Investigating Heterogeneous Combination of Functional Units for a Criticality–based Low–power Processor Architecture*

Akihiro Chiyonobu† Toshinori Sato†‡

† Department of Artificial Intelligence
‡ Center for Microelectronic Systems
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, 820-8502 Japan

Abstract

In recent years, advanced applications that require high processing performance are executed on portable and mobile devices. Those devices require high–performance and low–power processors. To satisfy this requirement, we propose to exploit information regarding instruction criticality. In order to reduce energy consumption with maintaining high performance, each functional units in our processor has different latency and energy consumption. This paper describes the best combination of functional units for the criticality–based low–power processor. The evaluated results show two fast and four slow combination is the best, when the total number of its functional units is six.

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

$$P_{active} \propto f C_{load} V_{dd}^2 \tag{1}$$

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

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[7]. 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

*This work is supported in part by the grant from Kitada Shogakukai Kinen Zaidan (No.03–003).

path are executed in slow and power-efficient units. Using this scheduling strategy, microprocessor power consumption can be reduced while maintaining its performance. Differences from the previous studies on utilizing instruction criticality are as follows. Tune et al.[9] and Kobayashi et al.[5] focus on performance improvement. Seng et al.[8] attack peak power but don't consider total power; i.e. energy. In construct, we are interested in energy reduction because energy rather than power is more important for battery-operated devices. Thus, our goal is reducing energy consumption of microprocessors with maintaining their performance. In this paper, we consider the most suitable combination of fast units and slow units.

2 Criticality–based Low–power Processor Architecture

2.1 Energy-Aware Instruction Scheduling

Most contemporary microprocessors execute instructions in an out-of-order fashion in order to reduce the execution time of a program. The execution time is determined by the microprocessor's computing capability and by dependences between instructions executed on the microprocessor. 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[5]. Figure 1 shows an example of a data flow graph(DFG). In this example, its critical path consists of instructions `I:0->I:3->I:4->I:6->I:7` when every instruction's latency is assumed to be 1 cycle.

We propose that a microprocessor 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, we can reduce microprocessor energy consumption while maintaining its performance.

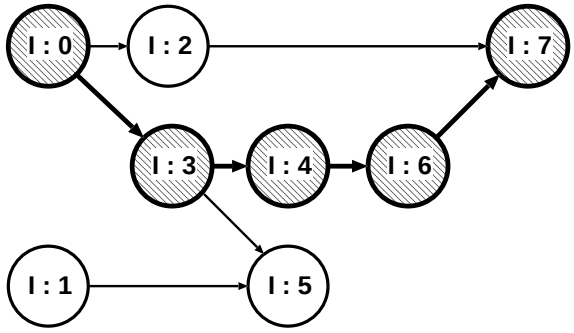

Figure 1: Critical path

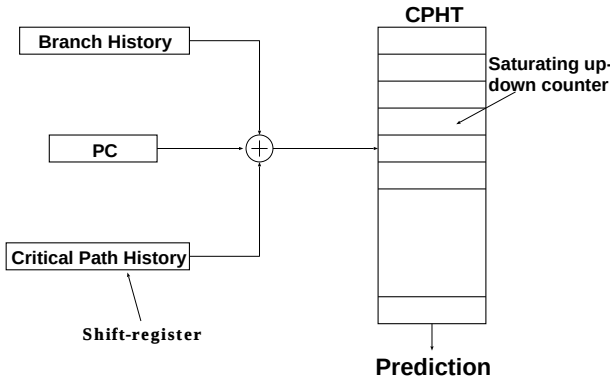

Figure 2: BOTH–type predictor

2.2 Critical Path Predictors

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, 9].

A critical path predictor proposed by Tune et al.[9] 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). Each 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 are proposed[9].

Tune et al.'s critical path predictor 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]. They are classified into three categories: GCPH-type, GBH-type, and BOTH-type predictors.

In order to utilize correlations between the histories of each instruction's criticality and branch histories, two shift registers are added to the CPHT. One shift register keeps the global branch outcome history (GBH) which is utilized by branch predictors. The other saves a history of the least instructions' criticality. Because this shift registers save global history through the latest instructions instead of each instruction's local history, we named it global critical path history (GCPH) register. Each bit in the GCPH register indicates if its corresponding instruction was critical at the last time. This information is inserted to the top of the shift register, and information kept in the bottom of register is removed. In GCPH-type predictor, the index to CPHT is made from GCPH and PC. The GBH-type predictor utilizes GBH and PC at its prediction time. The BOTH-type predictor exploits both GCPH and GBH. It is the combination of GCPH-type predictor and GBH-type predictor. The combination might result in an effective synergy. The BOTH-type predictor is shown in Figure2. We expect using those new information will improve critical path prediction accuracy.

2.3 The best combination of Functional Units

The combination of functional units is one of the key points in our energy–aware instruction scheduling. This is because replacing power–hungry units with power–efficient ones as much as possible with maintaining performance results in larger energy reduction. We have not known what is the best combination for high performance and power efficiency yet. So, we investigate the best combination of functional units for criticality–based low–power architecture.

3 Evaluation Methodology

In order to investigate what configuration of functional units is best in energy efficiency, we conducted a simulation study. We implemented a timing simulator using the SimpleScalar/PISA tool set[1]. It is a cycle-by-cycle simulator. Our evaluated processor has a 32–entry instruction queue, a 32–entry load/store queue, a 4K–entry 8–history–length gshare branch predictor, and 64KB 2–way set–associative instruction and data cache memories.

The proposed correlation-based predictors are included in the simulator in detail. In this paper, we use 2K-entry predictor. It is a direct-mapped table of 6-bit saturating up-down counters. When an instruction is identified as critical according to the QOLD heuristics[9], its associated counter is incremented by 8. Otherwise, it is decremented by 1. The counters are updated speculatively at the issue stage. 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* (ALU). We assume that the supply voltages for fast and slow units are assumed to be 1.1V and 0.7V, respectively[6]. To investigate the best ALU combination, we decrease the number of fast ALU from six to zero.

The SPEC2000 CINT benchmark suite is used for this study. For each program, 1 billion instructions are

skipped before the actual simulation begins, and the next 100 million instructions are executed. We do not count NOP instructions.

4 Simulation Results

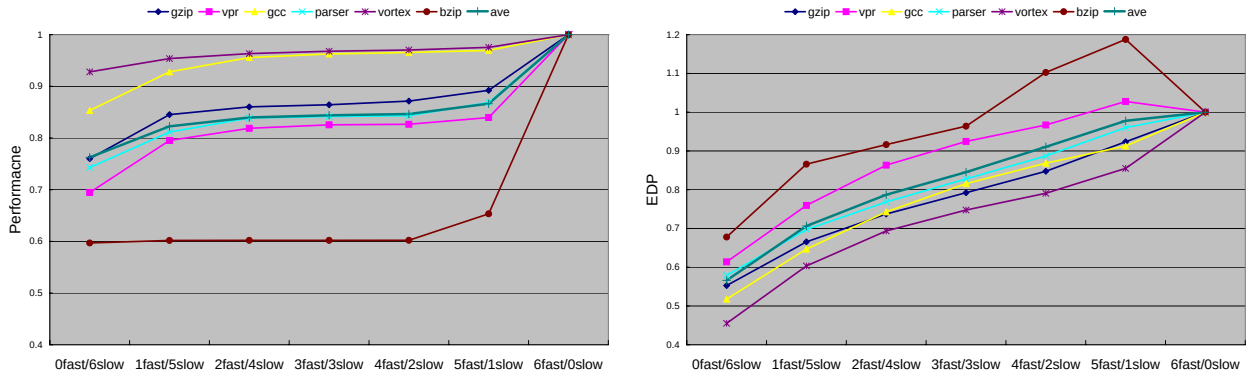

Figure 3: Per-address predictor (TUNE[9])

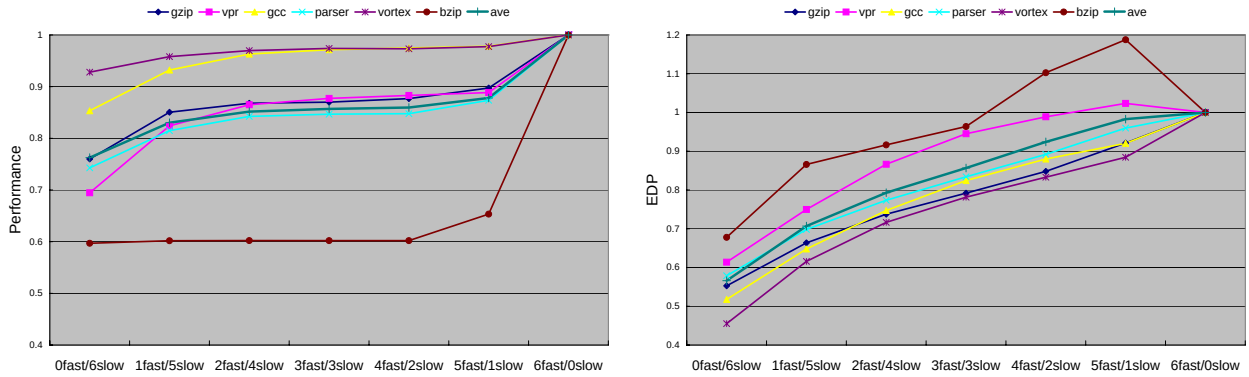

Figure 4: GCPH–type predictor

In this section, we show our simulation results. Our evaluation results are shown in Figures 3 – 6. The left part of each group of two figures shows processor performance, and the right part shows energy delay product (EDP). On performance, the higher is better. In contrast, the lower is better on EDP. The horizontal line indicates combinations of functional units. *n*`fast/`*m*`slow` means that the combination has *n* fast and *m* slow ALUs.

First, we consider the processor performance. We observe the processor performance is diminished, when the number of fast ALUs is reduced. We find that `2fast/4slow`, `3fast/3slow`, and `4fast/2slow` combinations mark the equivalent performance, which is approximately 15% below that of the `6fast/0slow` (baseline) model.When the number of fast ALUs are fewer than 2, however, the processor performance is seriously degraded.

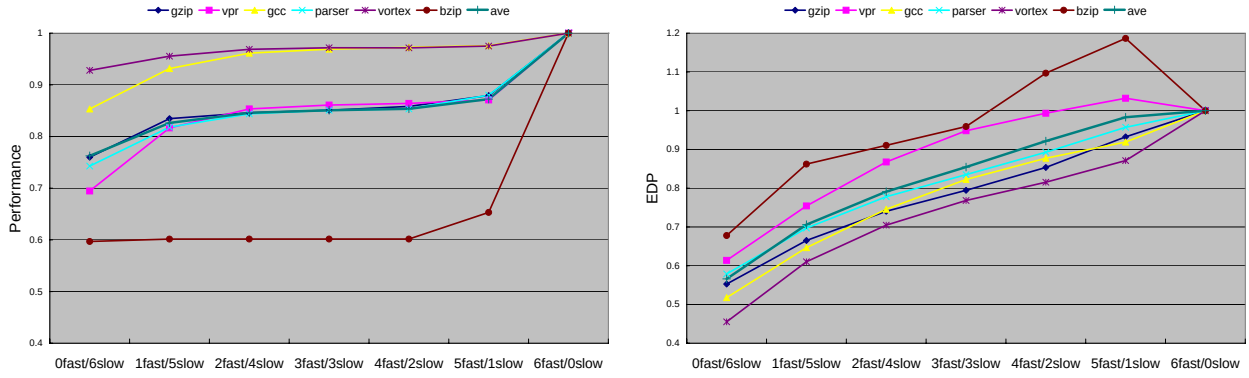

Figure 5: GBH-type predictor

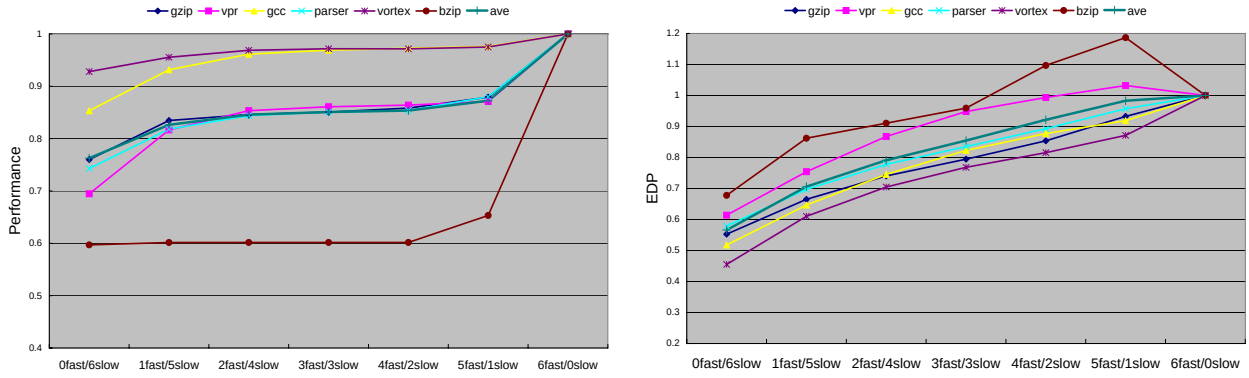

Figure 6: BOTH-type predictor

Next, we consider EDP. When the number of fast ALUs is decreased, EDP is improved in most cases. The noticeable exception is `bzip`. Its EDP is degraded from the baseline in the combinations of `5fast/1slow` and `4fast/2slow`. In these cases, the combination of the fast and slow units is so unbalanced in comparison with that expected by critical path prediction, and thus processor performance is significantly diminished, resulting in larger energy consumption due to long execution time.

From the considerations above, we can conclude that the case of `2fast/4slow` is the best combination, because it achieves good improvement in EDP with little degradation in performance. This results differ from our past evaluation results[3] using more accurate critical path information obtained from path information table (PIT)[5]. This difference is very interesting. We are investigating where the difference come from and how it affects energy efficiency.

5 Conclusions

In this paper, we investigated the best ALU combination in the criticality-based low-power processor architecture. From the detailed simulations, we found that the combination of two fast and four slow units is the best in the case where processor has six ALUs.

We have some future studies. First, we should examine other ALUs that have longer execution latency for further energy reduction. Second, we have to improve critical path prediction accuracy. As mentioned above, the best combination is different between the cases where the PIT and the CPPs are utilized for identifying instruction's criticality. We have already found that the PIT does good job on identifying criticality. Thus, we guess that the difference in the best combination is due to low prediction accuracies of the critical path predictors. We believe our criticality-based low-power architecture has better executing performance and more energy efficiency if our critical path predictor is more accurate. Therefore, we have to investigate how to improve critical path prediction accuracy. Third, we will try to expand the application range of criticality-based scheduling. We are planning to exploit instruction's criticality for the following applications; instruction scheduling, low power cache architectures, branch prediction, value prediction, thread prediction, and thread scheduling. Those research results will be shown in the near future.

References

[1] D. Burger, T. M. Austin: "The SimpleScalar Tool Set, Version 2.0", Technical Report CS-TR-97-1342, Computer Science Department, University of Wisconsin Madison, June 1997.

[2] A. Chiyonobu, T. Sato, I. Arita: "An Evaluation of Critical Path Predictors for Low Power Processor Architecture", Transactions of IEICE C, Vol.J86-C, No.8, August 2003 (in Japanese).

[3] A. Chiyonobu, T. Sato: "On Dynamic Identification of Instruction Criticality", 15th Summer United Workshop on Parallel, Distributed, and Cooperative Processing, August 2003 (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] 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).

[6] M. Levy: "SAMSUNG Twists ARM Past 1GHz", Information Quarterly, vol.1, no.1, 2002.

[7] 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.

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

[9] 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.