
LOOK-AHEAD ARCHITECTURE ADAPTATION TO REDUCE PROCESSOR POWER CONSUMPTION

AN EFFECTIVE APPROACH TO REDUCING PROCESSOR POWER CONSUMPTION IS TO ADAPTIVELY ACTIVATE AND DEACTIVATE HARDWARE RESOURCES. THE AUTHORS PROPOSE A LOOK-AHEAD SCHEME THAT ADJUSTS THE PROCESSOR ISSUE RATE TRIGGERED BY MAIN-MEMORY ACCESSES. THIS ARCHITECTURE-INDEPENDENT TECHNIQUE IS PARTICULARLY EFFECTIVE FOR MEMORY-INTENSIVE APPLICATIONS. COMBINED WITH AN EXISTING TECHNIQUE BASED ON IPC VALUES, IT ALSO REDUCES POWER CONSUMPTION FOR COMPUTATION-INTENSIVE APPLICATIONS.

Zhichun Zhu
University of Illinois at
Chicago

Xiaodong Zhang
College of William and
Mary

..... As a negative by-product of the dedicated pursuit of high performance in general-purpose processors, the last decade has seen a dramatic increase in power consumption. To address this issue, researchers have aimed at reducing the processor's power dissipation with minimum effect on performance. One effective architecture-level approach, architecture adaptation, dynamically activates and deactivates hardware resources in accord with the changes in a running program's execution behavior.¹⁻⁵ The two key factors in architecture adaptation are when to trigger the adaptation and what adaptation techniques to apply.⁵ Our work focuses on the first issue.

Most existing architecture adaptation solu-

tions try to meet the current program requirement with a minimum number of active resources. However, these approaches share a common limitation: They trigger the adaptation after the processor has detected a change in demand. Without foreknowledge of future demand variants, the processor keeps resources active when current demand is high, even if demand is going to drop. Figure 1a shows an example of power-saving optimization based on current system status. The monitored IPC value is compared with a threshold at the end of each sampling window (the dots). At time A, the scheme puts the processor into normal execution mode on the basis of current knowledge that the measured value is higher than the

threshold. However, the scheme does not foresee that a slack exists between times B and D, when the processor is almost idle.

Ideally, hardware resources should be deactivated earlier than the demand drop point to maximize power saving without affecting performance. In Figure 1b, a look-ahead scheme delays part of the work until time C. However, because the same amount of work is finished before time D, overall performance remains the same while the processor stays in low-power mode for a longer time. In this article, we show that some hardware events can accurately predict future demand degradation, and thus hardware resources can be deactivated in advance even if current demand is high.

Specifically, we can use main-memory accesses to trigger architecture adaptation. Memory access latency has increased consistently relative to processor cycle time. Once a low-level cache load miss falls to main memory, the processor almost certainly will stall for this miss (although the processor can perform some useful work for subsequent instructions before stalling). For example, considering a moderate system configuration, a four-way-issue processor with a 128-entry instruction window runs at a 2 GHz clock rate, and its memory access latency is 100 ns. Once a load miss occurs, the load cannot be resolved within 200 cycles, and the instruction window will become full in 32 cycles at the full issue rate. Thus, after a load miss, maintaining the full processor issue width is wasteful even if current program demand is high.

We propose a new scheme called load indicator, in which main-memory accesses trigger the issue rate adaptation. In particular, the scheme reduces the issue width when a load miss occurs and resumes the full issue rate when all outstanding loads finish. Previous studies have shown that adjusting the processor issue rate is an effective adaptation technique for power saving.²⁵ Our experiments show that for

memory-intensive applications, this scheme saves more power than the pipeline-balancing technique,² which periodically adjusts the processor issue rate to the average issued IPC (instructions per cycle), with a comparable performance loss. For seven memory-intensive applications from the SPEC2000 benchmark suite, the load indicator scheme reduces power consumption in the issue logic and execution units by 24 and 11 percent, respectively. It reduces total processor power consumption by 5.4 percent on average, with a performance loss of 0.5 percent, reaching 69 percent of the maximum power saving that reducing the issue rate can achieve. In comparison, the pipeline-balancing technique reduces processor power by 4.2 percent on average, with a performance loss of 0.7 percent.

The load indicator scheme foresees the degradation on resource demands caused by memory accesses, whereas the pipeline-balancing scheme captures current program behavior variants caused by program-inherent parallelism and hardware constraints. We have combined these two indicators into a new technique, the load-instruction scheme, which uses information about both load misses and IPC variants (in the absence of load misses). This technique captures more power-saving opportunities and covers a wider range of applications. (We summarize other research

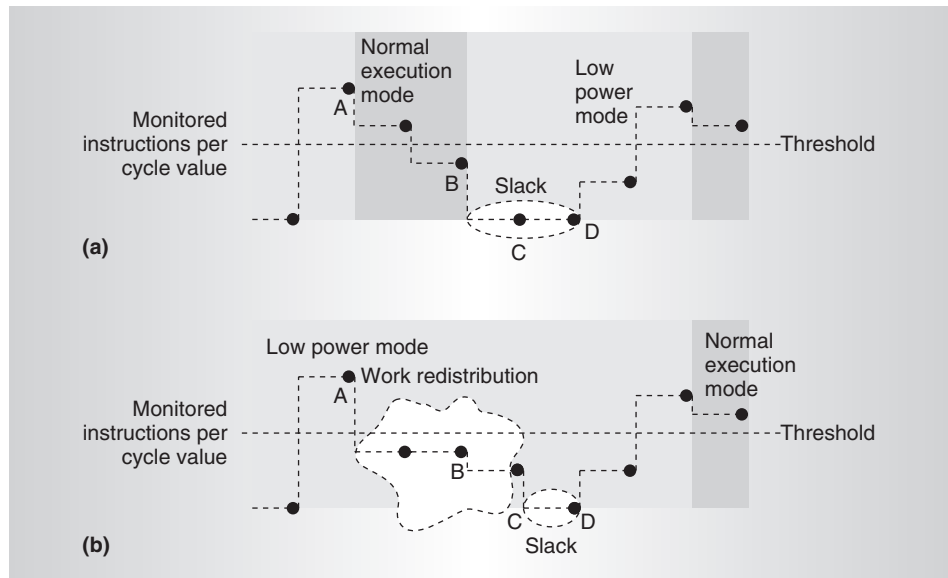


Figure 1. Comparison of optimization based on current system status (a) and look-ahead optimization (b).

Related work

An increasing number of researchers are investigating ways to reduce the power consumption of general-purpose processors. One technique, pipeline gating, reduces processor energy consumption by preventing wrong-path instructions from entering the pipelines.¹ Some researchers propose using the current rate of instructions passing through pipeline stages to throttle the processor front end²; others propose a combination of fetch gating and issue queue adaptation to reduce the power consumed by the front-end instruction delivery path.³ These techniques make continuous changes in processor resource utilization. In contrast, our method switches the processor between two or more power modes over relatively long periods. This allows us to more effectively shut down portions of the pipeline, including the clocking network, which we could not ordinarily fully shut down on a clock-by-clock basis, as making continuous changes would imply.

Issue logic is one of the most power-intensive system components and has been a research focus in recent years. Researchers have proposed several techniques to avoid unnecessary comparisons in the wake-up logic.⁴ Dynamically adjusting issue queue size, reorder buffer size, and load/store queue size is an effective approach to reducing issue logic's power consumption.^{4,6} In contrast, our method uses pipeline balancing, which saves power on both the issue logic and the execution units. This is also the goal of Bahar and Manne, but they don't consider load misses as a trigger.⁷

Other techniques make a processor run in low-power mode. One method switches processor execution between out-of-order and in-order modes under the guidance of an external performance indicator.⁸ The Pentium 4 processor explores the StopClock structure, which briefly halts the clock signal to a portion of processor logic.⁹ Another method targets energy saving in multimedia applications by using dynamic voltage scaling and architectural adaptation.¹⁰ Unlike these studies, our work focuses on reducing power consumption for memory-intensive applications.

References

1. S. Manne, A. Klauser, and D. Grunwald, "Pipeline Gating: Speculation Control for Energy Reduction," *Proc. 25th Ann. Int'l Symp. Computer Architecture* (ISCA 98), IEEE Press, 1998, pp. 132-141.
2. A. Baniasadi and A. Moshovos, "Instruction Flow-Based Front-End Throttling for Power-Aware High-Performance Processors," *Proc. 2001 Int'l Symp. Low Power Electronics and Design* (ISLPED 01), IEEE Press, 2001, pp. 16-21.
3. A. Buyuktosunoglu et al., "Energy Efficient Co-Adaptive Instruction Fetch and Issue," *Proc. 30th Ann. Int'l Symp. Computer Architecture* (ISCA 98), IEEE Press, 2003, pp. 147-156.
4. D. Folegnani and A. González, "Energy-Effective Issue Logic," *Proc. 28th Ann. Int'l Symp. Computer Architecture* (ISCA 01), IEEE Press, 2001, pp. 230-239.
5. A. Buyuktosunoglu et al., "An Adaptive Issue Queue for Reduced Power at High Performance," *Power-Aware Computer Systems*, LNCS vol. 2008, Springer Verlag, 2001, pp. 25-39.
6. D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing Power Requirements of Instruction Scheduling through Dynamic Allocation of Multiple Datapath Resources," *Proc. 34th Ann. Int'l Symp. Microarchitecture* (Micro 34), IEEE Press, 2001, pp. 90-101.
7. R.I. Bahar and S. Manne, "Power and Energy Reduction via Pipeline Balancing," *Proc. 28th Ann. Int'l Symp. Computer Architecture* (ISCA 01), IEEE Press, 2001, pp. 218-229.
8. S. Ghiasi, J. Casmira, and D. Grunwald, "Using IPC Variation in Workloads with Externally Specified Rates to Reduce Power Consumption," *Proc. Workshop Complexity-Effective Design, in conjunction with 27th Ann. Int'l Symp. Computer Architecture*, 2000; www.systems.cs.colorado.edu/Papers/Architecture/WCED2000-Adaptive/paper.pdf.
9. S.H. Gunther et al., "Managing the Impact of Increasing Microprocessor Power Consumption," *Intel Technology J.*, Q1, 2001, pp. 1-9.
10. R. Sasanka, C.J. Hughes, and S.V. Adve, "Joint Local and Global Hardware Adaptations for Energy," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 10), ACM Press, 2002, pp. 144-155.

on reducing power consumption in the "Related work" sidebar.)

Look-ahead schemes

Many low-power processor techniques switch the processor's execution between a normal mode and one or more power-saving modes. They normally exploit slack periods in program execution to utilize power-saving modes, making it critical to predict such slacks accurately and in a timely manner. Most of

these techniques use a slack period's initial phase as the indicator. For example, a processor in a notebook computer enters a lower-power mode after being idle for a certain time. This indicator is always available and reasonably reliable; thus, it is a good choice if the slack's duration is long enough. However, this technique doesn't exploit the power-saving opportunities in the initial phases.

Look-ahead techniques, on the other hand, use indicators whose changes can be detected

just before the start of a slack, maximizing the utilization of power-saving modes through the slack. These techniques are attractive, especially when slack periods are frequent but short, making the loss of power-saving opportunities in initial phases significant. A critical issue of any look-ahead technique, however, is to find a reliable slackness indicator.

There are ways to identify performance degradation in advance, such as by providing hints from the application, compiler, or operating system. For example, the cool-fetch technique uses compiler-driven static IPC prediction at the loop level to guide fetch throttling for energy saving.⁶ Another method, positional processor adaptation, associates the use of power-saving modes with program subroutines.⁷ Either static or dynamic instrumentation and decisions associate each subroutine with a power mode activated when the subroutine is executing.

A distinctive feature of our method is that it uses load miss information, a simple but accurate slack indicator. It does not require profiling, compiler analysis, or history-based hardware prediction. For superscalar processors, a load miss event is a reliable indicator of impending processor stalls. With the increasing gap between processor and memory speed, slacks due to memory accesses make up increasingly larger portions of programs' execution.

Load indicator scheme

Today's high-performance processors run at multi-GHz speeds and issue multiple instructions each cycle, whereas each main-memory access takes tens of nanoseconds. Once a cache load miss to main memory occurs, it is almost certain that the processor will stall because of long main-memory access time, even after applying latency-tolerant techniques such as nonblocking load/store and out-of-order execution. Thus, maintaining a partial processor issue width is enough to retain performance from the time the load miss occurs to the time the data returns.

Figure 2 shows the sampled IPC values of the SPEC2000 program *swim* and the number of outstanding cache load misses in a representative 1,024-cycle interval during the program's execution. For clarity, the figure presents only the floating-point IPC values. The integer IPC values show a similar pattern. We

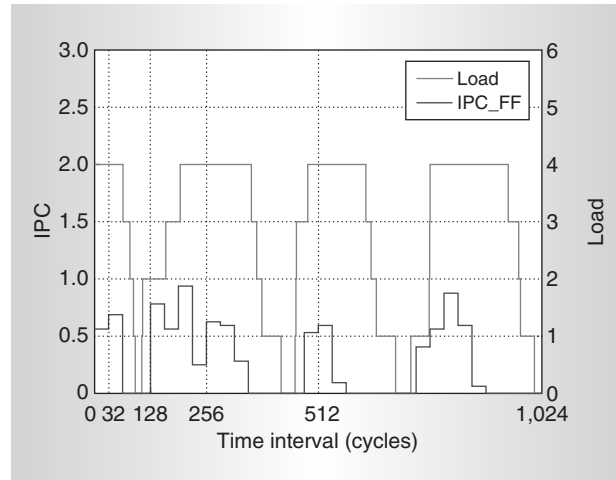


Figure 2. Sampled IPC values in 32-cycle windows and number of outstanding loads during an arbitrarily selected 1,024-cycle interval for program *swim*.

can see that for this application, multiple cache load misses normally occur together. This is also usually the case for many other memory-intensive applications.

From the time of the first cache load miss to the time all the load misses return, the program execution behavior forms a “frame”—an active period followed by an idle period. This means that load miss information can accurately predict future performance degradation.

In our load indicator scheme, when a load miss occurs, if the processor is in normal execution mode (that is, with the full issue rate), it shifts to low-power mode. (We consider only cache load misses because the write buffers usually handle write misses well.) In low-power mode, the processor issue rate and the number of active functional units are reduced by half. When all load misses return, the processor returns to normal execution. For simplicity, the scheme does not distinguish between true and speculative load misses. Our experiments show that this has little impact on power consumption and performance. The scheme does not consider load miss criticality and switches to the low-power mode on each load miss. This can cause some performance loss if the load miss is in the critical path. However, identifying the load's criticality would require complex logic and consume additional power. Thus, our scheme treats each load miss the same; and our experiments indicate that this simplification causes negligible performance

loss. Of course, if a mechanism identifying load criticality has already been implemented on a processor for performance improvement, the load indicator can use the information to put the processor into low-power mode only after noncritical load misses.

The load indicator identifies execution periods when a program does not require full computing power. Several power-saving techniques can be applied during those periods. For instance, if the processor has a dual-speed pipeline structure,^{8,9} instructions can be issued to the slow pipeline upon cache load misses. We use the technique of reducing the processor issue rate because this is an effective technique with low switching overhead and is applicable to most architectures. Li et al. propose scaling down the supply voltage of certain sections of the processor during an L2 miss.¹⁰ Because scaling voltage requires dozens of CPU cycles to stabilize the circuit, this technique necessitates additional effort to monitor the current degree of instruction-level parallelism and doesn't scale down the voltage when the ILP is above a certain threshold. This can cause the loss of power-saving opportunities.

The load indicator scheme's overhead is trivial. Only one register is added to record the number of outstanding loads. The control logic is also simple because it only checks the register value and then makes the adaptation. The scheme requires some additional logic to adjust the processor issue rate. However, compared with clock gating at each component on a cycle-by-cycle basis, the issue rate adjustment occurs at a coarser level. The additional power consumed by the scheme is negligible.

Load-instruction indicator

The existence of outstanding cache load misses is a strong indicator of future variants of program requirements on processor resources. However, this technique has significant effects only on memory-intensive applications. Bahar and Manne propose a technique called pipeline balancing, which dynamically adjusts processor issue width for each sampling window by comparing the average issued IPC measured in the previous window with a predetermined threshold.² We call the indicator to trigger the processor issue rate in this technique the *instruction indicator scheme* in the rest of this article.

The load indicator scheme simply uses the existence of load misses to adjust power modes, whereas the instruction indicator scheme needs predetermined thresholds for controlling issue width adjustments, and the optimal values of those thresholds depend on applications and platforms. The load indicator captures the future program variants caused by memory accesses, while the instruction indicator captures the current program variants caused by the program's inherent ILP and hardware constraints. We combine these two optimizations in the load-instruction indicator scheme.

When there are no outstanding load misses, the hardware monitors the issued IPC and adjusts the processor issue rate accordingly. When a cache load miss occurs and the processor is in normal execution mode, the processor switches to low-power mode. During cache load miss servicing, the hardware suspends its monitoring of the issued IPC. Monitoring resumes when the outstanding load misses finish. By combining local and look-ahead optimizations, the load-instruction indicator scheme switches the processor issue rate in a more effective and timely manner. The combined scheme's overhead is negligible because both the load indicator and the instruction indicator have trivial overhead.

Methodology

We used the Sim-Alpha simulator, which has been validated against a 466-MHz Alpha 21264 processor.¹¹ The processor can issue up to four integer instructions and two floating-point instructions every cycle. It contains 64-Kbyte, two-way instruction and data caches and a 2-Mbyte direct-mapped L2 cache. We scaled processor speed to 1 GHz and 2 GHz in our experiments. We scaled cache access latency, DRAM access latency, and bus bandwidth accordingly. We used the CACTI 3.0 model to estimate scaled cache access latency.¹² We assumed that DRAM access latency and bus bandwidth improve by 30 percent when processor speed doubles. Table 1 shows cache access latency and memory bus bandwidth as processor speed changes.

We modified the simulator to model our schemes and the instruction indicator technique. Each of the three schemes (load, load-instruction, and instruction indicators) is implemented independently. The processor

dynamically adjusts the processor issue rate when the conditions defined by the schemes are met. We used the precompiled Alpha version of SPEC CPU2000 benchmark programs as workloads and reference input data files as input. The simulator generated detailed statistics on 1 billion instructions after the first 4 billion instructions were fast-forwarded.

Reducing the processor issue rate reduces the power consumed by the issue logic and execution units.² The instruction issue units, integer execution units, and floating-point execution units consume about 18, 10, and 10 percent, respectively, of total processor power.¹³ We use these data in calculating power savings. We assume that as processor speed increases, the percentage of power consumed by the issue logic and the execution units stays the same. This conservative assumption doesn't favor our schemes. The Alpha 21264's pipeline structure consists of two integer clusters and one floating-point cluster. We also conservatively assume that reducing the processor issue rate saves power only on the integer clusters. One integer cluster is gated at the low-power mode.

Reducing the issue rate reduces only the power consumed by the clocks driving the execution units but not the execution units themselves (if we don't consider the small reduction in the number of wrong-path instructions obtained by lowering the issue rate). Assuming that the 32 percent of power consumed by clocks is evenly distributed for each component, halving the issue rate reduces the integer execution units' power consumption by $32\% \times 1/2 = 16\%$.

We usually don't know how many instructions can be issued in a given cycle until all choices are exhausted. Therefore, reducing the issue rate can directly save power in issue arbiters. For the issue logic, the arbiters, including the driving clocks, consume 70 percent of the power. Thus, reducing the issue

Table 1. Memory system parameters.

Processor speed	466 MHz	1 GHz	2 GHz
I cache latency	1 cycle	1 cycle	2 cycles
D cache latency	3 cycles	3 cycles	4 cycles
L2 cache latency	7 cycles	12 cycles	14 cycles
Bus bandwidth	1.2 GB/s	1.6 GB/s	2.1 GB/s

rate by half saves issue logic power by 35 percent. The total processor power reduction is $10\% \times 16\% + 18\% \times 35\% = 7.9\%$. This estimation method is the same as that used in the instruction indicator scheme.²

One might argue that reducing the processor issue rate cannot save much energy when aggressive clock-gating techniques are applied to each component cycle by cycle. However, reducing the processor issue rate works at a coarser grain than clock-gating techniques and decreases design complexity. In addition, it can save more power than clock gating because an entire resource or block can be powered off.^{2,14} Even with aggressive clock gating, it reduces the remaining power consumption.⁵

Results

We evaluated the load indicator scheme's effectiveness in memory-intensive applications. Then, we analyzed how various system configurations affect the scheme's effectiveness. Finally, we compared the load indicator, instruction indicator, and load-instruction indicator schemes.

Load indicator's effectiveness

From the SPEC2000 benchmark suite, we selected seven memory-intensive applications with memory stall portions greater than 20 percent under our experimental setup. (A program's memory stall portion is the percentage of time difference between running the program under a real system and under a system with an infinitely large L2 cache.) Table 2

Table 2. Execution time spent in low-power mode and power reduction in issue logic and execution units achieved by load indicator scheme.

SPEC2000 applications	<i>mcf</i>	<i>art</i>	<i>swim</i>	<i>lucas</i>	<i>applu</i>	<i>ampp</i>	<i>facerec</i>	Average
Time in low-power mode (%)	91.9	73.8	88.1	79.9	70.2	47.8	30.8	68.9
Power reduction in issue logic (%)	32.2	25.8	30.8	28.0	24.6	16.7	10.8	24.1
Power reduction in execution units (%)	14.7	11.8	14.1	12.8	11.2	7.6	4.9	11.0

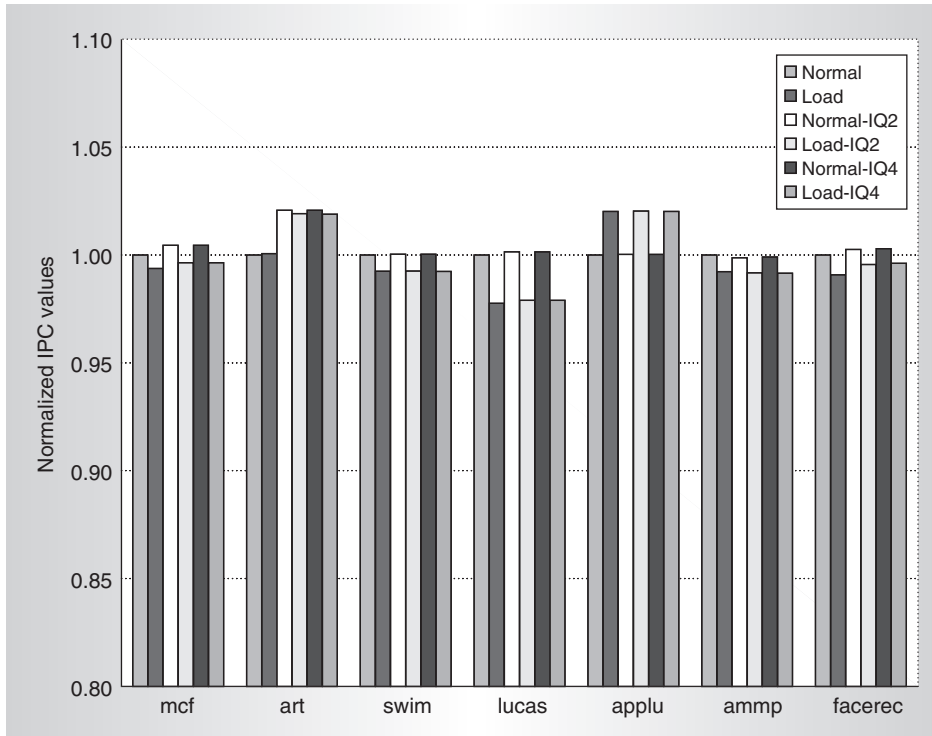


Figure 3. Variants of IPC values as issue queue (IQ) size increases. Both integer issue queue size and floating-point issue queue size double for “Normal-IQ2” and “Load-IQ2” and quadruple for “Normal-IQ4” and “Load-IQ4.”

shows the percentage of power reduction in the issue logic and execution units for the seven applications achieved by the load indicator scheme for systems with a 1-GHz processor. The load indicator scheme puts the processor in low-power execution mode at 30.8 to 91.9 percent of total execution time. On average, the processor spends 68.9 percent of the time in low-power mode. This translates to 24.1 percent ($35\% \times 68.9\%$) and 11.0 percent ($16\% \times 68.9\%$) average reductions in power consumed by the issue logic and execution units, respectively, compared with the normal execution scheme. Thus, the load indicator scheme is highly effective in capturing power-saving opportunities in memory-intensive applications.

Effect of system configurations

Figure 3 compares IPC values obtained in normal execution mode with those obtained by the load indicator scheme as integer and floating-point issue queue sizes increase. All IPC values are normalized to those of normal execution mode. The load indicator causes a

performance loss of up to 1.7 percent (*lucas*) when the integer issue queue has 20 entries and the floating-point issue queue has 15 entries. (Actually, the performance of program *applu* improved slightly because fewer misspeculative instructions were issued and executed.) The average performance loss is 0.54 percent.

As issue queue size doubles, a greater number of subsequent instructions can be held in the issue queue when a load miss occurs. Thus, reducing the issue rate might cause a greater performance loss, but because the issue queue is still not large enough to fully cover the memory access latency, the performance loss is negligible. When issue queue size doubles, the load indicator’s average performance loss increases slightly to 0.56 percent. When issue queue size

quadruples, average performance loss is only 0.59 percent. Significant performance loss might occur only when issue queue size is larger than the product of issue width and memory access latency in terms of processor cycles. However, with the increasing processor and memory speed gap, such large issue queues are unlikely to appear in future processors.

Figure 4 shows the percentage of reduction in total processor power consumption as processor speed scales. (Table 1 shows the memory system parameters.) As we expected, when processor speed scales from 466 MHz to 1 GHz and 2 GHz, the load indicator scheme’s average power reduction increases continuously from 5.1 percent to 5.4 percent and 5.8 percent. As processor speed increases, memory access latency in terms of processor cycles also increases. Thus, the load indicator has more opportunities to put the processor in low-power mode. Energy savings also increases, from 4.5 percent to 4.9 percent and 5.7 percent (not shown in the figure).

In summary, as the processor and memory speed gap enlarges, the load indicator scheme

becomes more effective in power and energy savings. With realistic issue queue sizes, doubling the issue queue entries causes almost no additional performance loss from the load indicator, and performance loss is still negligible. The load indicator's effectiveness is quite stable as the system configuration varies.

Comparing schemes

To compare the three different schemes' effectiveness on a wide range of applications, we used the seven memory-intensive applications mentioned earlier and added eight computation-intensive applications with small memory stall portions (3 to 14 percent). Figure 5 shows processor power reductions for the 15 programs. The instruction indicator scheme monitors each application's IPC values and then adjusts the processor's power mode every 64 cycles. When the processor runs at the full issue rate, if the issue IPC is lower than 1.1 and the floating-point issue IPC is lower than 0.4, the processor switches to low-power mode. If the processor is in low-power mode with an issue IPC higher than 1.2 and a floating-point issue IPC higher than 0.5, the processor switches back to normal execution mode. The parameters are system- and application-dependent and have already been tuned for these applications.

For the seven memory-intensive applications, the load indicator scheme saves more power than the instruction indicator. It reduces processor power consumption by 5.4 percent with a performance loss of 0.5 percent, compared with the instruction indica-

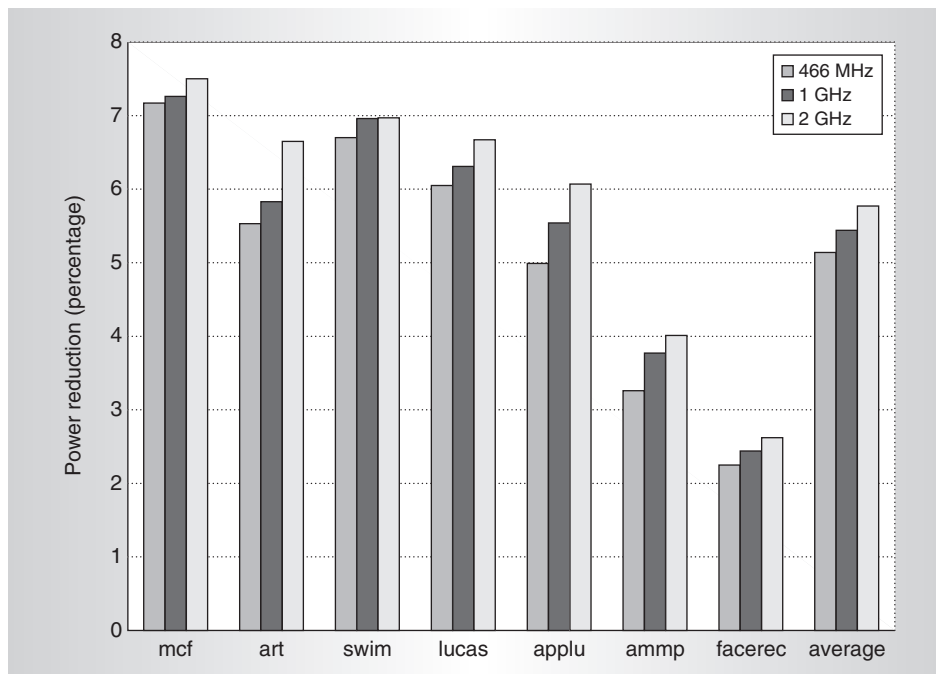


Figure 4. Load indicator's power reduction with processor speed scaling from 466 MHz to 1 GHz and 2 GHz.

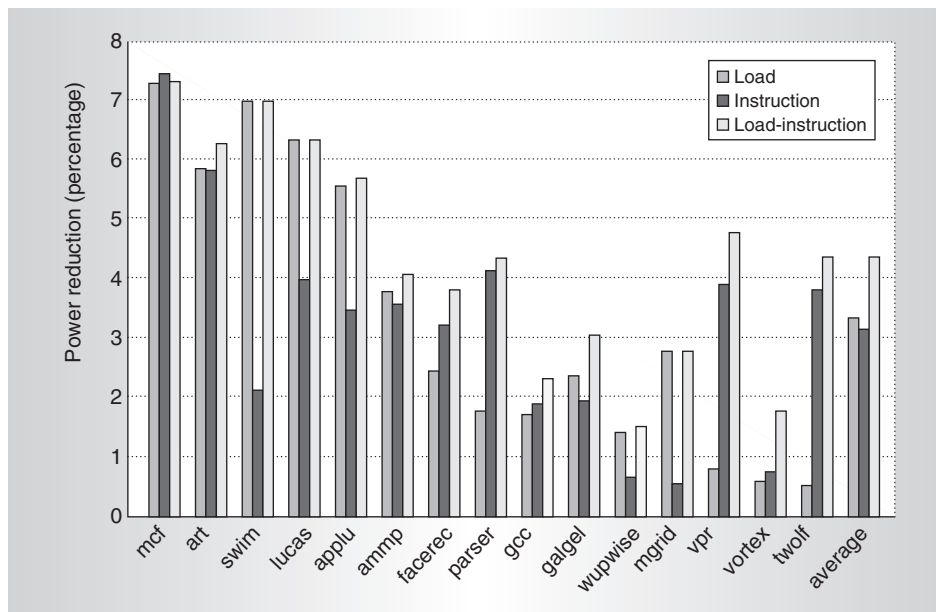


Figure 5. Processor power reduction achieved by three schemes.

tor's 4.2 percent power saving with a performance loss of 0.7 percent. For some applications, the power-saving difference is large. For example, for program *swim*, the load indicator saves 7.0 percent of total power while the instruction indicator saves 2.1 percent of total

power. For this program, 60.4 percent of sampling windows have IPC values higher than the thresholds under normal execution, although almost no instructions are issued in 21.9 percent of the sampling windows. The load indicator scheme foresees low-demand and idle periods and delays current work. Thus, the processor can stay in low-power mode far longer without causing performance loss. For all 15 applications, the average power reductions by the load indicator and the instruction indicator are 3.3 and 3.1 percent, respectively.

The load indicator performs better for memory-intensive applications, whereas the instruction indicator performs better for programs with small memory stall portions. The load-instruction indicator combines the merits of both techniques and achieves greater power savings than either. The load-instruction indicator's power reduction ranges from 1.5 to 7.3 percent. The average power savings is 4.4 percent.

The load indicator and the load-instruction indicator don't increase processor energy consumption for any program. However, the instruction indicator increases energy consumption for *facerec* slightly, because the energy saved by the scheme cannot pay off the increase caused by performance loss. The load-instruction technique achieves the lowest energy consumption for 11 programs. The load-instruction indicator's average energy reduction is 3.2 percent, compared with the load indicator's 2.9 percent and the instruction indicator's 2.2 percent. The load, instruction, and load-instruction indicators reduce the average energy-delay product (*EDP*) by 2.4, 1.3, and 2.0 percent; and the average *ED²P* (energy-delay-delay product) by 1.9, 0.3, and 1.0 percent, respectively.

Adjusting the processor issue rate according to program resource requirements effectively reduces processor power consumption with negligible performance loss. Our study has shown that the existence of memory accesses is a strong indicator reflecting the reduction of a near future resource requirement. For memory-intensive applications, simply applying this indicator can capture most program behavior variants in a timely manner. Compared with the instruc-

tion indicator scheme, the load indicator saves more power with comparable performance losses.

The load-instruction scheme adaptively utilizes both the load indicator and the instruction indicator used in the pipeline-balancing technique. Compared with the load indicator, the load-instruction scheme captures current power-saving opportunities caused by the program's ILP variants. Compared with the instruction indicator, it identifies future program variants and redistributes current work. The load-instruction scheme works well for both computation- and memory-intensive applications. In our continuing work, we are looking for other predictors that can foresee future program variants and perform look-ahead optimizations.

Acknowledgments

We thank the anonymous referees for their constructive comments. This work is supported in part by National Science Foundation grants CNS-0098055, CCF-0129883, and CNS-0405909, and a grant from Hewlett-Packard Labs.

References

1. A. Buyuktosunoglu et al., "An Adaptive Issue Queue for Reduced Power at High Performance," *Power-Aware Computer Systems*, LNCS vol. 2008, Springer Verlag, 2001, pp. 25-39.
2. R.I. Bahar and S. Manne, "Power and Energy Reduction via Pipeline Balancing," *Proc. 28th Ann. Int'l Symp. Computer Architecture (ISCA 01)*, IEEE Press, 2001, pp. 218-229.
3. D. Folegnani and A. González, "Energy-Effective Issue Logic," *Proc. 28th Ann. Int'l Symp. Computer Architecture (ISCA 01)*, IEEE Press, 2001, pp. 230-239.
4. D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing Power Requirements of Instruction Scheduling through Dynamic Allocation of Multiple Datapath Resources," *Proc. 34th Ann. Int'l Symp. Microarchitecture (Micro 34)*, IEEE Press, 2001, pp. 90-101.
5. R. Sasanka, C.J. Hughes, and S.V. Adve, "Joint Local and Global Hardware Adaptations for Energy," *Proc. 10th Int'l Conf. Architectural Support for Programming*

Languages and Operating Systems (ASPLOS 10), ACM Press, 2002, pp. 144-155.

6. O. Unsal et al., "Cool-Fetch: Compiler-Enabled Power-Aware Fetch Throttling," *IEEE Computer Architecture Letters*, vol. 1, April, 2002, pp. 100-103.
7. M. Huang, J. Renau, and J. Torrellas, "Positional Adaptation of Processors: Application to Energy Reduction," *Proc. 30th Ann. Int'l Symp. Computer Architecture (ISCA 03)*, IEEE Press, 2003, pp. 157-168.
8. R. Pyreddy and G. Tyson, "Evaluating Design Tradeoffs in Dual Speed Pipelines," *Proc. Workshop Complexity-Effective Design, in conjunction with 28th Int'l Symp. Computer Architecture*, 2001; <http://www.ece.rochester.edu/~albonesi/wced01/papers/rpyreddy.ps>.
9. J.S. Seng, E.S. Tune, and D.M. Tullsen, "Reducing Power with Dynamic Critical Path Information," *Proc. 34th Ann. Int'l Symp. Microarchitecture (Micro 34)*, IEEE Press, 2001, pp. 114-123.
10. H. Li et al., "VSV: L2-Miss-Driven Variable Supply-Voltage Scaling for Low Power," *Proc. 36th Int'l Symp. Microarchitecture (Micro 36)*, IEEE Press, 2003, pp. 19-28.
11. R. Desikan, D. Burger, and S.W. Keckler, "Measuring Experimental Error in Microprocessor Simulation," *Proc. 28th Ann. Int'l Symp. Computer Architecture (ISCA 01)*, IEEE Press, 2001, pp. 266-277.
12. P. Shivakumar and N. Jouppi, *An Integrated Cache Timing, Power, and Area Model*, tech. report, Compaq Western Research Lab, 2001.
13. M.K. Gowan, L.L. Biro, and D.B. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," *Proc. 1998 Design Automation Conf. (DAC 98)*, ACM Press, 1998, pp. 726-731.
14. D.M. Brooks et al., "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, vol. 20, no. 6, Nov./Dec. 2000, pp. 26-44.

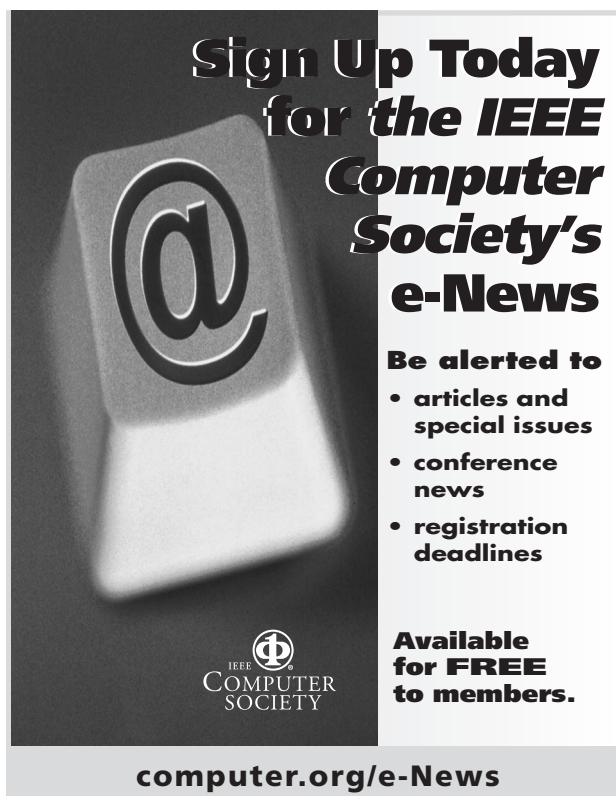
Zhichun Zhu is an assistant professor of electrical and computer engineering at the University of Illinois at Chicago. Her research interests include computer architecture, performance evaluation, and low-power designs. Zhu has a BS in computer engineering from

Huazhong University of Science and Technology, China, and a PhD in computer science from the College of William and Mary. She is a member of the IEEE and the ACM.

Xiaodong Zhang is the Lettie Pate Evans Professor of Computer Science and the department chair at the College of William and Mary. His research interests include parallel and distributed computing and systems and computer architecture. Zhang has a BS in electrical engineering from Beijing Polytechnic University and an MS and a PhD, both in computer science, from the University of Colorado at Boulder. He is a senior member of the IEEE.

Direct questions and comments about this article to Zhichun Zhu, Dept. of Electrical and Computer Engineering, 1020 SEO (M/C 154), University of Illinois at Chicago, Chicago, IL 60607-7053; zhu@ece.uic.edu.

For further information on this or any other computing topic, visit our Digital Library at <http://www.computer.org/publications/dlib>.



**Sign Up Today
for the IEEE
Computer
Society's
e-News**

Be alerted to

- articles and special issues
- conference news
- registration deadlines

Available for FREE to members.

computer.org/e-News