Evaluating Techniques for Exploiting Instruction Slack

Yau Chin, John Sheu, and David Brooks
Division of Engineering and Applied Sciences
Harvard University

Abstract

In many workloads, 25% to 50% of instructions have slack allowing them to be delayed without impacting performance. To exploit this slack, processors may implement more power-efficient, longer latency pipelines or provide dynamically scaled pipelines using multiple clock domains. Issuing instructions with slack to slower pipelines can result in substantial power savings, with minimal performance loss. Considering both dynamic and static power dissipation, we found that by using longer latency pipelines the power of functional unit pipelines decreases by 20% to 55% with a performance impact of 0% to 3% for SPEC2000 and MediaBench workloads. Dynamic scaling reduces the performance loss in intense multimedia workloads by up to 2%, but achieves lower power savings.

1. INTRODUCTION

Increasing processor clock rates leads to higher performance by reducing the run time of the critical instruction path, but this critical path typically constitutes less than 5% of instructions [3]. Non-critical instructions can be executed at a slower rate without affecting overall performance, so in reality, microprocessors run many instructions at higher speeds needlessly. Recent research develops ways to identify these non-critical instructions and introduces the concept of instruction slack [3], which is the number of cycles an instruction can be delayed without impacting subsequent instructions in a program.

Since many applications exhibit regular behavior, predictive schemes can be effective in identifying this slack. Slack predictors can be used to direct certain instructions to reduced speed pipelines with minimal performance loss and executing instructions with slack in slower, low-power circuits can significantly reduce power. Throughput remains unaffected as long as the microprocessor correctly predicts and issues instructions to the pipeline of the adequate speed.

At least two distinct techniques can be used to exploit slack in slower, less power intensive functional units. First, fixed-speed slow pipelines can be designed to augment or replace existing pipelines within the processor core. These fixed-speed pipelines are designed with less aggressive circuits and smaller transistors and provide higher latency operation but with significantly lower power dissipation. Alternatively, standard high-performance functional units could be employed but with the capability to apply dynamic frequency and voltage scaling at the individual functional unit level. These pipelines will be fairly easy to implement because they require little additional control logic.

Dynamic scaling requires multiple clock domains, that is, each functional unit has an independent clock frequency and voltage. Recent work used a clock domain for each type of functional unit [8], but we propose having a clock domain for each individual pipeline. Fine-grained clock domains permit the microprocessor to better fit the current instruction stream, and allow it to respond to increased demand for fast or slow pipelines. This approach offers the ability to conserve power when the opportunity arises without limiting maximum performance.

This paper compares the performance impact and active and leakage power savings through slack exploitation with fixed and dynamically scaled pipelines. The fixed approach with three full speed and three slow pipelines achieves power savings between 20% and 55% in the functional units with a performance loss of 0% to 3%. The dynamic approach with six pipelines is better able to respond to the demands of multimedia workloads. It minimizes performance loss by up to 2% but also reduces power savings.

2. RELATED WORK

Dynamic voltage scheduling examines application behavior at run-time to produce the schedules. Pering et al. [6] use the idea of application deadlines and target IPCs to dynamically vary processor frequency and voltage. This type of analysis works well for applications with established deadlines such as MPEG decoders.

Semeraro et al. [8] proposes dynamic voltage scaling at a finer level of granularity through the use of multiple clock domains. With multiple clock domains, major blocks (front-end, load-store unit, and execution units) may operate at different voltages and frequencies. The proposed online scheme for determining the voltage and frequency is based on the utilization of the units and does not consider whether an instruction is on the critical path or has slack.

Pyreddy et al. [7] and Seng et al. [9] identify non-critical instructions using techniques for determining criticality and direct those instructions to low-power pipelines. Their models have only two levels of granularity: critical and non-critical. Casmira et al. [2] extend the idea of critical path scheduling to evaluate how many instructions have slack at any given cycle. Fields et al. [3] continue in this domain and formally define three types of instruction slack, assess the availability of slack, and propose a prediction algorithm to identify slack. However, their research only briefly discusses how to exploit slack in microprocessors.

The main contribution of this work is to compare several schemes for exploiting predicted slack. In doing so, we build on the work of Fields et al. in identifying and predicting slack. We consider fixed, low-power pipelines as well as dynamically scaled pipelines under a multiple clock domain architecture. Dynamic scaling of the pipelines offers some advantages over fixed pipelines, and issuing instructions to pipelines of different speeds based on their slack provides a finer level of granularity than the previous work. In our multiple clock domain architecture, we also consider individual functional unit pipelines as separately scalable units, as opposed to the coarse grained blocks considered in [8].

Instruction Window	128-RUU, 64-LSQ
Integer Functional Units	4 iALU (1cycle), 2 iMul (4 cycle)
FP Functional Units	2 fpALU (4 cycle), 2 fpMul (6 cycle)
Pipeline Width	4-issue, 4-commit, 2 memory ports
L1 D-Cache	16KB 4-way, 3 cycle latency
L2 I/D-Cache	256KB 4-way, 9 cycle latency
Main Memory	100 cycle latency

Table 1. Baseline processor parameters.

3. EXPLOITING SLACK

We consider two approaches to exploit slack for low-power operation: fixed, low-power, longer latency pipelines and standard, high-performance pipelines that are broken into multiple clock and voltage domains and are dynamically tunable depending on the availability of slack at that point in the instruction stream. In both scenarios, each instruction is issued to a specific pipeline based on how much slack is predicted for it.

Table 1 describes our baseline processor architecture. For the fixed approach, we consider two changes to the execution units in the baseline processor: fixed-4 and fixed-6. In the fixed-4 scenario, which we call $3f + 1s$, we consider three fast ALUs combined with one ALU operating with a latency of 2 cycles, or half-speed. The alternative is the fixed-6 scenario with $3f+2s+1s'$ where two ALUs run at half speed and one ALU runs at one-third speed. This design includes a small area and leakage power overhead (discussed in detail in Section 4.2) because of the additional two ALUs. However, this configuration limits the performance loss of the processor when compared to the baseline. We chose these two configurations mainly because $2f + 2s$ had very poor performance and $3f+2s+1s'$ generally performed as well as $4f + 2s$ but had higher power savings.

Similarly, we examine two scenarios for the dynamic approach with 4-ALU and 6-ALU pipelines: dynamic-4 and dynamic-6. The ALU pipelines can run at 1x (full speed), 2x, and 3x. Each pipeline is in a separate frequency and voltage domain, but this design is simpler than the multiple clock domain architecture proposed in [8], since the pipelines are always running at an integer multiple of the global clock. The dynamic approach can reduce power dissipation for slack-dominated applications while retaining the capacity for full-speed performance.

For each execution unit, the dynamic scaling algorithm looks at the distribution of predicted slack within fixed time intervals and adjusts the frequency and voltage accordingly. For example, during one time interval, 55% of integer ALU instructions may have no local slack, 30% have 1 cycle, and 15% have 2 or more cycles. It records the cumulative distribution so it knows that 15% have at least 2 cycles and 45% have at least 1 cycle.

Each type of execution unit has predetermined thresholds specified as a percentage for each speed. The pipeline is scaled to the highest level of delay that meets its threshold. The cumulative distribution reflects the demand for slower functional units, and the threshold is the minimum demand required to slow down a pipeline. In the previous example, if the thresholds for 4 integer ALUs were 10%, 30%, 50%, and 80%, one pipeline would run at 3x and another pipeline would run at 2x. If fewer instructions have slack, the cumulative distribution of slack may fall below the thresholds, thus causing more pipelines to run at full speed. The thresholds for dynamic-4 are 15%, 70%, 80%, and 90%. The thresholds for dynamic-6 are 10%, 20%, 30%, 55%, 80%, and 90%.

Implementation Style	Delay	Fixed Active	Fixed Leakage	DVS Active	DVS Leakage
Kogge-Stone/Dyn.	1x	1x	1x	1x	1x
QuadTree/Static	2x	.1x	.2x	.125x	.5x
Carry-Skip/Static	3x	.05x	.1x	.015x	.33x

Table 2. Latency vs. Power Tradeoffs.

4. IMPLEMENTATION DETAILS

4.1. Slack Prediction

The slack predictor is required to predict the number of cycles of available slack for individual instructions. Our predictor is a PC-indexed table with 2K entries which we found to be sufficient to accommodate the large loop structures in our benchmark suites. Each entry has 4-bit fields expressed in cycles for the prediction, the observed slack, and a counter for the cycles elapsed since the completion of the instruction. The counter increments only when the active bit is set, which is waiting for the first child instruction to issue. The tag consists of the 3 bits above the bits in the PC used to select the entry. The entry is activated upon the writeback of the instruction until an instruction uses its result. In addition, the register metadata and instruction tags track instruction dependencies using 11-bit fields that index into the predictor table.

The writeback stage sets the counter to the actual delay and enables the 4-bit counter. The decode stage determines the parents of an instruction using the register metadata. The issue stage resets the active flag of the parents' predictor entries and updates the prediction.

The total predictor size is roughly 5 KB, but it is important to note that, unlike a branch predictor, the result of the prediction is not required until the issue stage which is typically four or more cycles later. The array structure can be built with power-efficient techniques that trade latency for power dissipation such as hierarchical banking techniques similar to phased second-level caches. In addition, the entries could most likely be further reduced in size by sharing data with other structures in the microprocessor that track similar data. For example, out-of-order architectures already track dependency information.

4.2. Fixed Approach

We consider two approaches for modifying the execution pipelines in the microprocessor to trade latency for power dissipation. The first approach replaces or augments the existing high-performance execution units with slower pipelines that operate at a fixed speed. There are many tradeoffs that designers may employ to trade latency for power dissipation in function units including choice of adder styles, dynamic vs. static logic, transistor sizing, and threshold voltage selection.

Oklobdzija et al. [5] analyze many energy-delay tradeoffs in the implementation of high-performance microprocessor adders. The paper reports a 2x increase in delay and a 5x decrease in energy moving from a fast, dynamic logic, Kogge-Stone adder to a static logic, reduced transistor sized, Quaternary-Tree adder. For our 3x slower pipeline, we switch from the Kogge-Stone adder to a carry-skip adder with high-Vt transistors. As for leakage power, the energy savings in [5] track closely with fewer, smaller transistor device widths. Thus, we assume a 5x reduction in the static power dissipation for the half-speed pipeline. Furthermore, Kim et al. [4] indicate that the use of high-Vt transistors can provide 4x leakage savings with a 35% slowdown. We estimate a 20x leakage reduction for the 3x slower pipeline. These assumptions are summarized in Table 2.

4.3. Dynamic Approach

Dynamic frequency and voltage scaling within multiple clock and voltage domains [8] provides another method to trade latency for power. We assume multiple clock domains for execution pipelines operating at integer multiples of the clock, 1x, 2x, and 3x, thus simplifying domain synchronization. We assume that increasing pipeline latency with DVS can provide cubic reduction in active power dissipation. In addition, leakage power reduces linearly with supply voltage. These assumptions are summarized in Table 2.

As described in Section 3, dynamic scaling compares the slack distribution against the preset thresholds to determine the appropriate frequency and voltage for the pipelines. It considers the slack distribution every 16K cycles, or 8 microseconds on a 2 GHz processor, and adjusts the frequency and voltage as needed. For each type of functional unit, the distribution of predicted slack is stored as a small array of three 13-bit integers representing the number of 1x, 2x, and 3x slack predictions. The implementation can also be very power-efficient because it does not need to complete in a short amount of time.

5. RESULTS AND ANALYSIS

We simulated the slack predictor with fixed and dynamic scaled pipelines of different speeds with SimpleScalar. The SPEC2000 benchmarks were executed for 100 million instructions starting at the 1-billion checkpoint, while the MediaBench benchmarks were executed in their entirety.

5.1. Simulation Results

Figure 1 shows the performance degradation of each benchmark under dynamic and fixed scaling. The performance loss of the fixed-6 scheme is at most 3%, with the exception of *mpeg2d*. Fixed-4 is significantly worse for the multimedia workloads due to the insufficient number of full speed pipelines, with performance losses of 2% to 5%. Dynamic-6 addresses this problem by increasing the speed of functional units, and so performs better than the fixed approach for many of the media applications. However, limiting the number of pipelines to four in the dynamic approach further impacts performance.

Figure 2 shows the reduction in dynamic power dissipation in the integer and floating point ALUs, relative to a baseline processor with fine-grained clock gating. The

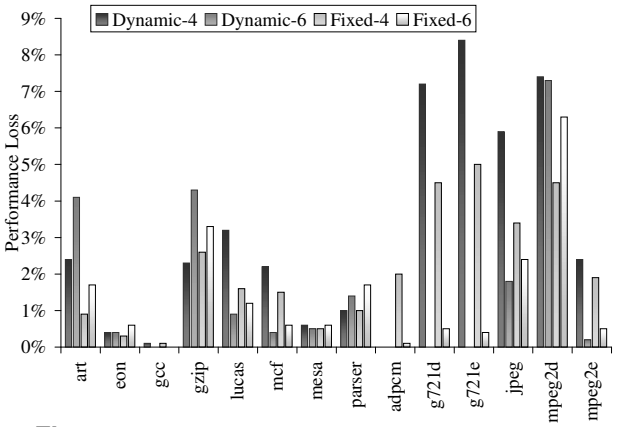

Figure 1. Performance loss after applying slack.

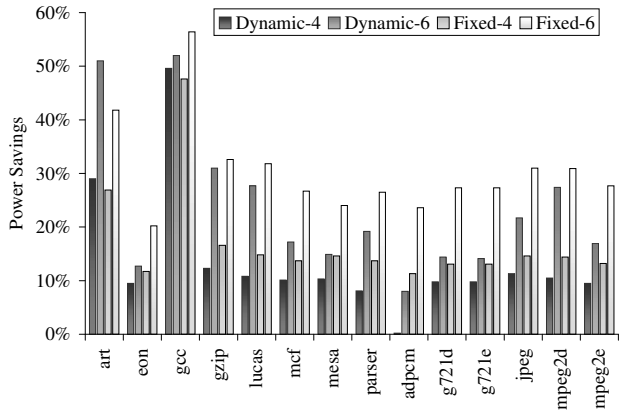

Figure 2. Dynamic power in execute pipelines.

power savings for *art* and *gcc* is much higher than the rest, by up to 20%, due to more instructions with slack and greater amounts of slack.

With six pipelines, the fixed scheme generally reduces power by 24%-33%, and the dynamic scheme performs slightly worse, saving between 15% and 27%. The fixed-4 configuration is less aggressive and consequently has a much lower power savings of 11% to 17%. The thresholds for dynamic-4 are even more conservative, and power savings are generally between 8% and 12%. The savings depend largely on the fixed speed configuration and the dynamic scaling thresholds. Making them more aggressive would yield greater energy savings, while further reducing performance.

The power savings of the microprocessor core is roughly 3% to 5% under dynamic scaling and 5% to 7% under fixed scaling, assuming the execution units consume 20% of core power, as reported by Bose et al. [1] for the POWER4.

5.2. Energy-Delay^2 Savings

Design modifications that demonstrate reduction in ED^2 indicate the techniques achieve additional power savings that would not result from simply scaling back the global clock frequency and voltage of the processor. Figure 3 shows the change in ED^2 for the entire microprocessor core after exploiting slack. Fixed-6 achieves an ED^2 reduction

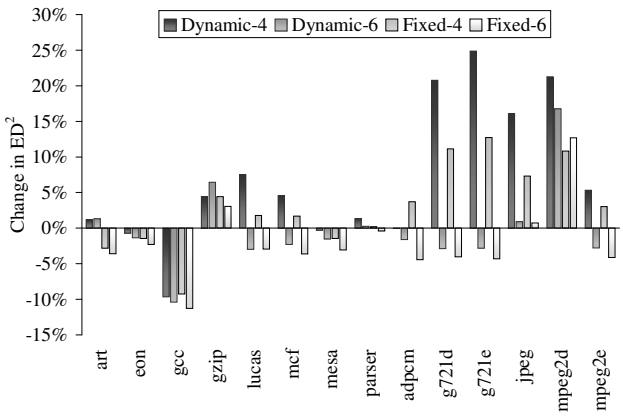

Figure 3. Processor Core Energy-Delay^2.

of 3% or more in the majority of the benchmarks and breaks even in a few, but it performs worse in *gzip* and *mpeg2d*.

The ED^2 chart highlights the effectiveness of the fixed-6 scheme, and at times, the dynamic-6 scheme. The improvements are substantial for *gcc* because it has large amounts of slack due to the high frequency of loads and stores. This may also indicate that memory and I/O bound workloads such as TPC-C and other server applications would benefit substantially. At the opposite extreme, the *mpeg2d* and *gzip* benchmarks have tight loops with thousands of iterations, and slack mispredictions incur high performance penalties. A better slack predictor would greatly enhance the ED^2 numbers for these benchmarks.

The media benchmarks underscore a clear difference between four and six pipelines. The latter configuration is significantly better in power and performance. For fixed-4, replacing a full speed pipeline with a half speed one decreases the bandwidth of the ALU pipelines from four to three and a half. Multimedia workloads often have enough parallelism to have four instructions issued per cycle, so substantial performance loss in a fixed-4 configuration is unavoidable.

In theory, dynamic-4 should be able to scale the pipelines appropriately to minimize the performance impact. However, our dynamic scaling is based on the distribution of slack meeting the preset thresholds. The media benchmarks do have at least 15% of instructions exhibiting slack, but our algorithm fails to consider whether a change in bandwidth will impact performance. Dynamic-6 does not have this problem, because even if it slows down three pipelines the bandwidth will still be at least four. As such, it performs better than fixed-6 on most of the benchmarks.

Nevertheless, the ED^2 numbers for fixed-6 are almost always better than those of dynamic-6, since the preset thresholds for dynamic-6 are fairly conservative, significantly reducing the power savings. Our analysis suggests that it is always best to have three full speed and two slower functional units, so we set the thresholds accordingly. This suggests a hybrid approach with several fixed fast and slow pipelines and one dynamic pipeline.

5.3. Static Power

The discussion so far has focused on dynamic power savings, but slower pipelines also reduce static power dissipation, and the additional pipelines require us to consider the leakage power of the increased area. For our analysis we assume that leakage power is roughly 20% of the total power representing next generation process technologies.

Our assumptions summarized in Table 2 anticipate that slowing down a pipeline decreases the active power more than the leakage power, the total power savings is actually less than the dynamic power savings. In dynamic-4, this delta is at most 1% with the exception of *gcc* and *art*, which had higher active power savings to begin with. However, for dynamic-6, the addition of two full-speed pipelines increases the area of the ALUs by 50%. The delta in this case is as much as 13%. This further emphasizes the need for a hybrid approach, where only fixed slower pipelines which have less area are added.

Like dynamic-4, the fixed-4 configuration has a total power savings of at most 1% less than the active power savings. As for fixed-6, the addition of two half-speed pipelines with a fifth of the area of a full-speed pipeline increases the area by only 10%. Unlike dynamic-6, the delta between total and active power savings is at most 2%. Even if leakage power is included, the fixed-6 scheme still provides substantial power savings.

6. CONCLUSION

This paper examines the use of dynamic slack scheduling based on a predictive algorithm to reduce power by issuing non-critical instructions to slower pipelines. We evaluate two approaches for configuring the pipelines: fixed and dynamic. The fixed-6 configuration achieved 20% to 55% reduction in the active power of the execution units while keeping performance loss between 0% and 3%. Certain workloads have enough parallelism so that the processor is able to issue four instructions at once, and fixed, slower pipelines limit the maximum bandwidth. Dynamic scaling of one or two of the pipelines mitigate this performance impact as demonstrated by the better performance of the multimedia benchmarks.

References

[1] P. Bose et al. Early-stage definition of LPX: A low power issue-execute processor. In *PACS'02 at HPCA*, 2002.

[2] J. Casmira and D. Grunwald. Dynamic instruction scheduling slack. In *Koolchips 2000 Workshop*, December 2000.

[3] B. Fields, R. Bodik, and M. D. Hill. Slack: Maximizing performance under technological constraints. In *29th International Symposium on Computer Architecture*, May 2002.

[4] S. Jung et al. Dual threshold voltage domino logic synthesis for high performance with delay and power constraint. In *Design, Automation and Test in Europe*, 2002.

[5] V. Oklobdzija et al. Energy-delay estimation technique for high-performance microprocessor VLSI adders. In *International Symposium on Computer Arithmetic*, June 2003.

[6] T. Pering, T. Burd, and R. Broderson. Voltage scheduling in the lpARM microprocessor system. In *ISLPED*, July 2000.

[7] R. Pyreddy and G. Tyson. Evaluating design tradeoffs in dual speed pipelines. In *Work. on Complex. Eff. Design*, June '01.

[8] G. Semeraro et al. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *35th International Symposium on Microarchitecture*, November 2002.

[9] J. Seng, E. Tune, and D. Tullsen. Reducing power with dynamic critical path information. In *MICRO34*, Dec. 2001.