Power and Performance Fitting in Nanometer Design

Toshinori Sato[1,2] Takenori Koushiro[1]
Akihiro Chiyonobu[1] Itsujiro Arita[1]

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

Abstract

Power consumption is becoming one of the most important constraints for microprocessor design in nanometer-scale technologies. Device engineers, circuit designers, and system architects are faced with many challenges. In the area of mobile and embedded computer platforms, power has already been a major design constraint. However, it is also a limiting factor in general-purpose microprocessors. In order to manage the impact of increasing microprocessor power consumption, some architectural-level techniques are required as well as circuit-level design improvements. This paper proposes criticality-based instruction scheduling and Contrail processor architecture, which utilize the criticality of instructions, and demonstrates their effectiveness.

Keywords: energy reduction, dual-voltage pipeline, critical path prediction, contrail processors, power estimation

1 Introduction

The increasing popularity of portable and mobile computer platforms such as laptop PCs and smart cell phones is a driving force in the investigation of high-performance and power-efficient microprocessors. For example, modern embedded microprocessors support out-of-order execution [9]. As the computing power of microprocessors for mobile devices increases, however, their power consumption also increases. In addition, while power is already a major design constraint in the area of mobile and embedded computer platforms, it has also become a limiting factor in general-purpose microprocessors.

The energy consumed in a microprocessor is the product of its active power and execution time. Thus, to reduce energy consumption, we should decrease either or both of them. The active power P_{active} and gate delay t_{pd} of a CMOS circuit are given by

$$P_{active} = fC_{load}V_{dd}^2 \quad (1)$$

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

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 [7]. Based on Eq.(1), it is easily found 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, the computing performance of the microprocessor is diminished.

In order to mitigate the performance loss, we can exploit information regarding circuit criticality. Design tradeoffs can be achieved using many device-level techniques such as transistor size optimizations [6], multiple supply voltages [22], and multiple threshold [24] approaches. Power reduction without performance loss is achieved by selecting non-critical paths as candidates for low-power design. In other words, performance-oriented design is used only in speed-critical paths. The same philosophy can be applied to architectural-level design. We can exploit dynamic information regarding instruction criticality in order to reduce power. We predict critical sections and selectively attack non-critical ones for the sake of lowering power consumption. Furthermore, we aggressively create non-critical instructions, and thus the efficiency of power reduction is improved. This paper introduces

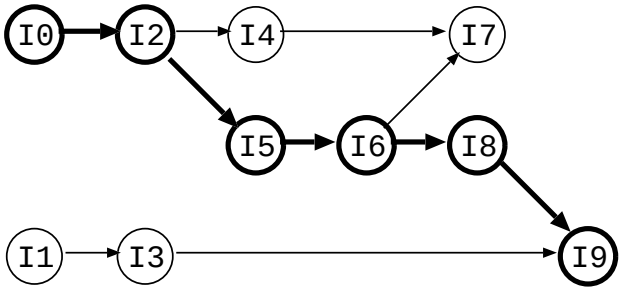


Figure 1: Data flow graph

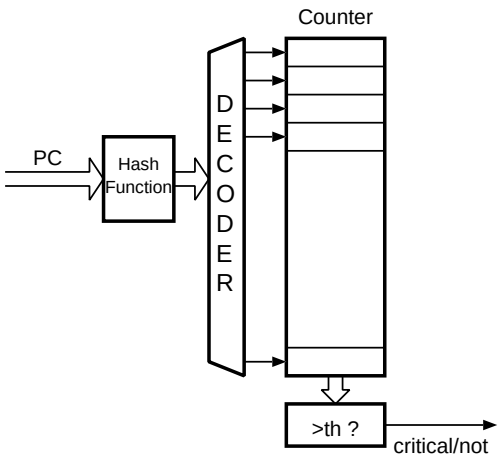


Figure 2: Critical path prediction buffer

Contrail processor architecture [16] as a promising solution in high-performance and energy-efficient microprocessor design.

The reminder of this paper is organized as follows: Section 2 explains how energy consumption is reduced by critical path prediction. Section 3 describes the Contrail processor architecture. Finally, Section 4 provides our conclusions.

2 Energy Reduction via Critical Path Prediction

As explained in the previous section, we exploits dynamic information regarding instruction criticality in order to reduce power. This is based on critical path prediction [5, 21]. In this section, we describe how to reduce power using the critical path prediction.

2.1 Architecture overview

Figure 1 is a data flow graph, where each node represents an instruction and each arc represents a dependence between instructions. When we assume that every execution latency is 1 cycle, it is easily observed the instruction path that determines the execution cycle of these instruction flows consists of instructions {`I0`, `I2`, `I5`, `I6`, `I8`, `I9`}. This is the critical path.

In contrast, instructions `I1`, `I3`, `I4`, and `I7`, which we call non-critical instructions, do not determine the execution cycle, and thus their latency can be increased without affecting the execution cycle. This is the key observation of our proposal. If we can find non-critical instructions dynamically, they can be executed on slow functional units whose supply voltage is low. In summary, only critical instructions are dispatched into fast and power-hungry functional units, while non-critical ones are dispatched into slow and power-efficient units. Since it is expected that the execution cycle is not increased, energy consumption will be reduced.

In order to find non-critical instructions dynamically, we use a critical path prediction (CPP) buffer [21]. Originally, the CPP buffer was proposed for improving processor performance. In this paper, we exploit the critical path predictability for the sake of energy reduction. The CPP buffer is a table and resembles caches, as shown in Figure 2. It is indexed by the program counter with each entry having a saturating up-down counter. When an instruction is identified as critical, its corresponding counter is incremented. Otherwise, it is decremented. When the counter value exceeds a threshold value, the instruction is predicted as critical. The counter length and the values sizes are dependent upon implementations. When the instruction is predicted as critical, it is dispatched into a fast and power-hungry functional unit. Otherwise, it is executed on a slow and power-efficient unit.

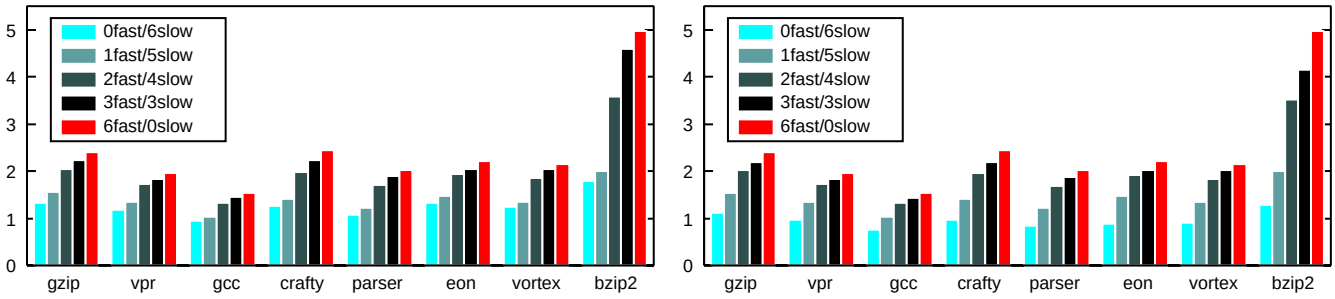
2.2 Methodology

We implemented a timing simulator using the SimpleScalar/Alpha tool set [2]. The baseline model is an out-of-order execution superscalar processor based on the register update unit [17], and its configuration is summarized in Table 1. The fast functional units can execute most integer operations in one cycle as shown in Table 1, and the slow functional units execute operations in two cycles. We will evaluate pipelined and non-pipelined slow units. The former can maintain the same throughput as that of the fast units. The latter diminished the throughput by half. We assume that the increase in power due to extra latches in the pipelined units is a factor of 1.15 [4].

We use Tune's critical path predictor [21]. It has a direct-mapped table of 6b saturating up-down counters. When an instruction is identified as critical according to the QOLD heuristic [21], its associated

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	64-entry instruction queue, 64-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
CPP Buffer	6b up-down counter, +8 on critical, -1 on non-critical, threshold = 8



(i) pipelined (ii) non-pipelined

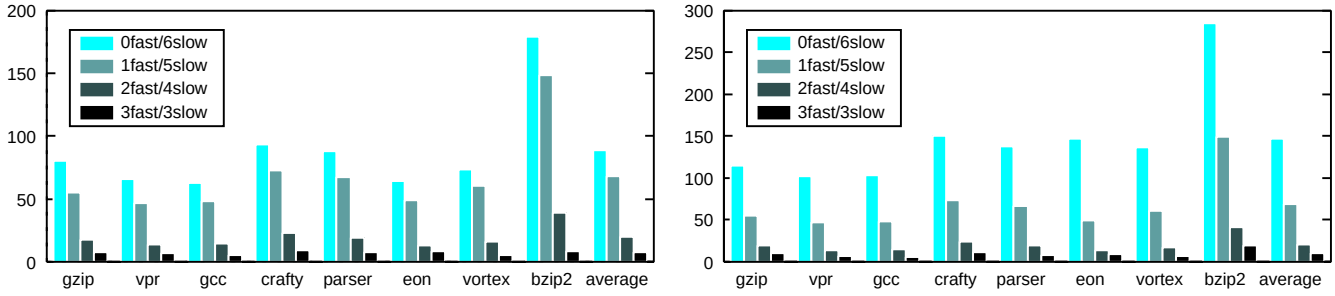
Figure 3: Instructions per cycle (64K)

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. These numbers follow Tune et al.'s study.

2.3 Results

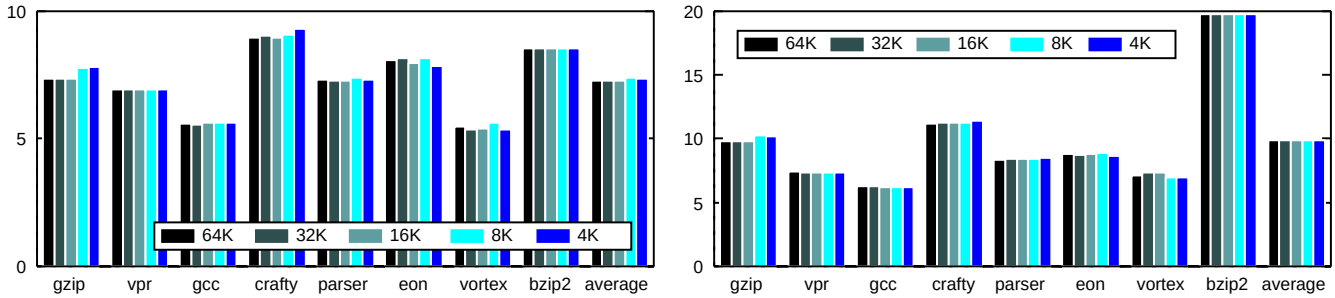
Figure 3 shows processor performance when a 64K entry CPP buffer is used. We use committed instructions per cycle (IPC) as a metric for evaluating performance. We evaluate five variations, each of which has various combinations of a total of 6 functional units. For each group of five bars, the first one is for the model that has 6 slow units and the last one is for the model that has 6 fast units. The figure shows the configurations of the remaining bars. As explained above, we use two types of slow units. One is the pipelined unit and the other is the non-pipelined one. Figure 3(i) shows the results when slow units are pipelined, and Figure 3(ii) shows the results when slow units are not pipelined. It is easily observed that processor performance is significantly diminished when all functional units are replaced by slow units. However, except in the case of `256.bzip2`, the configuration of `2fast/4slow` has comparable performance with the baseline configuration.

Because we are interested in energy efficiency, execution cycles are a more useful metric than IPC. Figure 4 shows the percent increase of execution cycles when the 64K entry CPP buffer is used. For example, in the case of `164.gzip`, the processor model `1fast/5slow` is 1.5 times slower than the baseline model. Figure 4 gives us a different impression from Figure 3. It is observed that 3 fast units are required in order to mitigate the performance degradation. In this case, processor performance is diminished by approximately 10% on average. Hence, we will use 3 fast functional units in the rest of this paper, except when



(i) pipelined (ii) non-pipelined

Figure 4: %Increase of execution cycles (64K)



(i) pipelined (ii) non-pipelined

Figure 6: %Increase of execution cycles (3fast/3slow)

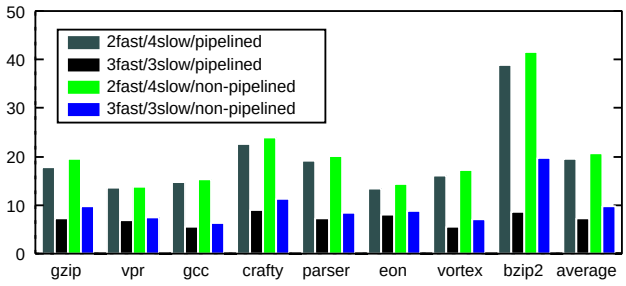
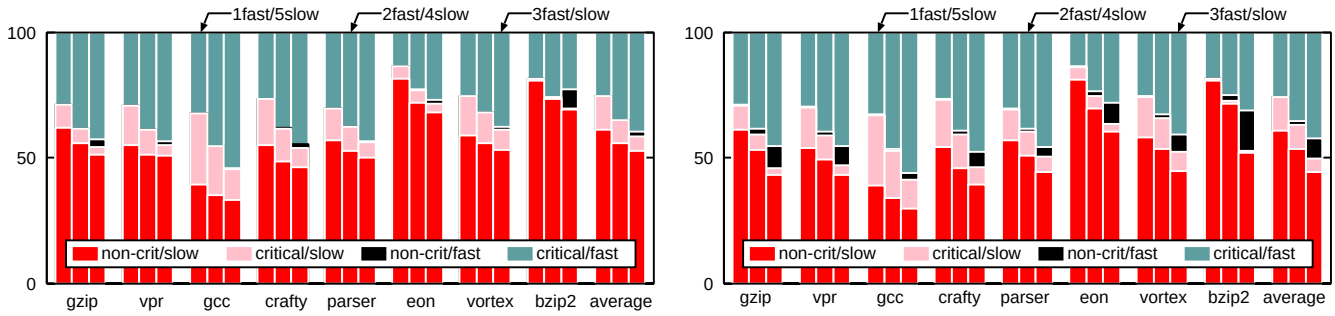


Figure 5: %Increase of execution cycles (64K)

specified otherwise. When we compare the pipelined and non-pipelined slow units, the latter seems to have serious impact on execution cycles. However, when 3 fast units are used, the difference is insignificant as shown in Figure 5.

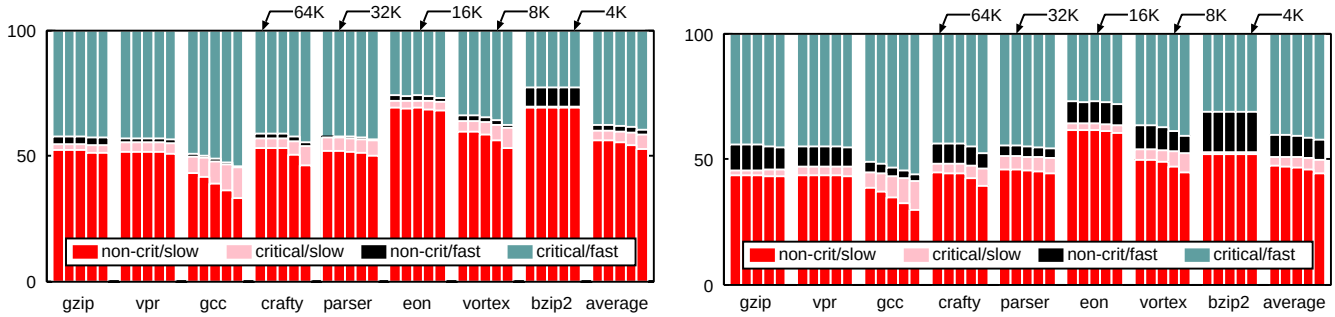
We now consider the influence of the number of the CPP buffer entries on execution cycles. Figure 6 shows the results when the number of the CPP buffer entry is reduced from 64K to 32K, 16K, 8K, and 4K. As can be easily observed, the influence is insignificant for the most cases. Thus, we will use a 4K entry CPP buffer in the rest of this paper, except when specified otherwise.

Next, we study how frequently the slow units are used. Figure 7 presents the percent distribution of dispatched functional units. We show 3 processor configurations, which are `1fast/5slow`, `2fast/4slow`, and `3/fast/3slow`. Each bar is divided into four parts. The bottom part indicates the percent of instructions that are predicted to be non-critical and are dispatched into the slow units (`NS`). The next indicates that of instructions that are predicted to be critical and are dispatched into the slow units (`CS`). The next bar is for the instructions that are predicted to be non-critical and are dispatched into the fast units (`NF`). The top bar indicates the percent of instructions that are predicted to be critical and are dispatched into the fast units (`CF`). As the sum of `NS` and `CS` is relatively larger, power consumption is reduced. However, execution cycles increase according to `CS`, resulting in a possible increase of energy consumption. On the other hand, large `NF` diminishes power efficiency. In



(i) pipelined (ii) non-pipelined

Figure 7: %Distribution of dispatched units (4K)



(i) pipelined (ii) non-pipelined

Figure 8: %Distribution of dispatched units (3fast/3slow)

summary, large `NS` and small `CS` and `NF` are expected.

First, as can be easily observed, as the number of the slow units increases, `CS` increases, resulting in performance degradation. This confirms the results shown in Figures 3 and 4. Second, when we use the non-pipelined slow units, `NF` becomes large. Because the non-pipelined units have low throughput, more units are required for non-critical instructions. And last, in the case of the `3fast/3slow` configuration, approximately 50% of instructions are dispatched into the slow functional units on average.

Figure 8 shows the influence of the number of the CPP buffer entries on the distribution, indicating that it is less significant than processor configuration.

Figure 9 presents the energy reduction of functional units. In this evaluation, we assume that both the clock frequency and the supply voltage for the slow units are half of those for the fast units. Therefore, the power consumption of the pipelined and non-pipelined slow units is $\frac{1}{4} * 1.15$ and $\frac{1}{8}$ of that of the fast units. Because execution cycles increase in general, energy reduction does not match power reduction. One of

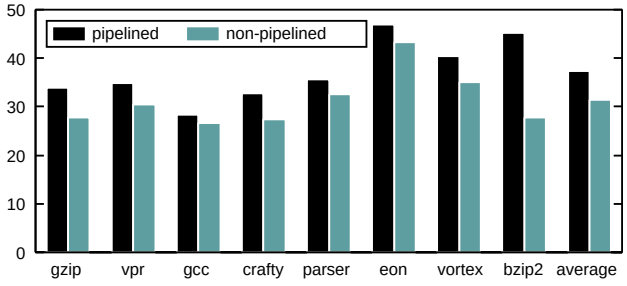


Figure 9: %Energy reduction of functional units (3fast/3slow/4K)

the interesting observations is that the energy efficiency of the pipelined units is better than that of the non-pipelined units, while the power reduction of the pipelined units is smaller than that of non-pipelined units. This is because performance degradation is insignificant due to the units' unchanged throughput. Anyway, the energy consumption of functional units

can be reduced by approximately 37% and 31% on average in the cases of the pipelined and non-pipelined units, respectively. Hence, it is confirmed criticality-based instruction scheduling based on critical path prediction is effective for energy reduction.

3 Contrail Processor Architecture

Because power efficiency is limited when we exploit only critical-path information, we aggressively create non-critical instructions in order to reduce the energy consumption. We call the structure of this technique Contrail processor architecture, a contrail being the trail of condensation left behind by a passing jet airplane. The reason for this analogy will be explained in the following explanations.

3.1 Architecture overview

In the Contrail, we divide an execution of an application into two streams. One is called the *speculation stream* and consists of the main part of the execution. However, it exploits trace-level value prediction [11,13], and thus several regions of the execution are skipped. In other words, the number of instructions in the speculation stream is smaller than that in the original execution, resulting in energy reduction. In contrast, the other stream is called the *verification stream* and supports the speculation stream by verifying each data prediction. The key idea is that the trace-level value prediction translates each critical path into a non-critical one and we move it from the speculation stream into the verification stream. Hence, the verification stream can execute slowly if the data prediction accuracy is considerably high. We can reduce the clock frequency of the datapath for the verification stream. Furthermore, the supply voltage is also reduced. From these considerations, its energy consumption is significantly reduced.

Each stream executes as a thread on a simultaneous multi-threading processor [8, 20] or a chip multiprocessor [10, 19], whose execution core consists of dual speed pipelines. The speculation stream is dispatched into a high-speed pipeline (speculation pipeline) and the verification stream is dispatched into a low-speed and low-supply-voltage pipeline (verification pipeline). In the ideal case, that means there are no mispredictions; the speculation stream finishes silently and waits for the verification process. In the case in which a misspredict occurs, the execution of the speculation stream is squashed at the point where the misspredict is detected and processor state is recovered by the verification stream.

This technique is called Contrail architecture, because the speculation stream runs ahead just like a jet plane, and the verification stream is left behind by the speculation stream and fades away in the manner of a contrail. One of the differences between this architecture and previously proposed pre-computing architectures [12, 18] is that the contrail processor architecture does not rely on redundant execution. In the ideal case, the number of executed instructions is unchanged. Another difference is that its target is the improvement of energy efficiency instead of the improvement of performance.

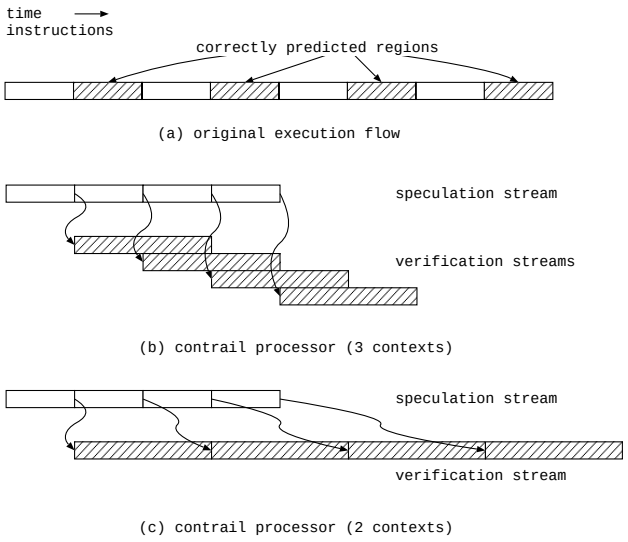


Figure 10: Execution on a contrail processor

3.2 Energy savings

The potential effect of the contrail processor architecture on energy-efficiency is estimated as follows: We determine the clock frequency and supply voltage for the verification pipeline at half those for the speculation pipeline. We assume that half of the original execution of an application is ideally predicted and is distributed uniformly as explained in Figure 10(a). This is a reasonable assumption, since it has been reported that 59% of dynamic traces can be reused with the help of the value prediction [11]. Under this assumption, the execution is divided into speculation and verification streams in a contrail processor with three contexts (1 and 2 for the speculation and verification streams, respectively) as depicted in Figure 10(b). The predicted regions are skipped in the speculation stream and execute in the verification streams while enlarging their execution time. Energy consumption is calculated as follows: For the speculation stream, energy consumption becomes half that of the original execution since the number of instructions is reduced by half. In contrast, for the verification

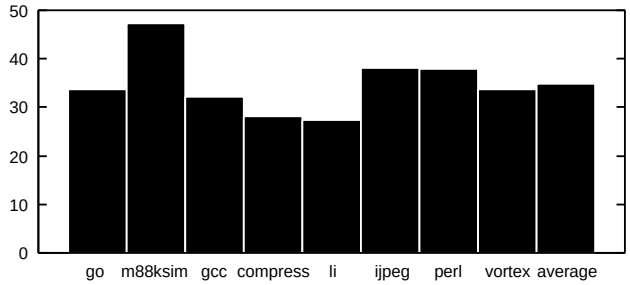


Figure 11: (%) Energy reduction on Contrail

streams, the sum of every execution time remains unchanged since the execution time of each instruction increases doubly while the total number of instructions is reduced by half. Its energy consumption is decreased by the reduction of the clock frequency and the supply voltage. Based on Eq.(1), it is reduced to $\frac{1}{8}$. Thus, the total energy savings is 37.5%. In addition, the application achieves higher performance. On the other hand, when the contrail processor has only two contexts, the execution time becomes slightly longer as explained in Figure 10(c). In this model, every thread constructing the verification stream is kept in a FIFO queue when it can not obtain its dedicated context. It is true that the effectiveness of contrail processors (energy savings) depends on the value prediction accuracy and the size of each predicted region. However, we have confirmed that the potential effect of contrail processors on energy savings is substantial.

The preliminary simulation results using a functional simulator derived from SimpleScalar/PISA tool set is shown in Figure 11. In this simulation, we assume that correctly predicted instructions ideally contribute to power reduction, and we ignore penalties caused by misspredictions and power overhead caused by the value predictor. It is observed that an approximately 35% energy reduction on average is achieved.

4 Conclusions

Power consumption is becoming one of the most important constraints for microprocessor design in nanometer-scale technologies. Device engineers, circuit designers, and system architects are faced with many design challenges regarding general-purpose processors as well as embedded processors. In order to manage the impact of increasing microprocessor power consumption, we proposed criticality-based instruction scheduling and Contrail processor architecture, which utilize criticality of instructions. We exploited dynamic information regarding instruction criticality based on critical path prediction and attacked selectively non-critical instructions for low power. In other words, only critical instructions are executed on the fast and power-hungry functional units, and non-critical instructions are executed on the slow and power-efficient units. In the Contrail, the trace-level value prediction translates each critical path into a non-critical one and moves it from the speculation stream into the verification stream, and the non-critical instructions are then executed in the slow units. Simulation results demonstrate we have found that both the criticality-based instruction scheduling and the Contrail architecture are effective for energy reduction.

We have recently finished evaluating correlation-based global critical path predictors. We will present the results on other articles in the near future. A future study regarding Contrail involves a detailed evaluation of trace-level value prediction and its energy efficiency. In addition, any type of architectural-level power estimation tool is required, because estimating power consumption in early design stage is required in order to examine tradeoffs between power and performance in architectural-level design. We have already built a power-performance simulation infrastructure for embedded processors [14, 15]. Its modeling resembles those proposed in [1, 3, 23], and thus can be applied to general-purpose microprocessors. We will try to build an architectural power and performance estimation tool.

Acknowledgments

This work is supported in part by the Grants-in-Aid for Encouragement of Young Scientists (No.12780273) and the Grants-in-Aid for Scientific Research (No.13558030) both from the Japan Society for the Promotion of Science, and a research grant from the Semiconductor Technology Academic Research Center.

References

[1] D.Brooks, V.Tiwari, M.Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," 27th International Symposium on Computer Architecture, June 2000.

[2] D.Burger, T.M.Austin, "The SimpleScalar Tool Set, Version 2.0," ACM SIGARCH Computer Architecture News, vol.25, no.3, June 1997.

[3] G. Cai, C. H. Lim, "Architectural Level Power/Performance Optimization and Dynamic Power Estimation," 32nd International Symposium on Microarchitecture, Cool Chips Tutorial, November 1999.

[4] A.P. Chandrakasan and R.W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," Proceedings of IEEE, vol.83, no.4, 1995.

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

[6] M.Hashimoto, H.Onodera, "Post-Layout Transistor Sizing for Power Reduction in Cell-Based Design," IEICE Transactions on Fundamentals, vol.E84-A, no.11, November 2001.

[7] T.Hiramoto, M.Takamiya, "Low Power and Low Voltage MOSFETs with Variable Threshold Voltage Controlled by Back-Bias", IEICE Transactions on Electronics, vol.E83-C, no.2, February 2000.

[8] Intel Corporation, "Introduction to Hyper-Threading technology", White paper, September 2001.

[9] M.Levy, "NEC Processor Goes Out of Order," Microprocessor Report, vol.15, archive 9, September 2001.

[10] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson, K. Chang, "The Case for a Single-Chip multiprocessor," 7th International Conference on Architectural Support for Programming Languages and Operating Systems, October 1996.

[11] M. L. Pilla, A. T. da Costa, F. M. G. Franca, P. O. A. Navaux, "Predicting Trace Inputs with Dynamic Trace Memoization: Determining Speedup Upper Bounds", 10th International Conference on Parallel Architectures and Compilation Techniques, WiP session, September 2001.

[12] A.Roth, G.Sohi, "Speculative Data-Driven Multithreading", 7th International Symposium on High-Performance Computer Architecture, January 2001.

[13] R.Sathe, K.Wang, M.Franklin, "Techniques for Performing Highly Accurate Data Value Prediction", Microprocessors and Microsystems, vol.22, no.6, November 1998.

[14] T.Sato, M.Nagamatsu, H.Tago, "Power and Performance Simulator:ESP and its Application for 100MIPS/W Class RISC Design," Symposium on Low Power Electronics, October 1994.

[15] T.Sato, Y.Ootaguro, M.Nagamatsu, H.Tago, "Evaluation of Architecture-level Power Estimation for CMOS RISC Processors," Symposium on Low Power Electronics, October 1995.

[16] T.Sato, I.Arita, "Contrail Processors for Converting High-Performance into Energy-Efficiency," 10th International Conference on Parallel Architectures and Compilation Techniques, WiP session, September 2001.

[17] G.S.Sohi, "Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers," IEEE Transactions on Computers, vol.39, no.3, March 1990.

[18] K. Sundaramoorthy, Z. Purser, E. Rotenberg, "Slipstream Processors: Improving Both Performance and Fault Tolerance" 9th International Conference on Architectural Support for Programming Languages and Operating Systems, November 2000.

[19] M.Tremblay, "An Architecture for the New Millenium," Hot Chips 11, August 1999.

[20] D.M.Tullsen, S.J.Eggers, H.M.Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism", 22nd International Symposium on Computer Architecture, June 1995.

[21] 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.

[22] K.Usami, M.Horowiz, "Clustered Voltage Scaling Technique for Low-Power Design," International Symposium on Low Power Design, April 1995.

[23] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, W. Ye, "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower," 27th International Symposium on Computer Architecture, June 2000.

[24] L.Wet, Z.Chen, M.Johnson, K.Roy, "Design and Optimization of Low Voltage and High Performance Dual Threshold CMOS Circuits," International Design Automation Conference, June 1998.