Energy Reduction via Critical Path Prediction

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

[1] Department of Artificial Intelligence, Kyushu Institute of Technology
[2] Center for Microelectronic Systems, Kyushu Institute of Technology
{tsato,chiyo,arita}@mickey.ai.kyutech.ac.jp

Abstract

Power consumption is a major concern in embedded microprocessors design. Reducing power has also been a critical design goal for general-purpose microprocessors. Since they require high performance as well as low power, power reduction at the cost of performance cannot be accepted. There are a lot of device-level techniques that reduce power with maintaining performance. They select non-critical paths as candidates for low-power design, and performance-oriented design is used only in speed-critical paths. The same philosophy can be applied to architectural-level design. We evaluate a technique, which exploits dynamic information regarding instruction criticality in order to reduce power. Regarding the effect of sustaining throughput on power and performance, it is found that pipelined functional units are better in energy reduction as well as in performance than non-pipelined units even if the increase in hardware due to extra latches are considered. We also evaluate an instruction steering policy for a clustered microarchitecture, which is based on instruction criticality, and find it is currently not a good design choice for energy efficiency.

Keywords: low power architecture, energy reduction, dual-voltage pipeline, critical path prediction, superscalar processors

1 Introduction

Power consumption is becoming one of the most important constraints for microprocessors design in nanometer-scale technologies. Device engineers, circuit designers, and system architects are faced with many challenges. In the area of embedded microprocessors, power has already been a major design constraint. However, it is also a limiting factor in general-purpose microprocessors, because they are used in mobile and embedded computer platforms. For example, microprocessors that support out-of-order execution are designed for modern embedded systems [11]. In order to manage the impact of increasing microprocessor power consumption, some architectural-level techniques are required as well as device-level design improvements.

Design tradeoffs can be achieved using many device-level techniques such as transistor size optimizations [8], multiple supply voltages [19], and multiple threshold voltages [21] 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. Seng et al. proposed to exploit dynamic information regarding instruction criticality in order to reduce power [15]. We have also studied this technique independently in parallel [5]. It should be noted that targets of the technique are different; Seng et al. aimed to improve performance/power density ratio for attacking hot spots. On the other hand, we are interested in energy, which is important in mobile devices.

The remainder of this paper is organized as follows. Section 2 discusses related studies. Section 3 explains how energy consumption is reduced by critical path prediction. Section 4 describes our evaluation methodology. Section 5 discusses our simulation results. Finally, Section 6 provides our conclusions.

2 Previous 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. [10] uses instruction criticality for instruction steering policy and proposed the path info table, which keeps dynamically-generated data flow graph. They calculate the length of each critical path based solely on the data dependence graph. Critical path prediction [6,18] 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 [3,13,15]. Casmira et al. [3] studied the potential benefit of exploiting slack, which is a metric that tells how critical an instruction is, for power reduction. They found there is significant availability of non-critical instructions, but did not mention any practical mechanism that can utilize their results for power reduction. Pyreddy et al. [13] use profile-based heuristics proposed by Tune et al. [18] 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. [15] 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. In addition, they assume the slow units to be pipelined.

We proposed Contrail processor architecture for energy reduction [14]. In Contrail, an execution of an application is divided into two streams. One is called the speculation stream. It consists of the main part of the execution and is dispatched into the fast functional units. However, several regions of the execution are skipped by utilizing trace-level value prediction. The other stream is called the verification stream. It supports the speculation stream by verifying each data prediction, and is dispatched into the slow units. The key idea is that the trace-level value prediction translates each critical path into non-critical one and moves it from the speculation stream into the verification stream, and then the non-critical instructions are executed on the slow units.

3 Energy Reduction via Critical Path Prediction

We exploits dynamic information regarding instruction criticality in order to reduce power. This is based on critical path prediction [6,18]. In this section, we describe how to reduce power using the critical path prediction.

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 that 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.

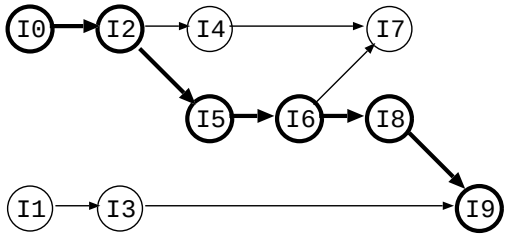


Figure 1. Data flow graph

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 this technique. 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 the slow and power-efficient units. Since it is expected that its execution cycle is not increased, energy consumption will be reduced. In the remainder of this paper, we assume the slow units is 2 times slower than the fast unit. Note that the slow units can be implemented either with and without pipelining technique.

In order to find non-critical instructions dynamically, we use a critical path prediction (CPP) buffer [18]. 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 value sizes are dependent upon implementations. When the instruction is pre-

dicted as critical, it is dispatched into the fast functional unit. Otherwise, it is executed on the slow unit.

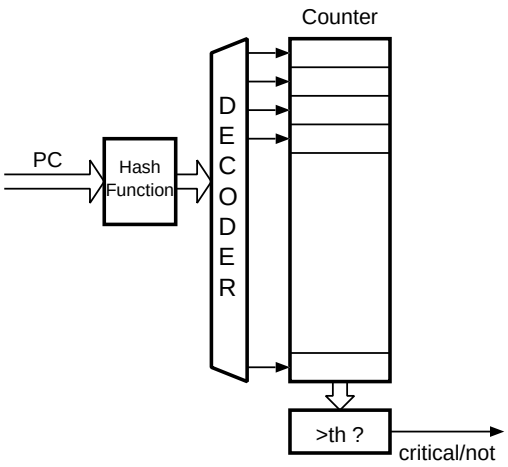


Figure 2. Critical path prediction buffer

In order to further reduce power consumption, we split the instruction queue into a fast queue and a slow one, as shown in Figure 3. In this clustered datapath, the fast cluster consists of the fast instruction queue and the fast functional units. Similarly, the slow cluster consists of the slow queue and the slow units. In other words, each cluster belongs to the different power-supply (and clock) domain. In order to connect different power-supply domains, level converters are generally used. In the case where any synchronizing mechanism is required between different clock domains, that is the case where the slow cluster is not pipelined, we assume that two clusters are connected by using small FIFOs. Since instruction queue is one of the most power-hungry block [7], reducing supply voltage and clock frequency in the slow queue is very effective. The splitting of instruction queue has another advantage of design complexity reduction [12].

In this clustered microarchitecture, we should consider an additional delay when inter-cluster bypassing occurs. In this study, we assume that the bypass delay is 1 cycle, regardless of the existence of FIFOs between clusters. In addition to the bypass delay, there is another delay when the instruction queue is not pipelined. In this case, the inter-cluster delay consists of the bypass delay and synchronizing delay, as shown in Figure 4. The upper part of Figure 4 explains the bypassing from the fast cluster to slow one, and the lower part explains the reverse. In the case at which the fast cluster finishes an execution at the beginning of the slow clock period, the synchronizing delay is required when the result is used in the slow cluster. Due to the additional delay, the clustered microarchitecture might

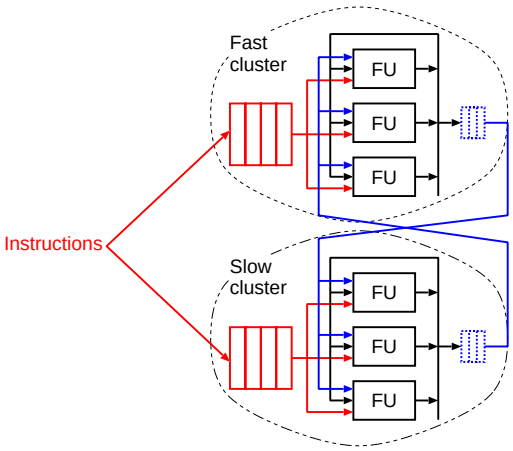


Figure 3. Clustered datapath

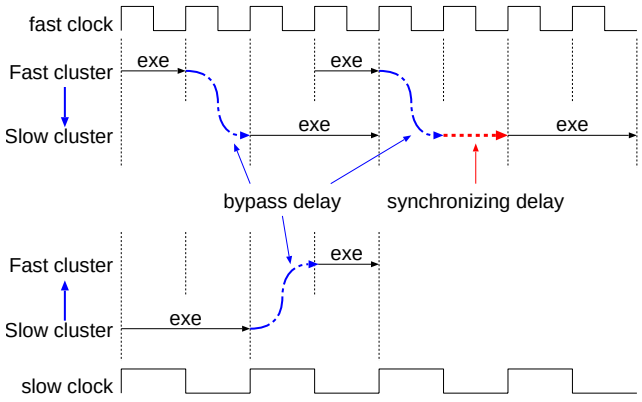


Figure 4. Inter-cluster bypassing

suffer performance loss.

4 Methodology

We implemented a timing simulator using the SimpleScalar/Alpha tool set (ver.3.0a) [2]. The baseline model is an out-of-order execution superscalar processor based on the register update unit (RUU) [16], 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. 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. Instructions can be dispatched into both fast and slow functional units. That is, if there are no free fast (slow) units, a critical (non-critical) instruction can be dispatched

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

Table 2. Supply voltage and frequency scaling

	clock frequency / supply voltage
TM5800	400MHz/0.93V — 800MHz/1.3V
XScale	500MHz/1.15V — 1.0GHz/1.6V

into the slow (fast) units. Note that this flexibility is lost when we split the instruction queue into the fast and slow queues, and that issuing instruction into the queues stalls when the either of the queues is full. We will evaluate pipelined and non-pipelined slow units. The former can sustain 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]. Note, in the split instruction queue model, each instruction dispatched into a functional units do not release its corresponding entry in the queue but remains there until it is committed.

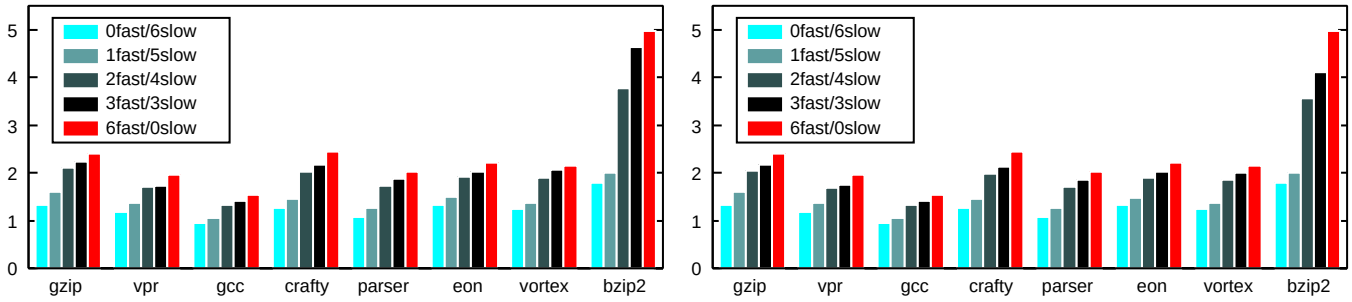
We will evaluate two scalings for supply voltage and clock frequency; one is based on Transmeta TM5800 [17], and the other is based on Intel XScale [9]. These scalings are summarized in Table 2. The higher frequency and voltage are applied to the fast functional units and instruction queue, and the lower ones are applied to the slow units and queue. Note that clock frequency and supply voltage are not dynamically adaptable but are unchanged. The energy consumed in the functional units is calculated as the product of its active power and execution time. Because power is proportional with clock frequency and quadratically with supply voltage, we can estimate the energy from the active rate of each unit and the execution cycles, using the supply voltage and frequency scaling shown in Table 2. While using Wattch [1] or SimplePower [20] will give us more detailed results, it is remained as a future study.

Table 3. Benchmark programs

program	input set
164.gzip	input.compressed
175.vpr	net.in arch.in
176.gcc	cccp.i
186.crafty	crafty.in
197.parser	test.in
252.eon	chair
255.vortex	lendian.raw
256.bzip2	input.random

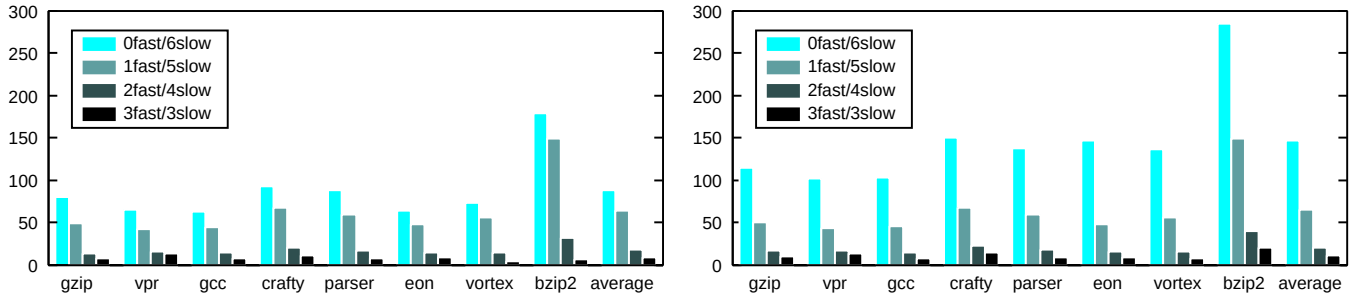
We choose to use the critical path predictor proposed in [18] due to its simplicity. Since we are interested in low-power architectures in this study, complex predictors can not be accepted. It has a direct-mapped table of 3-bit saturating up-down counters. When an instruction is identified as critical according to the QOLD heuristic [18], its associated counter is incremented by 1. Otherwise, it is decremented by 1. The QOLD heuristic marks the oldest instruction in instruction queue as critical. Because we use the RUU, the oldest instruction should be selected among those that are not dispatched into functional units. In other words, instructions that finish but are not committed are removed from the selection. The counter threshold at which an instruction is predicted as critical is set to 5. These numbers follow Seng et al.'s study [15].

The SPEC2000 CINT benchmark suite is used for this study. Table 3 lists the benchmarks and the input sets. We use the object files provided by the University



(i) pipelined (ii) non-pipelined

Figure 5. Instructions per cycle (64K)



(i) pipelined (ii) non-pipelined

Figure 6. %Increase in execution cycles (64K)

of Michigan. They were compiled by DEC C V5.9-008 on Digital UNIX V4.0 (Rev.1229). For each program except for `252.eon`, 1 billion instructions are skipped before the actual simulation begins. Each program is executed to completion or for 100 million instructions. We do not count nop instructions.

5 Results

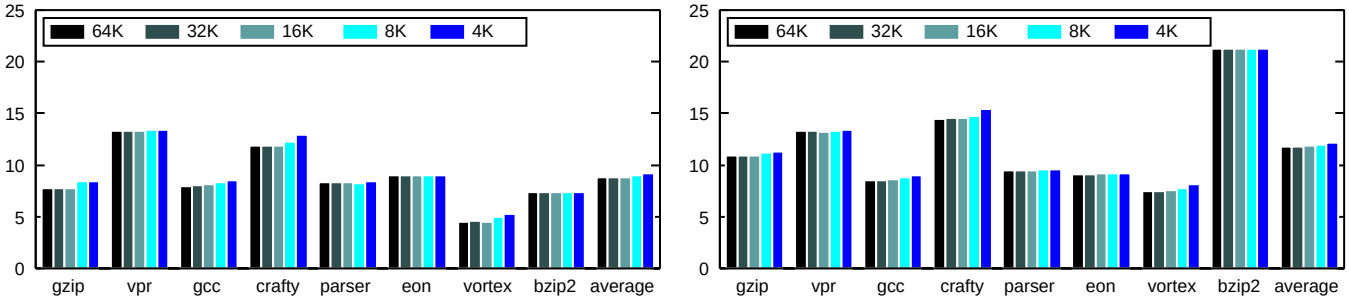
In this section, simulation results are presented. First, the criticality-based instruction scheduling is investigated. And then, the splitting of the instruction queue is evaluated.

5.1 Energy reduction via criticality-based scheduling

Figure 5 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. Note that the baseline model is `6fast/0slow`. As explained above, we use two types of the slow units. One is the pipelined unit and the other is the non-pipelined one. Figure 5(i) shows the results when the slow units are pipelined, and Figure 5(ii) shows the results when the slow units are not pipelined.

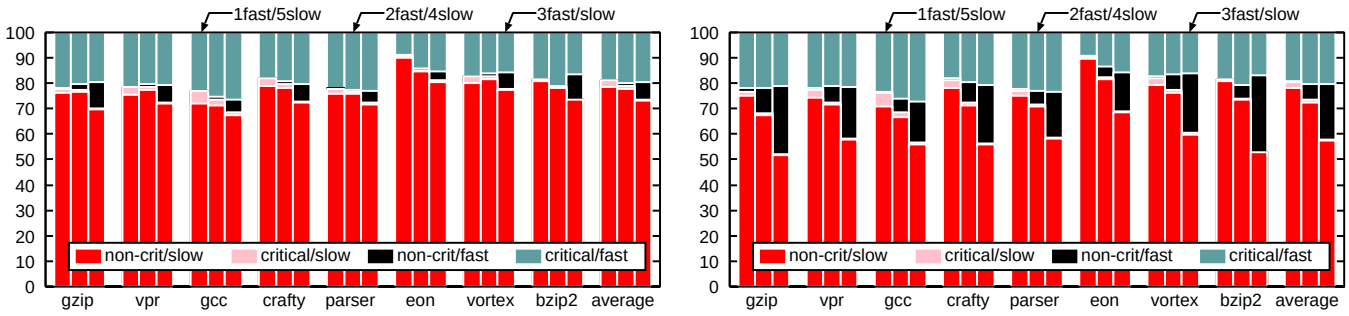
First, it is easily observed that processor performance is significantly diminished when all functional units are replaced by the slow units. However, except in the case of `256.bzip2`, the configuration of `2fast/4slow` has comparable performance with the baseline configuration. From these observations, we confirm that the critical path prediction mitigates performance loss due to lower supply voltage. Now, let's look at the results for the model using the non-pipelined units. We can see no significant differences between Figures 5(i) and (ii), and thus we expect that the non-pipelined units are more desirable than the pipelined units from the point of their lower power consumption. However, this will be denied below.

Because we are interested in energy efficiency, execution cycles are a more useful metric than IPC. Figure



(i) pipelined (ii) non-pipelined

Figure 8. %Increase in execution cycles (3fast/3slow)



(i) pipelined (ii) non-pipelined

Figure 9. %Distribution of dispatched units (4K)

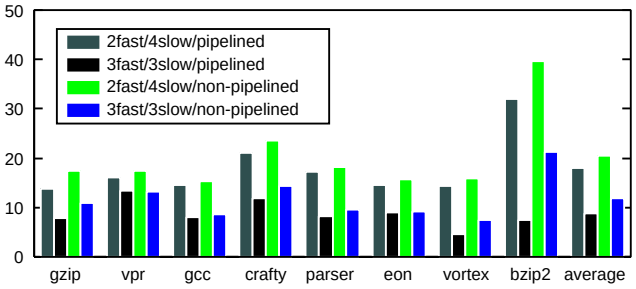
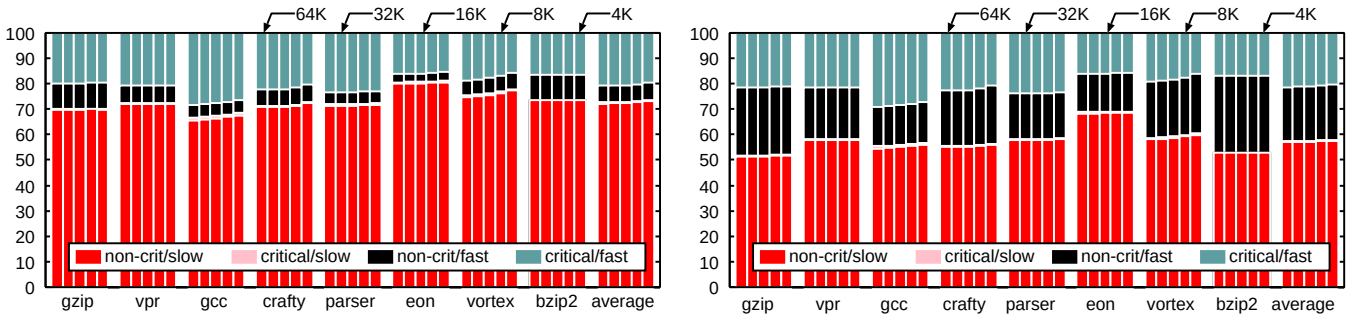


Figure 7. %Increase in execution cycles (64K)

Table 4. IPC (64K/0fast/6slow)

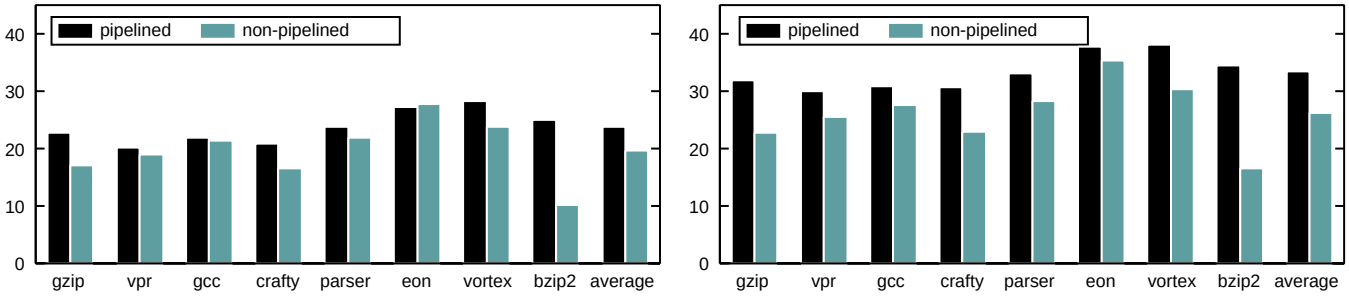
program	pipelined	non-pipelined	baseline
164.gzip	1.33	1.12	2.45
175.vpr	1.19	0.97	1.97
176.gcc	0.94	0.75	1.53
186.crafty	1.26	0.97	2.44
197.parser	1.08	0.85	2.02
252.eon	1.34	0.89	2.20
255.vortex	1.24	0.91	2.16
256.bzip2	1.78	1.29	4.98

6 shows the percent increase in 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 6 gives us a different impression from Figure 5, in which the difference between IPCs for each configuration disappears due to low resolution of the figure. To make this observation clear, we show the IPCs of the configuration `0fast/6slow` in Table 4 with that of the baseline (`6fast/0slow`). We can see considerable difference between each pair of IPCs. From Figure 6, it is observed that 3 fast units are required in order to keep the performance degradation, which is the increase in execution cycles, less than approximately 10% on average. Hence, we will use 3 fast functional units in the rest of this paper, except when specified otherwise. Another interesting observation is there are considerable differences between the results for the pipelined and non-pipelined models. The latter model suffers se-



(i) pipelined (ii) non-pipelined

Figure 10. %Distribution of dispatched units (3fast/3slow)



(i) 400MHz/0.93V — 800MHz/1.3V (ii) 500MHz/1.15V — 1.0GHz/1.6V

Figure 11. %Energy reduction in functional units (3fast/3slow/4K)

rious impact on execution cycles. However, when 3 fast units are used, the difference is insignificant as shown in Figure 7. In summary, we have found that the configuration of `3fast/3slow` is in a good tradeoff point when we consider the increase in execution cycles.

We now consider the influence of the number of the CPP buffer entries on execution cycles. Figure 8 shows the results, when the number of the CPP buffer entry is reduced from 64K to 32K, 16K, 8K, and 4K. The functional units configuration is `3fast/3slow`. 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 9 presents the percent distribution of dispatched functional units. We show three processor configurations, which are `1fast/5slow`, `2fast/4slow`, and `3fast/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 part is for the instructions that are predicted to be non-critical and are dispatched into the fast units (`NF`). The top 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 in energy consumption. On the other hand, large `NF` diminishes power efficiency. In summary, large `NS` and small `CS` and `NF` are expected.

First, it is easily observed that, as the number of the slow units increases, `CS` increases, resulting in performance degradation. This confirms the results shown in Figures 5 and 6. 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 70% and 60% of instructions are dispatched into the slow functional units on average in the cases of the pipelined and non-pipelined units models, respectively.

Figure 10 shows the influence of the number of the CPP buffer entries on the distribution, indicating that it is less significant than that of processor configuration.

Figure 11 presents the energy reduction in functional units. One of 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 the increase in execution cycles is insignificant due to the pipelined units' sustained throughput.

Figure 12 shows the relationship between supply voltage ratio and the average energy reduction. The supply voltage ratio is defined as the supply voltage for the slow units over that for the fast units. Thus, the smaller the ratio value is, the larger the expected energy reduction is. Note that every slow unit works at the same supply voltage. For example, the ratio is $\frac{1.15}{1.6} * 100 = 71.9\%$ for XScale. As can be easily observed, the pipelined units is better in energy reduction when the ratio is less than approximately 80%. Therefore, it is confirmed that the pipelined units is better both in performance and in energy reduction than the non-pipelined units.

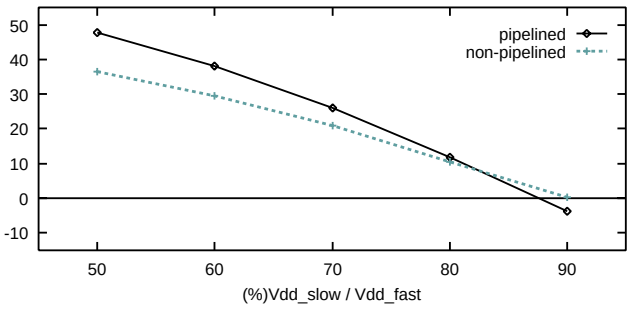


Figure 12. %Energy reduction vs. supply voltage ratio (3fast/3slow/4K)

5.2 Energy reduction via clustered microarchitecture

The power reduction in the instruction queue, which is explained in Section 3, is effective because it is one of the most power-hungry block [7]. This section evaluates two configurations for the queue; one consists of 32-entry fast queue and 32-entry slow queue (denoted as `32fastQ/32slowQ`). The other consists of 16-entry fast queue and 48-entry slow queue (denoted as `16fastQ/48slowQ`). The first configuration is chosen because both clusters have three functional units each. The second configuration is chosen because we have found that over 70% of instructions are dispatched into the slow units. Under the assumption of the voltage/frequency scaling in XScale, 30.4% and 36.3% power reduction in the queue can be attained for the `16fastQ/48slowQ` configuration with the pipelined and non-pipelined units models, respectively.

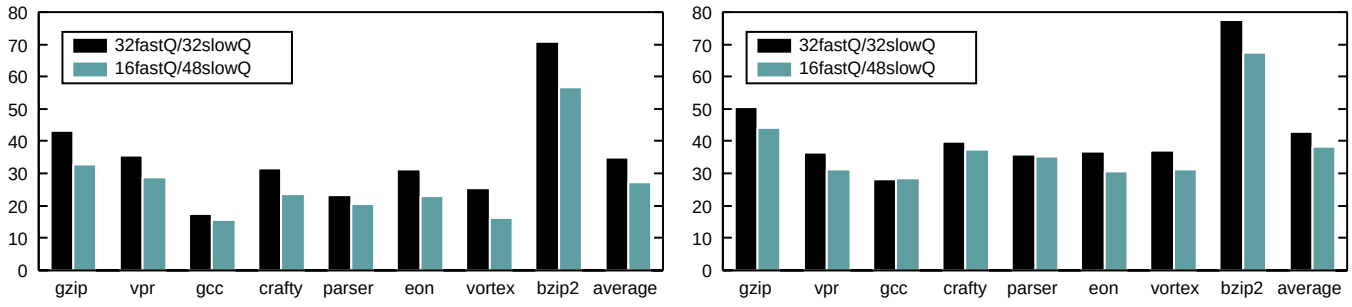
Table 5. %Inter-cluster dependence ratio

program	pipelined		non-pipelined	
	32f/32s	16f/48s	32f/32s	16f/48s
164.gzip	24.7	21.9	27.2	21.7
175.vpr	21.9	20.1	22.3	21.3
176.gcc	20.5	18.9	20.6	19.2
186.crafty	22.2	19.6	22.0	20.1
197.parser	17.4	15.1	16.8	14.6
252.eon	8.82	7.60	8.19	6.74
255.vortex	14.5	11.9	14.3	11.8
256.bzip2	26.9	25.6	22.9	22.5

Figure 13 shows the percent increase in execution cycles, when the splitting of the instruction queue is applied to the processor configuration of `3fast/3slow` with the 4K-entry CPP buffer. The significant performance loss is observed. The major reason of the increase in execution cycles is the decrease in instruction throughput due to the slow instruction queue. Compared with the centralized RUU used in the previous subsection, it can attain at most half instruction dispatch rate. The other reason is the extra delay due to inter-cluster bypassing. Table 5 presents how frequently inter-cluster bypassing occurs. Approximately 20% of bypassing are between two clusters. This is serious especially in the non-pipelined model. Another reason is the loss of flexibility in instruction issuing and dispatching. However, 11.7% energy reduction in the instruction queue is achieved, when we use the `16fastQ/48slowQ` configuration with the voltage/frequency scaling in XScale.

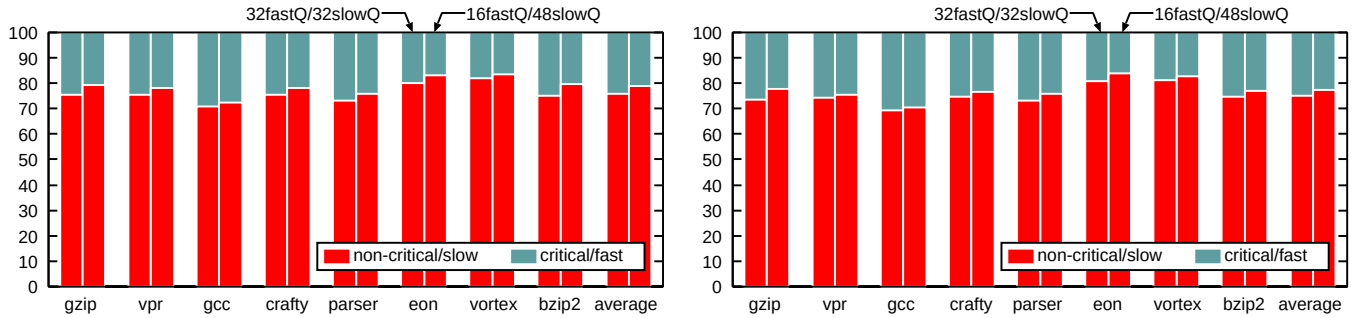
Figure 14 shows the distribution of dispatched unit. It is observed that approximately 80% of instructions are dispatched into the slow units, and thus are issued into the slow instruction queue. Therefore, it is confirmed that the `16fastQ/48slowQ` configuration is in a good tradeoff point.

Figure 15 shows an average of 6.0% energy reduction from the baseline model in functional units, when the `16fastQ/48slowQ` configuration is used with the voltage/frequency scaling in XScale. The decrease in energy efficiency is due to the significant increase in execution cycles. From these observations, currently the energy reduction by splitting the instruction queue is not always a good design choice, and any improvement in the technique is required. In order to utilize the



(i) pipelined (ii) non-pipelined

Figure 13. %Increase in execution cycles (clustered/3fast/3slow/4K)



(i) pipelined (ii) non-pipelined

Figure 14. %Distribution of dispatched units (clustered/3fast/3slow/4K)

technique in battery-operated devices, further study is required to mitigate the increase in execution cycles.

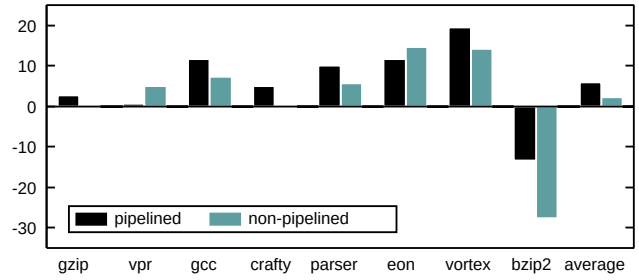


Figure 15. %Energy reduction in functional units (clustered/3fast/3slow/4K)

6 Conclusions

This paper investigated to exploit information regarding instruction criticality for improving energy efficiency in microprocessors. We evaluated to utilize critical path prediction for this purpose, and found that it is effective for power reduction. In addition, we evaluated the effect of sustaining throughput on power and performance. We found that the pipelined functional units are better in energy reduction as well as in performance than the non-pipelined units even if the increase in hardware due to extra latches are considered.

Our another study is on the splitting of the instruction queue. We found that currently the energy reduction via the splitting combined with critical path prediction is not always a good design choice. Energy efficiency is significantly diminished by the increase in execution cycles.

One direction of our future study is improving the accuracy of critical path prediction. If we predicted non-critical instructions more accurately, the performance loss would be decreased. Currently, we are studying low-cost and high-accuracy critical path predictors. We have recently finished evaluating correlation-based critical path predictors. They utilize global information in program execution as well as local information regarding to each instruction.

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.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 and T.M.Austin, "The SimpleScalar Tool Set, Version 2.0," ACM SIGARCH Computer Architecture News, vol.25, no.3, June 1997.

[3] J.Casmira, D.Grunwald, "Dynamic Instruction Scheduling Slack," Kool Chips Workshop, December 2000.

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

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

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

[7] D.Folegnani, A.Gonzalez, "Energy-Effective Issue Logic," 28th International Symposium on Computer Architecture, July 2001.

[8] 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.

[9] Intel Corporation, "Intel$^{(R)}$ XScale^{TM} Technology," `http://developer.intel.com/design/intelxscale/`, 2002.

[10] 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).

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

[12] S.Palacharla, N.Jouppi, J.Smith, "Complexity-Effective Superscalar Processors," 24th International Symposium on Computer Architecture, June 1997.

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

[14] 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.

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

[16] 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.

[17] Transmeta Corporation, "Crusoe^{TM} Processor Model TM5800," Product Brief, May 2001.

[18] 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.

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

[20] 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.

[21] 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.