On Identifying Instruction Criticality for Energy-Aware Applications

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

Abstract

Recent studies show that microprocessor performance or its energy efficiency can be improved, if we utilize information regarding instruction criticality. This paper investigates why we could not achieve low energy consumption and high performance simultaneously, and reveals there is a fundamental problem in heuristics to identify critical instructions.

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

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

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

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[2]. 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.[9] and Kobayashi et al.[6] focus on performance improvement. Seng et al.[8] attacks 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. Unfortunately, however, it has not been achieved[3, 4]. Thus, this paper investigates why instruction scheduling utilizing critical path information does not improve energy efficiency with maintaining processor performance.

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, 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(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-

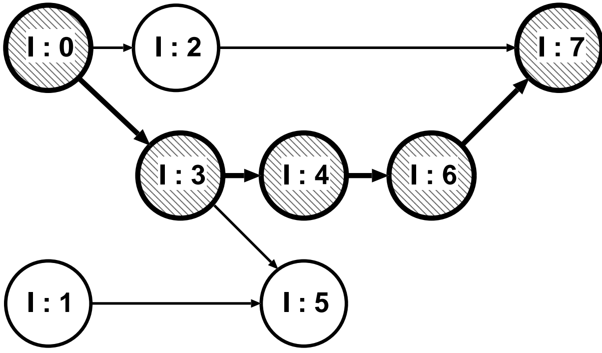

Figure 1. Critical Path

efficient units. Using this scheduling strategy, we can reduce microprocessor energy consumption while maintaining its performance. This paper describes critical path identification methods, which are an essential component for this dual-pipeline architecture.

3 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[5, 9]. This section describes critical path predictors.

3.1 Per-address Critical Path Predictors

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), as shown in Figure 2. 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]. If we can identify which instructions are critical, we can accelerate their execution by any means[5, 6, 9].

3.2 Correlation-based Critical Path Predictors

Tune et al.'s critical path predictor[9] utilizes only the local history of each instruction. We assume that each instruction's criticality is affected by dependences between

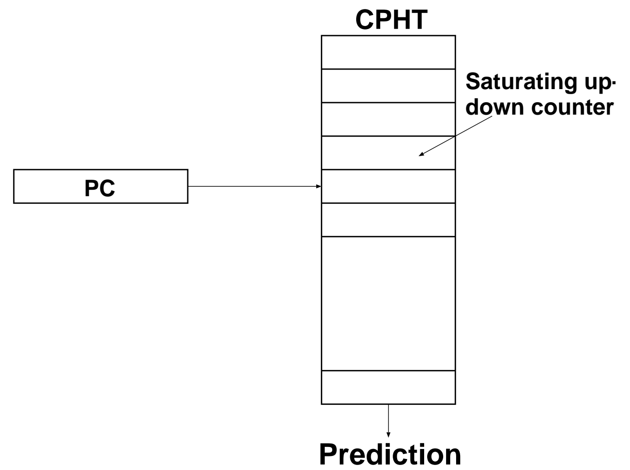

Figure 2. Tune's Critical Path Predictor

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. The following sections describe them in detail.

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.

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. 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-

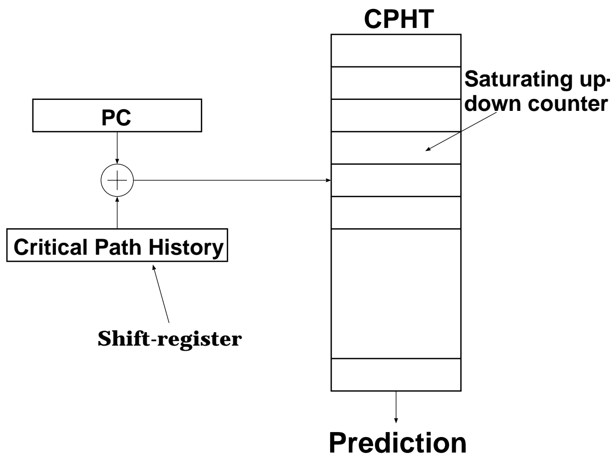

Figure 3. GCPH-type Critical Path Predictor

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.

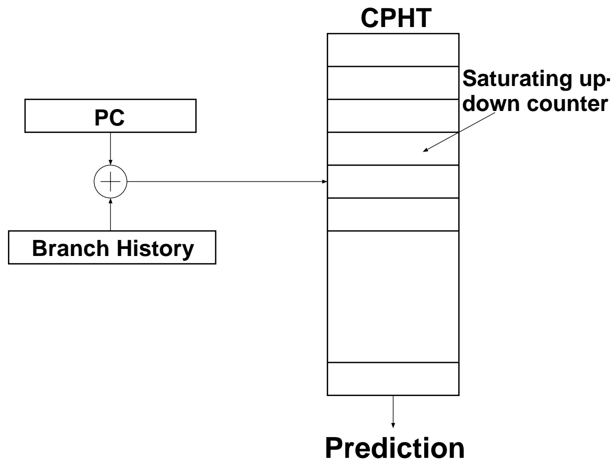

Figure 4. GBH-type Critical Path Predictor

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 and the GBH-type predictor. 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.

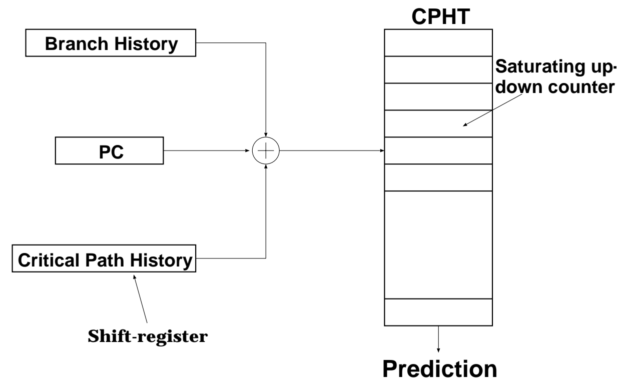

Figure 5. BOTH-type Critical Path Predictor

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. In this papaer, we use 2K-entry GCPH-type 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, ALOLD, QCONS, or FREED3 heuristics[9], its associated counter is incremented by 8. Otherwise, it is decremented by 1. We will present simulation results using the QOLD heuristics, when we do not specify which heuristics is used. 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*. 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 voltages for fast and slow units are assumed to be 1.1V and 0.7V, respectively[7]. 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

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

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

5 Difficulty in Identifying Critical Instructions

In order to utilize information regarding instruction criticality, we rely on critical path predictor (CPP)[2, 9]. Unfortunately, its accuracy is not enough high to achieve our goal. We have already known that we can perform low energy consumption and maintain processing performance if exact critical path information is used. Figure 6 shows processor performance and energy delay product (EDP). The first bar is for our baseline model, which does not exploit instruction criticality, and the second one is for the model utilizing CPP. The third one shows processor performance and EDP, if path information table (PIT)[6] is used for identifying critical instructions. PIT obtains accurate critical path information, since it creates DFG in the instruction window for every cycle. We guess the energy consumed in PIT is very large, however, because of its complexity in hardware. In construct, the CPP's hardware structure is very simple, and it is estimated its energy consumption is 1.7% of 64KB data cache. Hence, it is better to improve prediction accuracy of the CPP rather than utilizing PIT.

We guess that the low prediction accuracy of the CPP is due to the information which trains the predictor. Identifying critical instructions is based on heuristics, such as QOLD, ALOLD, QCONS, and FREED3, which were proposed by Tune at el.[9]. We compare the identification results of heuristics with those of PIT. Surprisingly, we find approximately 40% of identification based on the heuristics does not agree with PIT, as shown in Figure 7. And therefore, almost half of predictions does not agree with PIT. In order to confirm that the low prediction accuracy is due to the heuristics, we replace the training heuristics with PIT. That is, the CPP is trained by PIT and instructions are scheduled based on critical path prediction. The last bar in Figure 6 shows processor performance and EDP in this case. While processor performance is improved, EDP is getting worse. This means that maintaining processing performance is realized by utilizing fast and power-hungry units. This result shows prediction accuracy is not improve even if the CPP is updated using exact critical path information. A reason why the CPP still can not perform accurate predictions might be that each predicted criticality never changes while PIT-reported criticality follows dynamic changes in DFG.

6 Conclusions

The evaluation performed in this paper shows that the energy-saving architecture is effective when exact critical path information is utilized. Unfortunately, however, PIT which can supply accurate critical path information is expected to be very power-hungry since hardware of PIT is

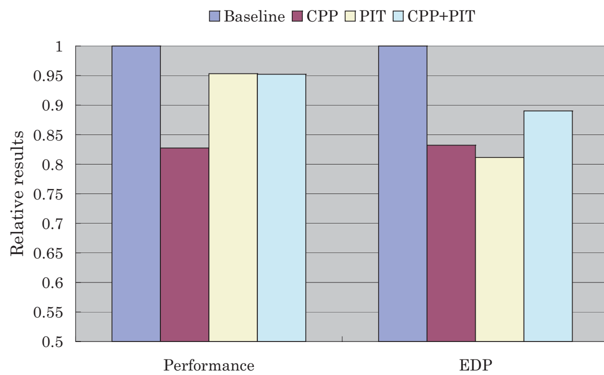

Figure 6. Simulation Results

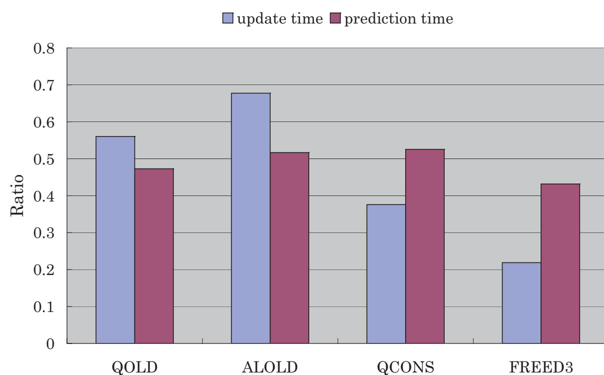

Figure 7. Agreement Ratio

complicated. Therefore, we have to investigate why critical path prediction accuracy is not improved, because the CPP are more desirable than PIT due to its simple hardware and energy efficiency.

Acknowledgments

This work is supported in part by the grants from Japan Society for the Promotion of Science (No.13558030) and 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: "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] A. Chiyonobu, T. Sato, I. Arita: "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.

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

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