Exploring Configuration of Dual Speed Pipelines for Criticality–based Energy–efficient Processors

Akihiro Chiyonobu[1] Toshinori Sato[1,2]
[1]Department of Artificial Intelligence, Kyushu Institute of Technology
[2] Center for Microelectronic Systems, Kyushu Institute of Technology
680–4 Kawazu, Iizuka, Fukuoka, 820–8502, Japan

Abstract

Intelligent mobile information devices require low–power and high–performance processors. In order to reduce energy consumption with maintaining computing performance, we propose to utilize information regarding instruction criticality. Microprocessors have two types of functional units distinguished in terms of their execution latency and power consumption. Only critical instructions are executed on power–hungry functional units, and the total energy consumption can be reduced. In order to achieve large energy reduction, it is required to replace power–hungry units with power–efficient units as many as possible. In this paper, we explore what configuration of functional units is best in energy efficiency. From detailed simulations, we find the configuration of three fast and three slow units is best in the case where the processor has six units.

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 addition, it is required that the threshold voltage is proportionally scaled down with the supply voltage in order to maintain high transistor switching speeds. The leakage power can be given by

$$P_{off} = I_{off} V_{dd} \tag{3}$$

where I_{off} is the leakage current. The subthreshold leakage current I_{off} is dominated by threshold voltage V_{th} in the following equation:

$$I_{off} \propto 10^{-\frac{V_{th}}{S}} \tag{4}$$

where S is the subthreshold swing parameter and is around 85mV/decade[9]. Thus, lower threshold voltage leads to increase subthreshold leakage current and then static power. Maintaining high transistor switching speeds via low threshold voltage gives rise to a significant amount of leakage power consumption.

In order to solve the problems, 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 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.[10] 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.

This paper is organized as follows. Section 2 explains how to exploit dynamic information regarding instruction criticality for energy reduction. Section 3 provides the motivation of this study on investigating configuration of the dual speed pipelines. Section 4 explains a mechanism to identify critical path. Section 5 describes the simulation methodology and tools. Section 6 presents simulation results and discusses what configuration of functional units is best in energy efficiency. Section 7 concludes.

2 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 (DFG), 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 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.

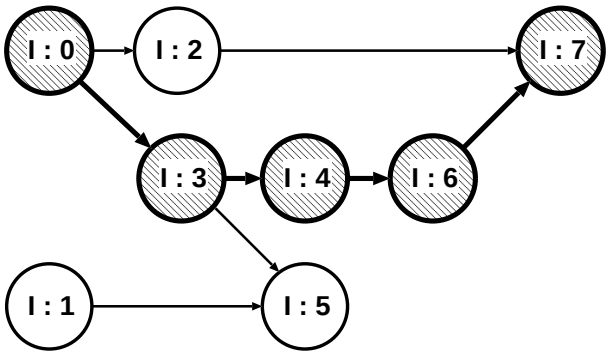

Figure 1. Critical Path

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.

In order to benefit from the energy–aware scheduling, there are two key components, power–efficient functional unit and mechanism to identify critical path. The following sections provide them in details.

3 Power–efficient Functional Unit

Utilizing slow and power–efficient functional units for executing instructions on non–critical path is the key idea behind the proposed architecture. Power–efficient units are realized by the following circuit designs. Simple and slow circuit such as ripple carry adder consumes less power than complex and fast circuit such as carry lookahead adder. Delivering lower supply voltage with slow units while sharing a single circuit design for both units improves energy efficiency of slow units. Reducing threshold voltage diminishes circuit speed performance but decreases power due to leaking current.

The configuration 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 configuration for high performance and power efficiency yet. So, we investigate the best configuration of functional units for the energy–aware instruction scheduling.

4 Identifying Critical Instructions Method

In order to identify if each instruction is critical or not, we utilize path information table (PIT) proposed by Kobayashi at el.[5]. The reason why we do not use critical path predictors[2, 10] is that we are interested in the effect of configurations on energy efficiency in the case where critical paths are ideally identified. We have already found that critical path predictors do not have enough prediction accuracies in comparison with PIT[3]. PIT create a DFG in the instruction window every cycle. We explain how PIT expresses DFG.

(1) At every node where some edges meet, only the latest edge on execution time is kept and the remaining edges are removed from the current DFG, resulting the DFG is transformed to a tree from a DAG.

(2) The transformed tree is expressed as follows. First, the tree is disassembled to partial paths. Second, a FIFO keeps the node included in each partial path.

Table 1. Processor Configuration

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 regsters
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

Joint information between the partial paths is kept in PIT. PIT is utilized to search the longest path to terminal point from the starting point.

(3) The latest instruction in the instruction window is kept separately.

To make DFG as above, the decoded instruction is inserted in the FIFO entry immediately after the instruction that makes its source operands or in the empty FIFO entry. And a table with the entry corresponding to each instruction is prepared. The table holds a pointer to the instruction which defines the operand and the execution finishing time of the instruction. Every dispatched instruction refers this table in order to find the instruction which defines the operand and the execution finishing time of the instruction. If some operands exist, the instruction which has the latest execution finishing time is chosen. Based on this information, the instruction is inserted behind the dependence instruction. To get the execution finishing time, the time when the operands are generated and the execution latency of this instruction are added. If the execution finishing time of this instruction is later than the current latest execution finishing time, it is replaced by the new one. To find the instructions along the longest path, the DFG is scanned from the terminal point of the longest path using information of joint between paths.

5 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. PIT explained in Section 4 is implemented in detail. Table 1 shows the processor's configuration. 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). In this evaluation, we assume the followings. Our processor model has six ALUs. Both the fast and slow ALUs can share their circuit design, while each transistor's size and threshold voltage might be optimized independently. Power consumption due to leakage current is out of consideration. It is remained for the future study. The supply voltages for fast and slow ALUs are assumed to be 1.1V and 0.7V, respectively[6].

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 million 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

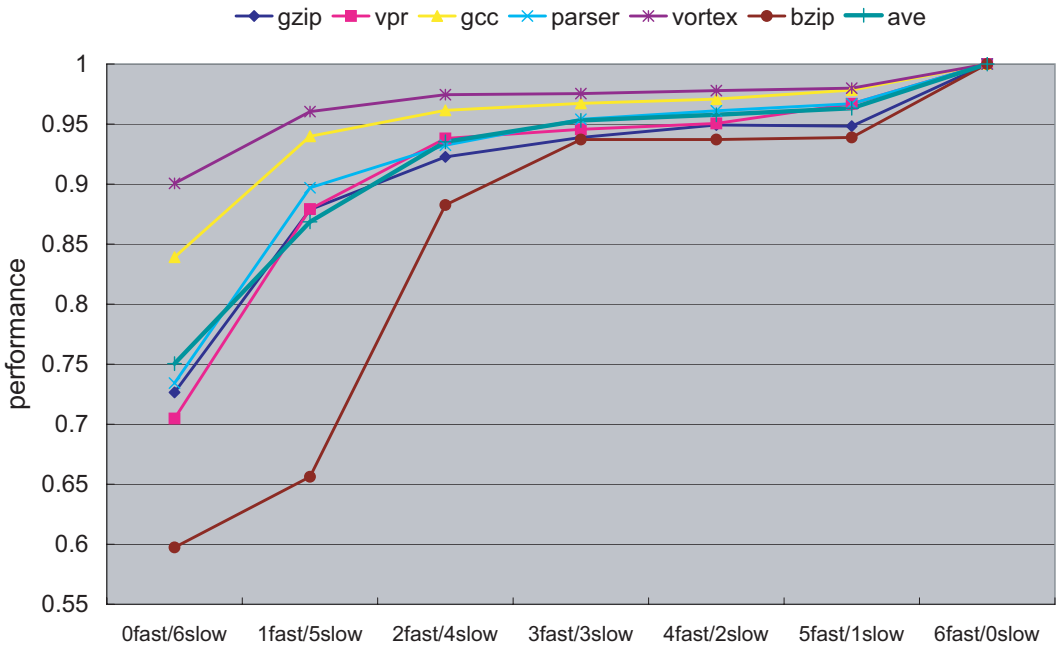

Figure 2. Performance Results

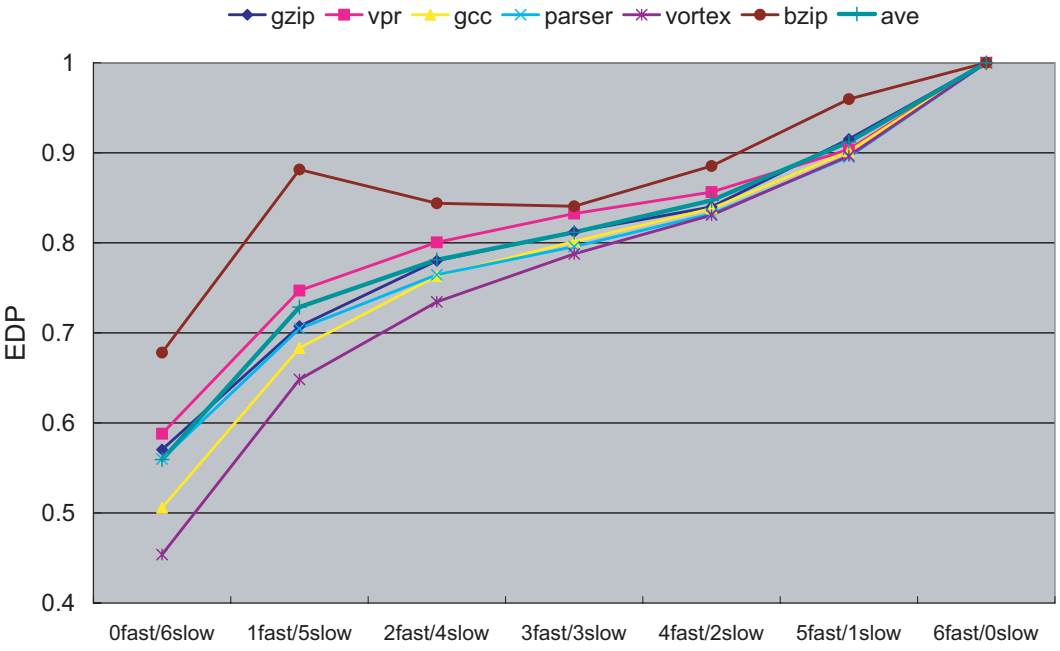

Figure 3. EDP Results

6 Evaluation Results

Our evaluation results are shown in Figures 2 and 3. The higher the value is, the better performance is. The lower the value is, the better energy delay product (EDP). In those figures, `0fast/6slow` means that all ALUs are slow. We increase the number of fast ALUs to six. They are denoted as `1fast/5slow`, `2fast/4slow`, `3fast/3slow`, `4fast/2slow`, `5fast/1slow`, and `6fast/0slow`.

First, we consider the processor performance. When the number of fast ALUs becomes fewer, we observe the processor performance is monstrously diminished. If we paid our attention to the average value, we recognize the `3fast/3slow` case is the best. This is because the performance loss is suppressed to about 5%. When the number of fast ALUs are fewer than 3, the processor performance is significantly degraded.

Next, we think about EDP in the average value. When the number of fast ALUs decreases, EDP is getting better in most cases. We think that the reason why EDP is getting worse in `1fast/5slow` on `bzip` is increasing execution time. We observe `3fast/3slow` keeps EDP about 20% loss. If fast ALU decrease under three, EDP is worse significantly.

As mentioned above, we think that the case of three fast and three slow units is the best combination. Because it can prevent degradations in performance and EDP the minimum.

There are some previous studies investigating configurations of dual speed pipelines. In [8], Seng et al. conclude using just a single functional units is in a good tradeoff point. The different conclusion from ours is due to the different focuses of the two studies. Seng et al. are interested in hot spots on a chip and try to reduce peak power consumption. In contrast, our goal is reducing total power consumption; i.e. energy. Sato et al. find that three fast units are required in order to mitigate performance loss[7]. They determine the configuration by only considering execution cycles. Power and energy efficiency is ignored for the exploration. Our previous study [4] unveils that utilizing only two fast units is in best trade off point. All these studies above rely on critical path predictors for determining which instructions are critical. In the current paper, we use PIT which accurately identifying instruction criticality. We have already found that critical path predictors do not have enough prediction accuracies in comparison with PIT[3]. We guess the different conclusions are due to the low prediction accuracy.

7 Conclusions

In this paper, we investigated the best ALU combination in criticality based energy–efficient architecture. In the result of examination, we conclude that the combination of three fast and three slow units is the best in the case where processor has six ALUs.

We have some future studies. First, we should examine other ALU configurations. Second, we have to propose any simple mechanism to identify critical path. We estimate that PIT is power hungry, because it has many complex operations. We have to estimate the energy consumption of PIT. Third, we will try to expand the application range of criticality based scheduling. Because we are also interested in criticality based processor architecture. We are planning the application to the following portions. There are instruction scheduling, cache architecture, branch prediction, value prediction, thread prediction, and thread scheduling. Those research results will be shown in the near future.

Acknowledgments

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

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: “Correlation-based Critical Path Predictors for Low Power Microprocessors,” 6th International Workshop on Innovative Architecture for Future Generation High–Performance Processors and Systems, January 2003.

[3] A.Chiyonobu, T.Sato: “On Identifying Instruction Criticality for Energy-Aware Applications,” 12th International Conference in Parallel Architecture and Compilation Techniques, September 2003.

[4] A.Chiyonobu, T.Sato: “Investigating Heterogeneous Combination of Functional Units for a Criticality–based Low–power Processor Architecture,” 3rd International Symposium on Information and Communication Technologies, June 2004.

[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] D. Sylvester, H. Kaul: “Power-driven challenges in nanometer design,” IEEE Design & Test of Computers, vol.18, no.6, 2001.

[10] 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.