Area-Aware Pipeline Gating for Embedded Processors

Babak Salamat and Amirali Baniasadi
Electrical and Computer Engineering
University of Victoria
{salamat, amirali}@ece.uvic.ca

Abstract. Modern embedded processors use small and simple branch predictors to improve performance. Using complex and accurate branch predictors, while desirable, is not possible as such predictors impose high power and area overhead which is not affordable in an embedded processor. As a result, for some applications, misprediction rate can be high. Such mispredictions result in energy wasted down the mispredicted path. We introduce area-aware and low-complexity pipeline gating mechanisms to reduce energy lost to possible branch mispredictions in embedded processors. We show that by using a simple gating mechanism which comes with 33-bit area overhead, on average, we can reduce the number of executed instructions by 17% (max: 30%) while paying a negligible performance cost (average 1.1%).

1. Introduction

Modern embedded processors aim at achieving high-performance while maintaining die area and power consumption at a low level. While technology advances continue to provide embedded processors with more resources, we are still far away from affording advanced complex techniques. It is due to such restrictions that embedded processors do not exploit many techniques frequently used by high-performance desktop machines or servers. One way to narrow this gap is to revisit solutions introduced for high performance processors to develop affordable implementations.

Branch prediction is essential as it provides steady instruction flow at the fetch stage which in turn results in shorter program runtime. However, unfortunately, predictors are not perfect and make mispredictions. Such branch mispredictions result in longer program runtimes and energy wasted down the mispredicted instruction path. Compared to a high-performance processor, the energy lost to mispredictions appears to be more costly in embedded processors. This is due to the fact that in an embedded processor power resources are limited to batteries, making efficient power management even a more critical task.

In addition, embedded processors are bound to exploit simple and possibly less accurate branch predictors compared to high performance processors. Using more simple and less accurate branch predictors by embedded processors compared to high performance processors is exemplified by the XScale processor: Intel's XScale which was introduced in 2001 [2] uses a 256-bit bimodal predictor while the Alpha EV6 [3] that was released in 1997 used 36Kbits.

More importantly, it is expected that as future embedded processors start exploiting deeper pipelines, branch misprediction cost will increase even further.

To address these concerns finding techniques to reduce energy lost to speculation while maintaining performance is essential.

A previous study has suggested using pipeline gating to reduce energy lost to branch misspeculation [1]. Unfortunately, previously suggested pipeline gating mechanisms are either too complex to be used in an embedded processor or depend on complex branch predictors which are not affordable in an embedded processor.

The goal of this work is to introduce a set of power-efficient and area-aware pipeline gating methods to reduce misprediction cost while maintaining performance. In particular, we introduce three very low-overhead pipeline gating mechanisms to reduce energy lost to branch misprediction while maintaining performance.

Pipeline gating relies on accurate branch confidence estimation [6]. We also introduce low-overhead branch confidence estimation techniques to be used in embedded processors.

Our most aggressive technique reduces the number of mistakenly fetched instructions by 17% (max: 30%) with an average performance loss of 1.1%.

The rest of the paper is organized as follows. In section 2 we discuss our motivation. In section 3 we explain our techniques in more details. In section 4 we present methodology and results. In section 5 we review related work. Finally, in section 6 we offer concluding remarks.

2. Motivation

As explained earlier, modern processors lose energy due to possible branch mispredictions. We refer to the mistakenly fetched instructions as *wasted activity* (WA). Note that mispredicted instructions do not commit and are flushed as soon as the mispredicted branch is resolved. As such, we define WA as:

$$WA = \frac{fetched - commited}{fetched}$$

Where *fetched* and *committed* are the numbers of instructions fetched and committed during execution of a program, respectively.

In figure 1 we report the percentage of instructions fetched down the mispredicted path. We report for a subset of MiBench benchmarks [5] and for a processor similar to Intel's XScale. We include benchmarks with both high and low WA in figure 1 and through this study. As presented, WA is more than 20% for three of the applications studied here. To understand why different applications come with different WAs it is important to take into account other parameters including branch misprediction rate.

To explain this further, in figure 2 we report misprediction rate of the bimodal predictor used in XScale for the selected benchmarks. As reported in figures 1 and 2, the four applications with higher WAs also show higher misprediction rates.

While both high-performance processors and embedded processors lose energy to branch mispredictions, it is important to take a different approach in embedded processors. Our study shows that front-end gating should be done more carefully in an embedded processor where front-end buffers are usually small in size. In a high-performance processor, exploiting large size buffers makes a steady instruction flow possible in the event of a fetch stall (*e.g.,* stalls caused by a cache miss). However, in embedded processors, where less resource is available to the processor front-end, stalling the front-end for correctly predicted instructions can impact performance dramatically (*more on this later*).

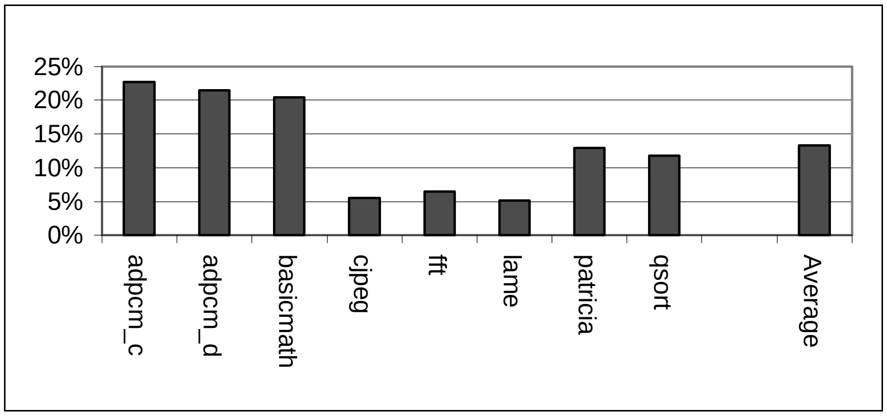

Figure 1. Wasted activity in fetch stage.

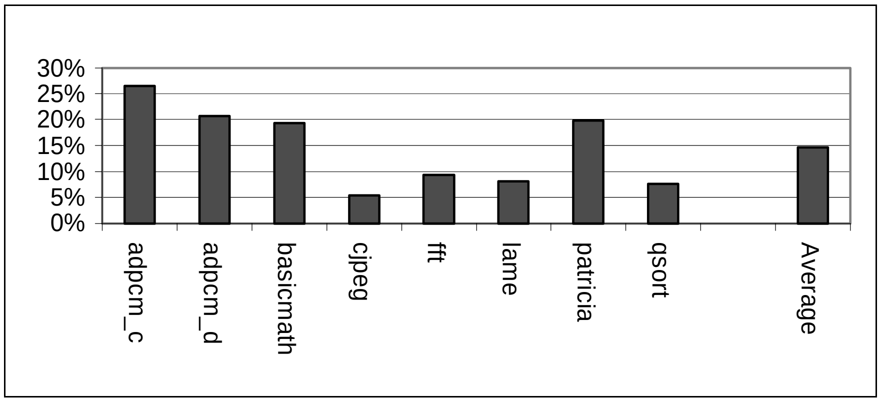

Figure 2. Misprediction rate for XScale's bimodal predictor.

3. Area-Aware Confidence Estimation

We present the schematic of a processor using pipeline gating in figure 3.

Our goal is to stall instruction fetch when there is a high chance that the fetched instructions will be flushed. To do this we gate the pipeline front-end when there is low-confidence in the executed instructions. In order to gate the pipeline while maintaining performance we need a mechanism to identify low confidence branches.

Several studies have introduced accurate confidence estimators. However, previously suggested estimators rely on exploiting complex structures which may not be affordable in an embedded processor. To apply pipeline gating to an embedded processor, we introduce three accurate confidence estimation techniques which impose very little overhead.

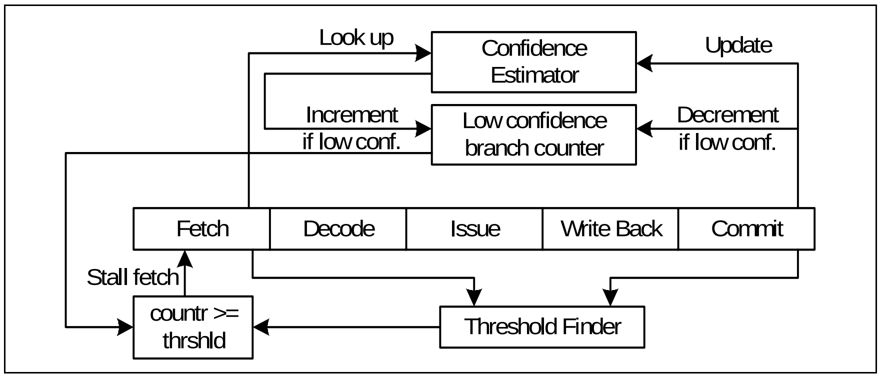

Figure 3. A schematic of a processor using confidence estimation and pipeline gating.

3.1. History-Based Confidence Estimation

In this method we assume that recently mispredicted branches are more likely to be mispredicted in the future. As such we keep track of recently fetched branch instructions' confidence using a very small 16-bit structure. This structure is a PC-indexed 8-entry table where there is a 2-bit counter associated with each entry. The 2-bit counter is incremented for accurately predicted branches. We reset the associated counter if the branch is mispredicted. We look up this structure at fetch and in parallel to probing the branch predictor. If the 2-bit counter is not saturated we consider the branch as low confidence.

Previously suggested pipeline gating methods gate the front-end if the number of low-confidence branches exceeds a pre-decided threshold. Our study shows that a one-size-fits-all approach does not work well across all applications and may result in either low WA reduction or high performance penalty. Therefore, we add a level of adaptivity and decide the gating threshold dynamically. To decide the gating threshold dynamically, we use the number of in-flight branch instructions and average misprediction rate.

For applications with small number of branches, aliasing is low and our confidence estimator is more likely to do a more effective job. This is particularly true if average misprediction rate for an application is low. As such, for applications with lower number of branches or low misprediction rate, we gate the pipeline if the number of low-confidence branches exceeds one. For applications with higher number of branches and higher misprediction rates, we gate the pipeline if the number of low confidence branches exceeds two. Intuitively, for application with high number of branches and high misprediction rates, we need to see at least two low-confidence branch instructions before losing confidence in the following instructions.

Accordingly, history-based confidence estimation requires measuring the number of instructions per branch (IPB) and the number of mispredictions occurring during regular intervals. We measure IPB every 256 instructions and set the threshold to two if IPB drops below 4 (indicating a high number of branch instructions) and if the misprediction rate is above 12.5%. Misprediction rate is measured by shifting the number of branches 3 bits to right (divide by 8) and comparing the result with the number of mispredicted branches.

3.2. Predictor-Based Confidence Estimation

In the second method we assume that the saturating counters which are already being used by the branch predictor indicate branch instruction confidence. By using the already available structures we minimize the hardware overhead associated with pipeline gating.

At fetch, and while probing the branch predictor to speculate the branch outcome, we mark a branch as low confidence if its corresponding branch predictor counter is not saturated. Similar to the history-based method we gate the pipeline if the number of low-confidence branches exceeds a dynamically decided threshold. We increase the gating threshold from 1 to 2 if IPB drops below 4.

3.3. Combined Confidence Estimation

Our study shows that, often, each of the two methods discussed above captures a different group of low-confidence branch instructions. To identify a larger number of low-confidence branches, in this method we use a combination of the two techniques. A branch is considered low-confidence if either the history-based or predictor-based confidence estimator marks it as low-confidence. By using this technique we are able to achieve higher WA reduction while maintaining performance. Similar to the methods discussed above, we also maintain area overhead at a very low-level.

3.4. Area Overhead

In the history-based technique we use an 8-entry confidence estimator which contains 8 2-bit counters. Besides, we need an 8-bit counter to count the instruction intervals, a 6-bit saturating counter with shift capability to count the number of branches in each interval and a 3-bit saturating counter to keep track of mispredictions. The total area requirement is equivalent to 33 bits which is very small.

The area overhead is even lower for the predictor-based method. For this method we need only an 8-bit counter and a 6-bit saturating counter to keep track of instruction intervals and the number of mispredicted branches respectively. Thus, the total required area is only 14 bits.

For the combined method, we use the same structures as we used in the history-based technique. We also look up branch predictor counters which already exist in the processor. Thus, the area requirement is the same (*i.e., 33-bits*) as the history-based technique.

4. Methodology and Results

In this section we present simulation results and analysis for the three proposed methods. We report WA reduction is section 4.1. We report the impact of pipeline gating on performance in section 4.2.

To evaluate our techniques, we used a subset of MiBench benchmark suite compiled for MIPS instruction set. We picked benchmarks with both high and low WA. The results were obtained for the first 100 million instructions of the benchmarks. We performed all simulations on a modified version of the SimpleScalar v3.0 tool set [4]. We used a configuration similar to that of intel's XScale processor for our processor model. Table 1 shows the configuration.

Table 1. Configuration of the processor model.

Issue Width	In-Order:2
Functional Units	1 I-ALU, 1 F-ALU, 1 I-MUL/DIV, 1 F-MUL/DIV
BTB	128 entries
Branch Predictor	Bimodal, 128 entries
Main Memory	Infinite, 32 cycles
Inst/Data TLB	32 entries, fully associative
L1 - Instruction/Data Caches	32K, 32-way SA, 32-byte blocks, 1 cycle
L2 Cache	None
Load/Store queue	8 entries
Register Update Unit	8 entries

4.1. Wasted Activity Reduction

As explained earlier a considerable number of instructions fetched in an embedded processor are flushed due to branch mispredictions. As expected, WA is higher for benchmarks with higher misprediction rates and low IPBs. An immediate consequence of WA is higher power dissipation. Thus, reducing extra work will ultimately result in lower energy consumption as long as performance is maintained.

In figure 4 we report WA reduction for the three proposed confidence estimation techniques. As it can be seen, the highest WA reduction is achieved using the combined confidence estimation technique. Maximum WA reduction is 30% and achieved by the combined method for *fft*.

4.2. Performance

In figure 5 we report performance. Bars from left to right report performance for history-based, predictor-based and the combined method compared to a processor that does not use pipeline gating. Reportedly, the predictor-based technique has the lowest amount of performance loss among the three techniques. Average performance loss is only 0.2% for this technique. Our study shows that, for *patricia*, applying pipeline gating results in an increase in L1 cache hit rate which explains why we witness about 0.4% performance improvement for this benchmark. Average performance loss for history-based and combined techniques is 1.1% and 1.2% respectively.

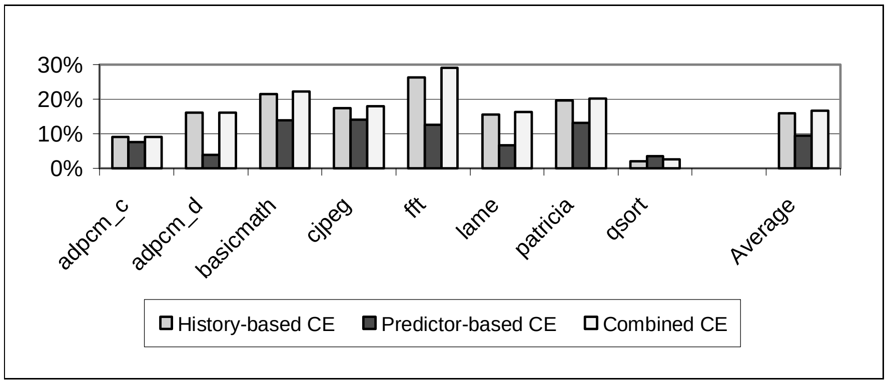

Figure 4. WA reduction. *Higher is better.*

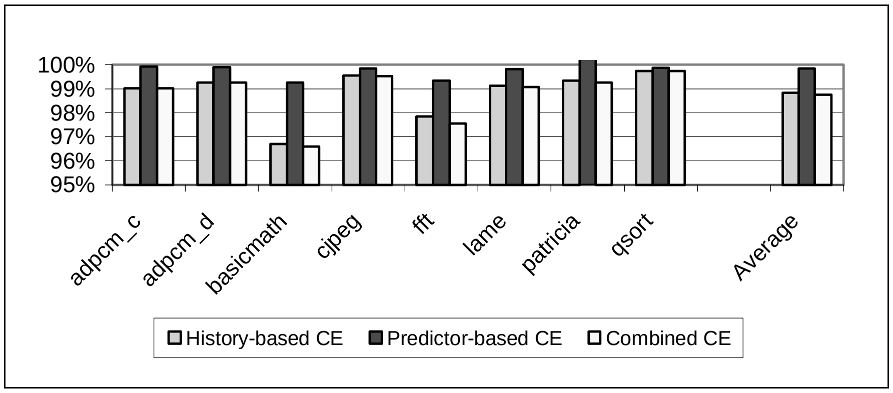

Figure 5. Performance compared to a conventional processor. *Higher is better.*

The predictor-based method maintains performance cost below 1% for all applications. History-based and combined maintain performance cost below 1% for 6 of the 8 applications. Both techniques result in a performance cost of about 3% for *basicmath*. This is the result of frequent changes of behavior for branch instructions in *basicmath*. This makes capturing branch instruction confidence very challenging by using simple confidence estimators designed to perform under resource and area constraints.

5. Related Work

Several studies proposed power efficient architectures for embedded processors. Our work focuses on reducing wasted activity by using pipeline gating.

S. Manne, A. Klauser and D. Grunwald [1] have investigated effects of pipeline gating on reducing wasted activity in high-performance processors. They proposed techniques for branch confidence estimation and used them for pipeline gating in high performance processors. Our study is different from theirs as we propose low overhead techniques for embedded processors. We also propose a dynamic method to change the gating threshold.

6. Conclusion

We proposed three low-overhead pipeline gating techniques for embedded processors. To evaluate our techniques, we used a representative subset of MiBench benchmarks and simulated a processor similar to Intel's XScale processor.

We showed that by using simple confidence estimation techniques, it is possible to reduce the number of mispredicted instructions fetched by up to a maximum of 30%. All proposed techniques maintain average performance cost below 1.2%.

7. Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program, Canada Foundation of Innovation, New Opportunities Fund and the University of Victoria Fellowship.

References

1. Manne, S., Klauser, A., Grunwald, D.: Pipeline Gating: Speculation Control for Energy Reduction. *Proc. 25th Ann. Int'l Symp. Computer Architecture,* pp. 132-141, June 1998.
2. Intel, *Intel XScale Microarchitecture*. 2001.
3. Digital Semiconductor: *DECchip 21064/21064A Alpha AXP Microprocessors: Hardware Reference Manual*, June 1994.
4. Burger, D.C., Austin, T.M.: The SimpleScalar tool set,version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.
5. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representive embedded benchmark suite. *IEEE 4th Annual Workshop on Workload Characterization.* 2001.
6. Grunwald, D., Klauser, A., Manne, S., Pleszkun, A.: Confidence estimation for speculation control. *Proceedings 25th Annual International Symposium on Computer Architecture,* SIGARCH Newsletter, Barcelona, Spain, Jun. 1998.