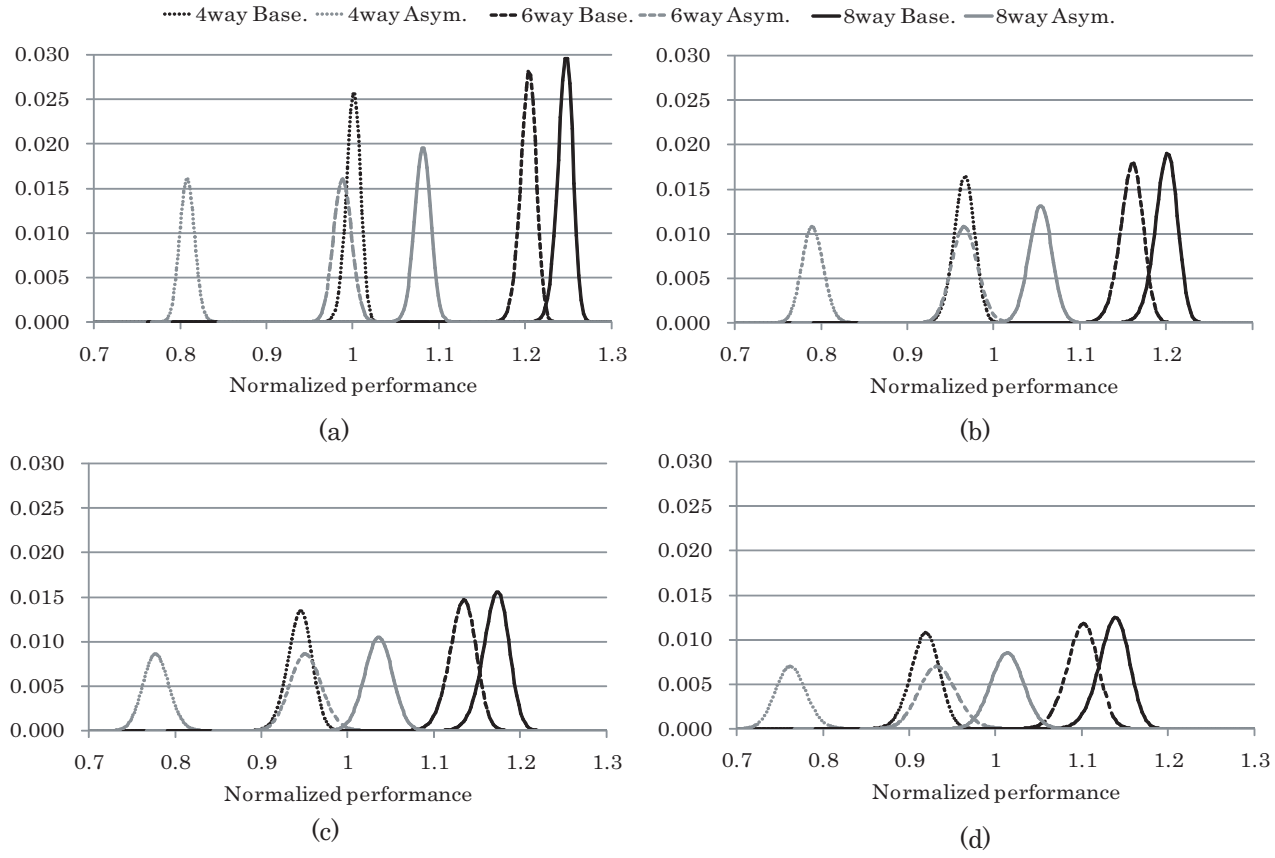

Figure 10. Processor performance distributions: σ/μ of gate delay = (a) 0.040, (b) 0.064, (c) 0.080, (d) 0.100

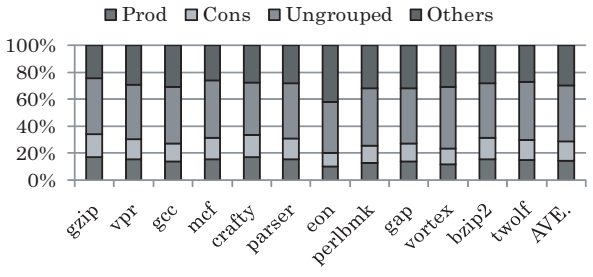

Figure 9. Breakdown of instructions

obtained from the SSTA results. Therefore, performance distributions are obtained from the SSTA and IPC results.

Figure 10 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 4-way baseline processor when the σ/μ value of gate delay is 0.040. Symmetry CASCADE processor results are not shown here, because its performance is always smaller than that of Asymmetry one.

First, the effectiveness of cascading on performance yield is investigated. For all issue widths and for all σ/μ values of gate delay, the baseline processor provides better performance than CASCADE processor does. Even though IPC is larger in CASCADE processor than in the baseline one in the cases of 6- and 8-instruction issue, net performance is better in the baseline processor than in CASCADE one. This is because the gain in IPC does not compensate for the decline in clock frequency. A surprising result is that σ/μ is larger in CASCADE processor than in the baseline one. In other words, timing variability in the processor level is not improved, even though that in the component level (i.e. adder) 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 the number of

critical paths of ALU2Is. On the other hand, that in CASCADE processor is determined by the number of critical paths of ALU3Is. When the issue width is same, the number of ALU2Is in the baseline processor is larger than that of ALU3Is in CASCADE processor, as shown in Tables 3 and 4. 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 is smaller than that of the increase in the number of critical paths, when a part of ALU2Is are replaced by ALU3Is.

Next, the influence of the gate σ/μ value on performance distributions of processor is investigated. As the gate σ/μ value increases, the mean delay of adder increases. Consequently, processor's clock frequency is declined, resulting in performance degradation. The influence differently appears between on the baseline and on CASCADE processors. The SSTA results in Section 5.2.1 explain the influence is more significant on CSLA2I than on CSLA3I. Hence, when the gate σ/μ value increases, the baseline processor suffers larger decline in clock frequency than CASCADE processor does. Comparing the 4-way baseline processor and the 6-way CASCADE one makes it clear to understand this situation. When the gate σ/μ value is 0.040, the 4-way baseline processor has better performance yield than the 6-way CASCADE one. As the value increases, the baseline's performance decreases. When it is 0.100, the 6-way CASCADE processor has better performance yield than the 4-way baseline one does.

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 for the decline in clock frequency. Second, since the dependence of path delay on the logic depth and that on the number of critical path are almost equivalent, the increase in logic depth on CASCADE processor is smaller than that of the increase in the number of critical paths on the baseline processor. 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.

8. Conclusions

This paper considered to utilize the cascaded ALU to improve performance yield of processors. The SSTA results explained that timing variability of an adder is mitigated. However, unfortunately, architectural-level performance evaluation showed the IPC gain is not enough to compensate for 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 unveil that fundamental changes are required to mitigate parameter variations in the microarchitecture level.

Acknowledgements

Hitachi 0.18μm standard cell libraries are provided by VDEC (VLSI Design and Education Center) in the University of Tokyo. This work is partially supported by Grant-in-Aid for Scientific Research (KAKENHI) (A)#19200004 and (B)#20300019 from Japan Society for the Promotion of Science, and by the CREST program of Japan Science and Technology Agency.

References

[1] T. Austin et al., "SimpleScalar: an infrastructure for computer system modeling", IEEE Computer, 35(2), 2002.

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

[3] K. Bernstein et al., "High-performance CMOS variability in the 65-nm regime and beyond", IBM Jour. of Res. and Dev., 50(4/5), 2006.

[4] S. Borkar et al., "Parameter variations and impact on circuits and microarchitecture", 40th Design Automation Conf., 2003.

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

[6] D. Ernst et al., "Razor: a low-power pipeline based on circuit-level timing speculation", 36th Int. Symp. on Microarchitecture, 2003.

[7] M. Hashimoto et al., "Increase in delay uncertainty by performance optimization", Int. Symp. on Circuits and Systems, 2001.

[8] H. Li et al., "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 Conf., 2006.

[9] X. Liang et al., "Mitigating the impact of process variations on processor register files and execution units", 39th Int. Symp. on Microarchitecture, 2006.

[10] D. Mohapatra et al., "Low-power process-variation tolerant arithmetic units using input-based elastic clocking", Int. Symp. on Low Power Electronics and Design, 2007.

[11] H. Onodera, "Variability: modeling and its impact on design", IEICE Trans. on Electronics, E89-C(3), 2006.

[12] P. G. Sassone et al., "Dynamic strands: collapsing speculative dependence chains for reducing pipeline communication", 37th Int. Symp. on Microarchitecture, 2004.

[13] A. Tiwari et al., "ReCycle: pipeline adaptation to tolerate process variation", 34th Int. Symp. on Computer Architecture, 2007.

[14] S. Vassiliadis et al., "Interlock collapsing ALU's", IEEE Trans. on Comput., 42(7), 1993.

[15] X. Vera et al., "X-Pipe: an adaptive resilient microarchitecture for parameter variations", Workshop on Architectural Support for Gigascale Integration, 2006.