Correlation-based Critical Path Predictors for Low Power Microprocessors

Akihiro Chiyonobu[1] Toshinori Sato[1,2] Itsujiro Arita[3]

[1] Department of Artificial Intelligence
[2] Center for Microelectronic Systems
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, 820-8502 Japan
{tsato,chiyo}@mickey.ai.kyutech.ac.jp

[3] Department of Intelligent Informatics
Kyushu Sangyo University
2-3-1 Matsukadai, Higashi-ku,
Fukuoka, 813-8503 Japan
arita@is.kyusan-u.ac.jp

Abstract

Recent studies show that the speed of microprocessors can be accelerated if we can identify which instructions are critical. Exploiting information regarding instruction criticality is effective not only for improving processor performance but also for improving energy efficiency. In this paper, we propose three critical path predictors in order to identify critical instructions. Experimental results demonstrate the future research direction of our approach.

Keywords: Critical Path, Low Power, Microprocessor

1 Introduction

Currently, smart mobile devices and embedded systems require high computing capability, thus employing high performance microprocessors. In addition, however, they require low power consumption as well as high performance. Because there is a tradeoff between power consumption and performance in microprocessors, power is the primary design constraint in embedded microprocessors for mobile devices. The active power P_{active} and gate delay t_{pd} of a CMOS circuit are given by

$$P_{active} \propto f C_{load} V_{dd}^2 \quad (1)$$

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \quad , \quad (2)$$

where f is clock frequency, C_{load} is load capacitance, V_{dd} is supply voltage, and V_{th} is the threshold voltage of the device. α is a factor depending upon the carrier velocity saturation and is about 1.3–1.5 in advanced MOSFETs. Eq.(1) shows clearly that power supply reduction is the most effective way to lower power consumption. However, Eq.(2) tells us that supply voltage reduction increases gate delay, resulting in a slower clock frequency. Thus, microprocessor performance is diminished. In order to solve this problem, we decided to exploit information regarding critical path in executing a program. Actually, a microprocessor has dual-power functional units. That means that it has several functional units distinguished in terms of their execution latency and power consumption, and that instructions on the critical path, which determines the execution time of the program, are executed in fast and power-hungry units and instructions on the non-critical path are executed in slow and power-efficient units. Using this scheduling strategy, microprocessor power consumption can be reduced while maintaining its performance. This paper describes a critical path predictor, which is an essential component for this dual-pipeline architecture.

The remainder of this paper is organized as follows: Section 2 explains how energy consumption can be reduced by using critical path information. Section 3 proposes correlation-based critical path predictors. Section 4 describes the evaluation methodology, and Section 5 presents simulation results. Section 6 surveys related works. Finally, Section 7 provides our conclusions.

2 Power Reduction via Critical Path Information

Most contemporary microprocessors execute instructions in an out-of-order fashion in order to reduce the execution time of a program [5]. The execution

time is determined by the microprocessor's computing capability and by dependences between instructions executed on the microprocessor.

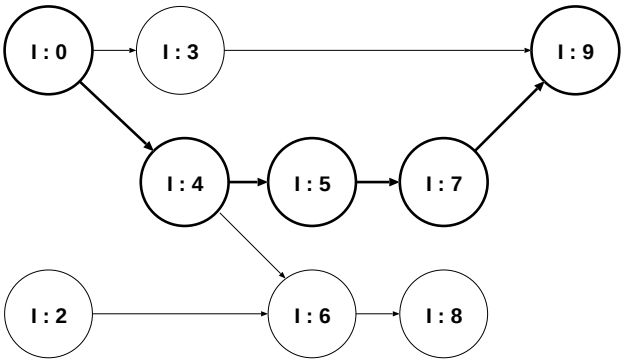


Figure 1: Critical path

The critical path is the longest path in a data flow graph, where each node represents an instruction and each arc represents a dependence between instructions, and it determines the execution time of the program [6]. Figure 1 shows an example of a data flow graph. In this example, its critical path consists of instructions `I:0->I:4->I:5->I:7->I:9` when every instruction's latency is assumed to be 1 cycle.

In order to identify if each instruction is critical or not, we utilize critical path predictors. Recently, critical path prediction has been proposed for improving instruction schedulers [4, 12]. One proposed by Tune et al. [12] predicts every instruction's criticality based on its past history regarding criticality. A critical path history table (CPHT) is its main component and consists of up-down counters indexed by the program counter (PC). At the commit stage, a counter is incremented if its associated instruction is critical. Otherwise, it is decremented. Later during instruction scheduling, the CPHT is referred to and if the counter value corresponding with the instruction is larger than the threshold, it is predicted as critical. Otherwise, it is predicted as non-critical. The bit width, the threshold for determining criticality, incremental value, and decrement value depend upon each implementation decision. In order to identify every instruction's criticality at the commit stage, several heuristics such as QOLD and ALOLD are proposed [12]. The QOLD and the ALOLD heuristics mark the oldest instruction in the instruction queue and active list, respectively, as critical. If we can identify which instructions are critical, we can accelerate their execution by any means [4, 6, 12].

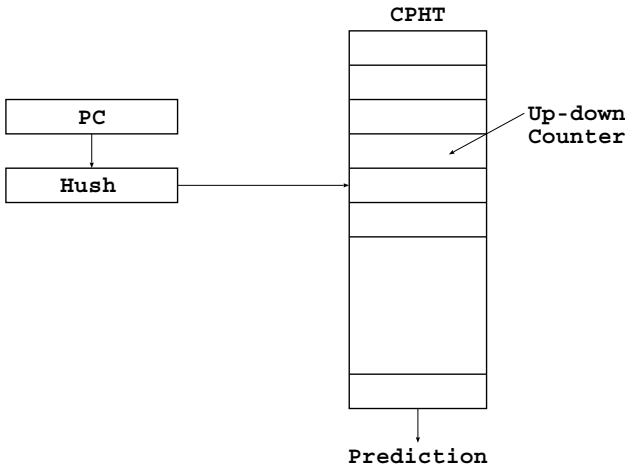


Figure 2: Tune's Critical Path Predictor

In order to reduce power consumption with maintaining performance, we decided to exploit the characteristics of the critical path. That is, instructions on a critical path determine the execution time of the program while other instructions do not [9, 10]. This is the key principle of this technique. If we can find non-critical instructions dynamically, they can be executed on slow and power-efficient functional units. Only critical instructions are dispatched into fast and power-hungry functional units [7, 9, 11]. Since it is expected that the execution cycle is not increased, energy consumption will be reduced.

3 Correlation-based Critical Path Predictors

Tune et al.'s critical path predictor [12] utilizes only the local history of each instruction. We assume that each instruction's criticality is affected by dependences between instructions and that prediction accuracy is improved by utilizing correlations between the histories of each instruction. We then proposed correlation-based critical path predictors [2, 3]. They are classified into three categories: GCPH-type, GBH-type, and BOTH-type predictors. The following sections describe them in detail.

3.1 GCPH-type predictor

In order to utilize correlations between the histories of each instruction's criticality, a shift register is added to the CPHT. This shift register keeps a history of the latest instructions' criticality. Each bit in the register

indicates if its corresponding instruction was critical at the last step. When the latest instruction's criticality is determined, its information is inserted to the top of the shift register, and information kept in the bottom of the register is removed. Because this shift register saves global history through the latest instructions instead of each instruction's local history, we named it the Global critical path history (GCPH) register. The GCPH-type correlation-based critical path predictor is shown in Figure 3. It is possible to utilize the correlation between instructions by maintaining the global criticality history in the GCPH register. Because each instruction's criticality affects its dependent instructions' criticality, the global history might be useful for accurate critical path prediction.

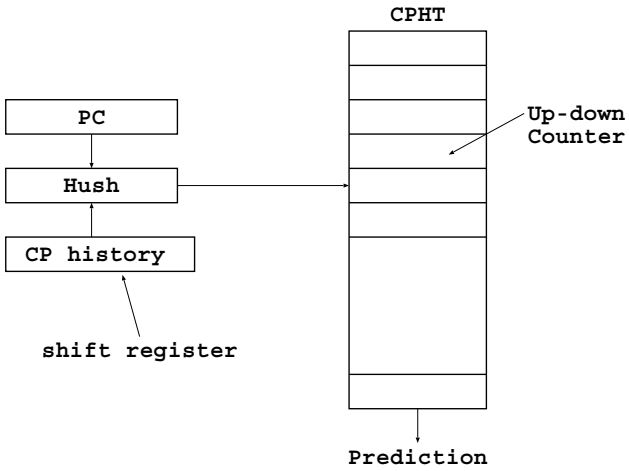


Figure 3: GCPH-type Critical Path Predictor

3.2 GBH-type predictor

Branch instructions reduce instruction-level parallelism in programs and thus diminishes processor performance. This is because it is impossible to find the target instruction of each branch instruction until it is executed. In order to solve this problem, many studies have addressed branch prediction [8]. One of the branch predictors is the correlation-based branch predictor. It saves the global branch outcome history in a shift register as does the GCHP-type critical path predictor. This register is called the Branch history register (BHR). We assume that branch outcome history affects every instruction's criticality and thus propose a correlation-based predictor utilizing branch outcome history. Figure 4 shows how the critical path predictor utilizes branch history. The index to CPHT is made from the PC and the BHR. We expect that this combination of global branch history and local criticality history will improve critical path prediction accuracy. We named it the GBH-type critical path predictor.

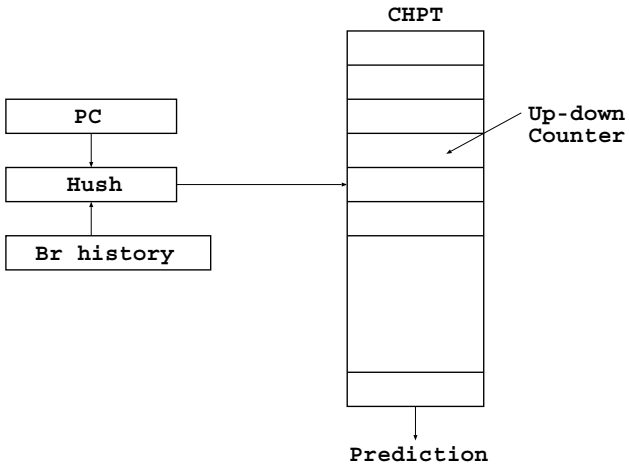


Figure 4: GBH-type Critical Path Predictor

3.3 BOTH-type predictor

In brief, we expect that utilizing both global criticality history and global branch history will improve critical path prediction accuracy. Hence, we propose to combine the GCPH-type predictor explained in Section 3.1 and the GBH-type predictor described in Section 3.2. Their combination might result in an effective synergy, and we named this type of predictor the BOTH-type critical path predictor. Figure 5 shows the BOTH-type predictor. The PC, the GCPH register, and the BHR are used for generating the index to CPHT.

3.4 Table-update-timing consideration

In order to achieve high prediction accuracy, the prediction table should be updated as soon as possible. Tune et al.'s critical path predictor updates its CPHT at the commit stage. Therefore, critical path information regarding instructions in the instruction window cannot be utilized. In other words, the predictor does not exploit the latest information concerning instruction criticality. This delay regarding critical path information might have serious impact on instruction scheduling for the sake of power reduction. To solve this problem, we propose to update the CPHT as soon as each instruction's criticality is found.

Using this technique, the latest information regarding instruction criticality can be utilized in the current

Table 1: Processor model

Fetch Bandwidth	8 instructions
Branch Predictor	1K-set 4-way set-associative BTB, 4K-entry 8-history-length gshare predictor, 64-entry return address stack, 6-cycle miss penalty, updated at commit stage
Insn. Windows	32-entry instruction queue, 32-entry load/store queue
Issue Width	8 instructions
Commit Width	8 instructions
Functional Units	6 Int, 3 FP, 4 Ld,St
Latency (total/issue)	iALU 1/1, iMUL 8/1, iDIV 32/1, fADD 4/1, fCMP 4/1, fCVT 4/1, fMUL 4/1, fDIV 32/1, fSQRT 32/1, Ld/St 2/1
Register Files	32 32-bit Int registers, 32 32-bit FP registers
Insn. Cache	64KB, 2-way, 64B blocks, 18-cycle miss penalty
Data Cache	64KB, 2-way, 64B blocks, 4-port, write-back, non-blocking load, hit under miss, 18-cycle miss penalty
L2 Cache	unified, 1MB, 4-way, 64B blocks, 80-cycle miss penalty

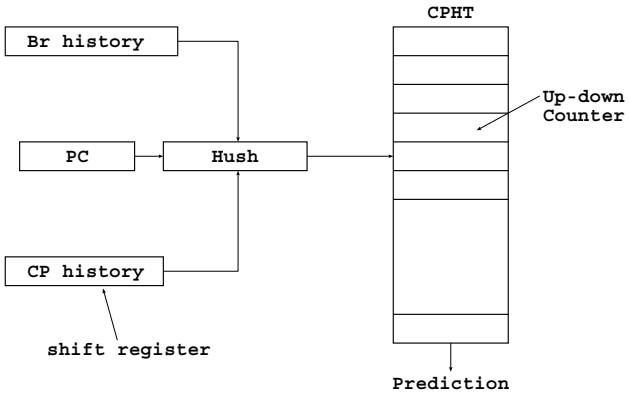


Figure 5: BOTH-type Critical Path Predictor

instruction's criticality prediction. However, its implementation involves several issues. First, because every instruction's criticality is checked while it remains in the instruction window, there is a possibility that its associated entry in the CPHT will be updated multiple times. The entry has to be updated only once for a dynamic instance of the instruction. Thus, we decided to mark the instruction when its associated entry is updated. Second, if branch misprediction occurs and the instruction is thus removed, its criticality history maintained in the CPHT is not accurate. This might lead to low critical path prediction accuracy. Thus, in the case of branch misprediction, the state of the CPHT is recovered to the point before the branch instruction was mispredicted. This can be realized by checkpointing in the same manner as does the register renaming table.

4 Evaluation Methodology

This section describes our processor model and benchmark programs, which are used in simulations, in order to explain what environment is used for our evaluation.

We use the sim-outorder simulator from the SimpleScalar tool set (version 3.0b) [1] to implement our simulator. It is a cycle-by-cycle simulator. The proposed correlation-based predictors are included in the simulator in detail. The CPHT sizes are varied between 2K and 64K. It is a direct-mapped table of 6-bit saturating up-down counters. When an instruction is identified as critical according to the QOLD or the ALOLD heuristics [12], its associated counter is incremented by 8. Otherwise, it is decremented by 1. The counter threshold at which an instruction is predicted as critical is set to 8. The criticality history length saved in the GCPH register is 8 instructions. The incremental value, the threshold value, and the history length were determined based on preliminary simulations.

The fast functional units can execute most integer operations in one cycle, while the slow functional units execute operations in two cycles. In the rest of this paper, *functional units* means *integer units*. Both the fast and slow units share their circuit design, while each transistor's size and threshold voltage might be optimized independently. We assume that the supply voltage for the slow units are half of that for the

fast units. Our processor model has 3 fast functional units and 3 slow units. Table 1 shows the processor's configuration.

Instruction set architecture (ISA) is the SimpleScalar/PISA ISA, which is an extension of MIPS R10000 ISA. The SPEC2000 CINT benchmark suite is used for this study. Table 2 lists the benchmarks and the input sets. For each program, 1 billion instructions are skipped before the actual simulation begins, and the next 100 instructions are executed. We do not count NOP instructions.

Table 2: Benchmark programs

Benchmark	*input set*
164.gzip	input.compressed
175.vpr	net.in arch.in
176.gcc	cccp.i
197.parser	test.in
255.vortex	lendian.raw
256.bzip2	input.random

5 Simulation Results

Figures 6—9 show the simulation results. We use relative processor performance and relative energy-delay products (EDP) as metrics for our evaluation. For each group of two figures, the left one shows relative performance and the right one shows relative EDP. For performance, the higher the bar, the more effective processor's performance. For EDP, the lower the bar is, the higher power efficiency. For each group of six bars, the first four bars from left to right are for Tune et al.'s predictor, and GCPH-type, GBH-type, and BOTH-type predictors, respectively. The bar labeled `6fast/0slow` is for the model whose functional units are all fast ones, and the bar labeled `0fast/6slow` is for the model whose functional units are all slow ones.

5.1 Baseline

First, as a baseline we evaluate 64K-entry CPHT using the QOLD heuristic with updating at the commit stage. Figure 6 shows the results. First, in comparison with the `6fast/0slow` processor model, processor performance is degraded by approximately 20% for all critical path predictors. Next, the correlation-based predictors are compared with Tune et al.'s predictor. The GBH-type and the BOTH-type show improved EDP, while processor performance is slightly diminished. In contrast, the GCPH-type improves performance while EDP is diminished. This is because the GCPH-type tends to predict more instructions as critical. In summary, the difference between the correlation-based and Tune et al.'s predictors is small.

5.2 Effect of heuristics

This section investigate how heuristics for identifying each instruction's criticality affects processor performance and EDP. In this evaluation, the QOLD heuristic is replaced by the ALOLD one. Figure 7 shows the results. It is found that the ALOLD heuristic tends to identify more instruction as critical than does the QOLD one, and thus fast functional units are used more often, resulting in an improvement in performance and a degradation in EDP. Processor performance is improved by 2% on average, while EDP is diminished by 7% on average. From these results, we conclude that the QOLD is better than the ALOLD.

5.3 Effect of update timing

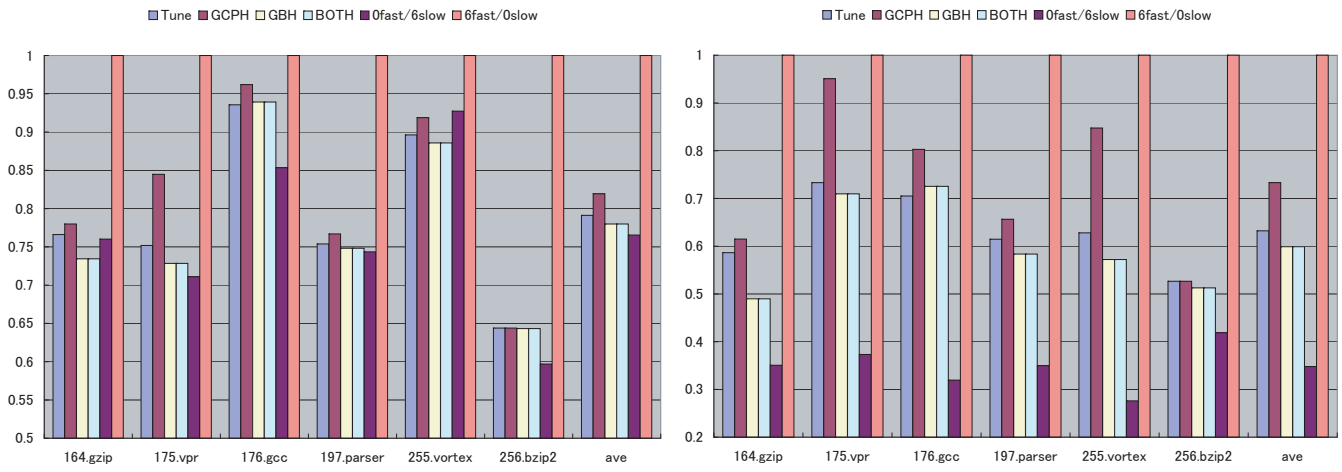
Next, we evaluate how update timing of the CPHT affects performance and EDP. In this evaluation, predictors are speculatively updated at the execution stage. Figure 8 shows the results. A comparison with Figure 6 shows that performance is reduced by 4% on average, while EDP is significantly improved by 17% on average. This means the improvement in EDP is much bigger than the degradation in performance. Hence, we conclude that the speculative update is effective.

5.4 Effect of table size

Finally, we evaluate how the prediction table size affects performance and EDP. Figure 9 shows the results when the table size is reduced from 64K to 2K. Processor performance improved by 2% on average, while EDP is increased by 5% on average. This is due to aliasing in the table. Small table causes many conflicts, and results in lower prediction accuracy. This effect is more significant in the correlation-based predictors than in Tune et al.'s predictor.

5.5 Summary

As observed in the previous sections, we could not so far achieve both high performance while consuming low amounts of power. Improving EDP diminishes performance, and conversely good performance leads to bad EDP. We found that the QOLD heuristic is better than the ALOLD one, that speculative update of the CPHT is effective, and that a relatively large table is required for the correlation-based predictor.



(i) performance (ii) EDP

Figure 6: 64K-entry, QOLD, update-at-commit

6 Related Work

The critical path is a chain of dependent instructions, which determines the number of cycles executing the program. And thus, the performance of the processor is limited by the speed at which it executes the instructions along the critical path. If we can identify which instructions are critical, we can accelerate their execution by any means. Kobayashi et al. [6] uses instruction criticality for instruction steering policy and proposed the path info table, which saves a dynamically-generated data flow graph. They calculate the length of each critical path based solely on the data dependence graph. Critical path prediction [4, 12] is another technique for identifying critical instructions dynamically.

Exploiting information regarding instruction criticality is effective not only for improving processor performance but also for reducing power consumption [7, 11]. Pyreddy et al. [7] use the profile-based heuristics proposed by Tune et al. [12] for identifying critical instructions. From a profile run, each instruction is marked as critical or non-critical. When the program is executed, the critical instructions are executed on fast and power-hungry functional units while the non-critical ones are executed on slow and power-efficient units. They concluded that dual pipeline had the potential for low power without suffering performance loss, but did not make any measurement of power savings. In contrast, Seng et al. [11] utilized a dynamic mechanism. They proposed to use the critical path predictor to identify non-critical instructions, and reported significant gains in the ratio of performance and power density. However, they did not mention how energy consumption can be reduced.

7 Conclusions and Future Study

In this paper, we have proposed a correlation-based critical path predictor to improve prediction accuracy. From the evaluations presented in this paper, unfortunately, we could not derive a significant improvement in prediction accuracy from Tune et al.'s CPHT [12]. However, we produced some new findings as follows. First, we found that the QOLD heuristic is effective than the ALOLD one in almost every case. Second, speculative update of the CPHT is more effective than an update at the commit stage. Third, a relatively large table is required for the correlation-based critical path predictor.

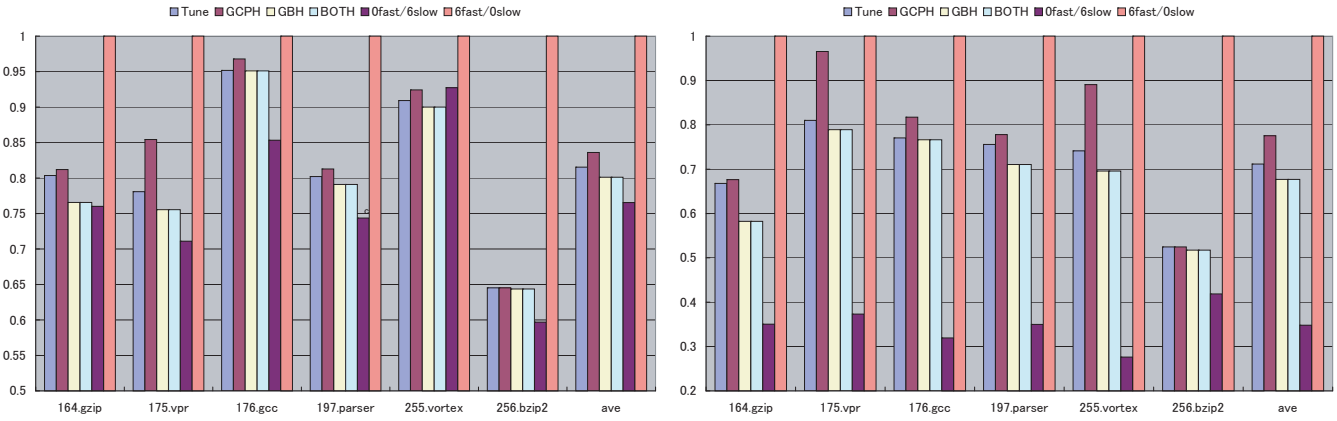
A future study regarding critical path prediction involves considering heuristics for identifying every instruction's criticality. We are also interested in the theoretical limit of the dual-pipeline architecture on power reduction while maintaining processor performance.

Acknowledgments

This work is supported in part by the Grants-in-Aid for Scientific Research (No.13558030) from the Japan Society for the Promotion of Science.

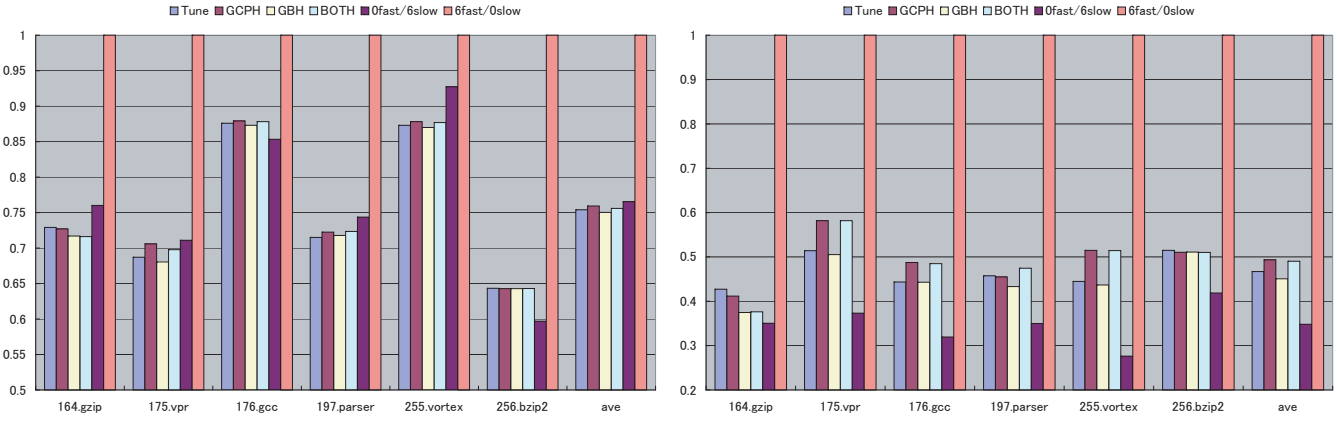
References

[1] D. Burger, T. M. Austin: "The SimpleScalar Tool Set, Version 2.0", Technical Report CS-TR-97-



(i) performance (ii) EDP

Figure 7: 64K-entry, ALOLD, update-at-commit



(i) performance (ii) EDP

Figure 8: 64K-entry, QOLD, speculative update

1342, Computer Science Department, University of Wisconsin Madison, June 1997.

[2] A. Chiyonobu: "Study on Critical Path Predictors for a Low Power Processor Architecture", Bachelor's Thesis, Kyushu Institute of Technology, February 2002 (in Japanese).

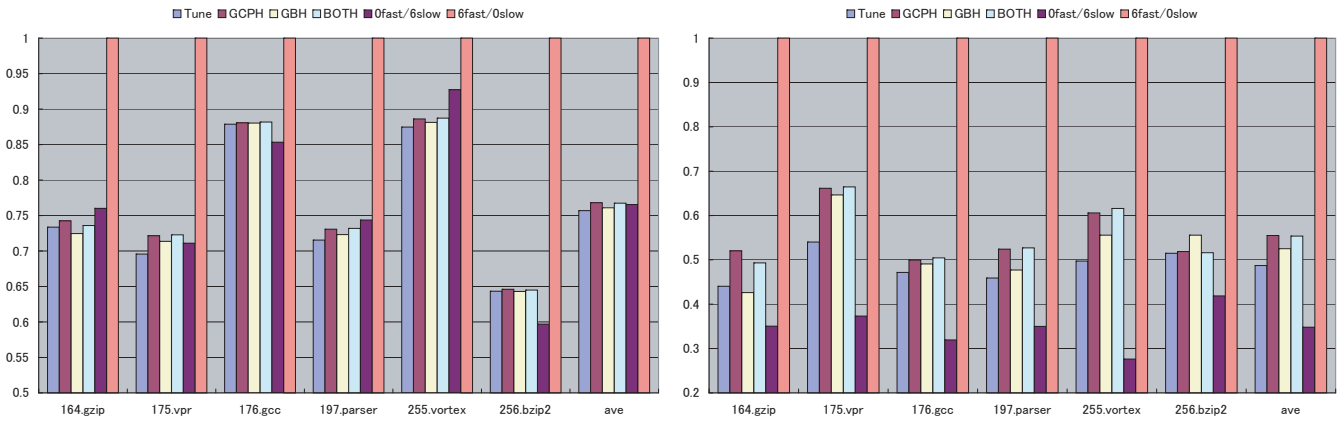
[3] A. Chiyonobu, T. Sato, I. Arita: "A Proposal of Critical Path Predictors for Low Power Processor Architecture", 15th Summer United Workshop on Parallel, Distributed, and Cooperative Processing, August 2002 (in Japanese).

[4] B. Fileds, S. Rubin, R. Blodik: "Focusing Processor Policies via Critical-Path Prediction", 28th International Symposium on Computer Architecture, July 2001.

[5] M. Johnson: "Superscalar Microprocessor Design", Prentice Hall, 1991.

[6] R. Kobayashi, H. Ando, T. Shimada: "Instruction-Issue Mechanism for a Clustered Superscalar Processor Focusing on a Critical Path in a Data Flow Graph", 13th Joint Symposium on Parallel Processing, June 2001 (in Japanese).

[7] R. Pyreddy, G. Tyson: "Evaluating Design Tradeoffs in Dual Pipelines", Workshop on Complexity-Effective Design, June 2001.



(i) performance

(ii) EDP

Figure 9: 2K-entry, QOLD, speculative update

[8] F. Saito, H. Yamana: "The Latest Technical Trends in Speculative Execution", IPSJ SIG Notes, 2001-ARC-145-11, November 2001 (in Japanese).

[9] T. Sato, T. Koushiro, A. Chiyonobu, I. Arita: "Power and Performance Fitting in Nanometer Design", 5th International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, January 2002.

[10] T. Sato, A. Chiyonobu, I. Arita: "Energy Reduction via Critical Path Prediction", Workshop on Complexity-Effective Design held in conjunction with 29th International Symposium on Computer Architecture, May 2002.

[11] J. S. Seng, E. S. Tune, D. M. Tullsen: "Reducing Power with Dynamic Critical Path Information", 34th International Symposium on Microarchitecture, December 2001.

[12] E. Tune, D. Liang, D. M. Tullsen, B. Calder: "Dynamic Prediction of Critical Path Instructions", 7th International Symposium on High Performance Computer Architecture, January 2001.