Power-Aware Scoreboard for Multimedia Processors

Amirali Baniasadi
ECE Department, University of Victoria
Victoria, Canada
amirali@ece.uvic.ca

Babak Salamat
Information and Computer Science Department
University of California, Irvine
bsalamat@uci.edu

ABSTRACT

Intel's XScale which has powered many multimedia applications uses scoreboard to execute instructions. Scoreboard stalls the pipeline whenever a source operand or functional unit is needed but not available. While waiting for the availability of the resources, the processor accesses the scoreboard every cycle. Such accesses consume energy without contributing to performance.

We address this inefficiency by investigating stall behavior and introducing heuristics to avoid regular access to the scoreboard during stall periods.

Our study shows that by using our technique and for the representative subset of MiBench benchmark suite studied here, it is possible to eliminate up to 85% of the useless accesses while maintaining performance cost within 0.5%.

1. INTRODUCTION

The Intel XScale processor has powered multimedia cell phones, handheld computers, in-vehicle (telematics) systems and other wireless Internet products.

Examples of the Intel XScale applications include HP's iPAQ pocket PC H3900 series [1], handheld devices from Fujitsu, Toshiba and Casio Computer [2], Dell Axim X5 handheld [3], and Intergrity RTOS that uses the Intel XScale to deliver high reliability [4].

The complexity associated with detecting, and dealing with data dependencies has become a deciding factor in processor design. With increases in execution bandwidth, processor frequency and the necessity for maintaining power within practical limits, it has become very difficult to accommodate complex algorithms in any superpipelined processor including the Intel XScale. Accordingly, Intel XScale uses a stall-based scoreboard to reduce the complexity associated with detecting data hazards.

XScale uses scoreboard to control the bypass logic and to direct instruction issuing [5]. XScale checks the availability of source operands and functional units at the RF (registerfile/shifter) stage. If either of them is not available, the scoreboard stalls the pipeline. Quite often, the pipeline may stall for several consecutive cycles as it may take sometime before the required data or functional unit becomes available. For example, a cache miss, may result in a long latency memory access which can take several cycles. During this waiting period the scoreboard is accessed frequently and regularly without contributing to performance. This is inefficient as it results in extra power dissipation.

In this work we study stall cycle distribution and use our findings to introduce energy optimization techniques for scoreboard.

Our technique relies on the observation that instructions that stall the pipeline in a scoreboard-based processor tend to repeat their behavior. We use this to predict when an instruction will stall the pipeline and avoid accessing the scoreboard during the stall period.

We show that by using a very small filter referred to as the *Scoreboard Access Filter (SAF)* we can eliminate a considerable number of useless scoreboard accesses while maintaining performance.

To the best of our knowledge this is the first work to introduce power optimization techniques for scoreboard.

The rest of the paper is organized as follows. In Section 2 we discuss stall cycle distribution in more detail. In Section 3 we introduce stall prediction. In Section 4 we present our experimental evaluation. Finally, in Section 5, we summarize our findings.

2. Stall Cycle Accounting

Our goal is to avoid unnecessary scoreboard accesses. These are accesses made while the processor is waiting for a source operand or a resource to become available. We refer to scoreboard accesses occurring during waiting periods as *extra accesses* or *EAs*.

The amount of energy lost due to EAs depends on how often they occur. To investigate this further in Figure 1 we report EA frequency for different benchmarks studied in this work. As presented, EA frequency can be as high as 33%.

EAs can occur both due to the unavailability of data or functional units. Our study shows that, for the applications studied here, the majority (*i.e.*, more than 95%) of stalls are due to source operand unavailability rather than a shortage in execution resources.

If we had an oracle and we knew in advance when a source operand or functional unit becomes ready we could have waited for the moment and avoided all the waisted accesses. Of course, we cannot have such an oracle and do not know when the source operand or execution unit would be ready. An alternative, therefore, is to speculate or predict the availability time.

3. Stall Prediction

We have observed that instructions stalling the scoreboard show a repeating behavior.

In Figure 2 we report stall predictability as measured by an infinite size predictor. Figure 2 reports how often an instruction stalling the scoreboard stalls the scoreboard next time encountered. Minimum predictably is 76% while maximum predictability is

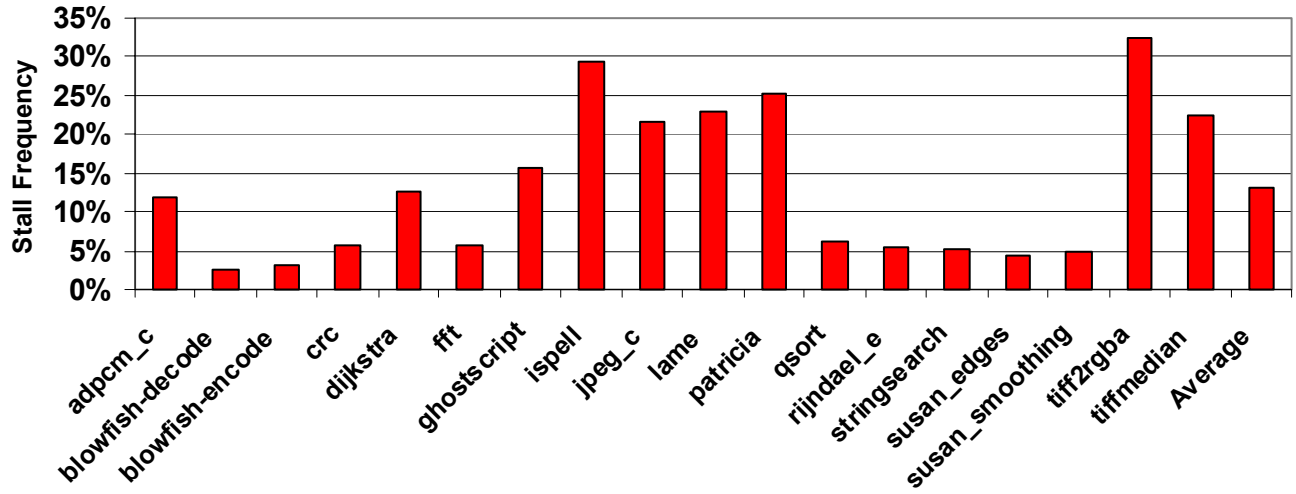

Figure 1: Stall frequency

more than 99%. As presented there is an average chance of 97% that an instruction stalling the pipeline last time will do so again.

We conclude from Figure 2 that there is an opportunity to predict EAs.

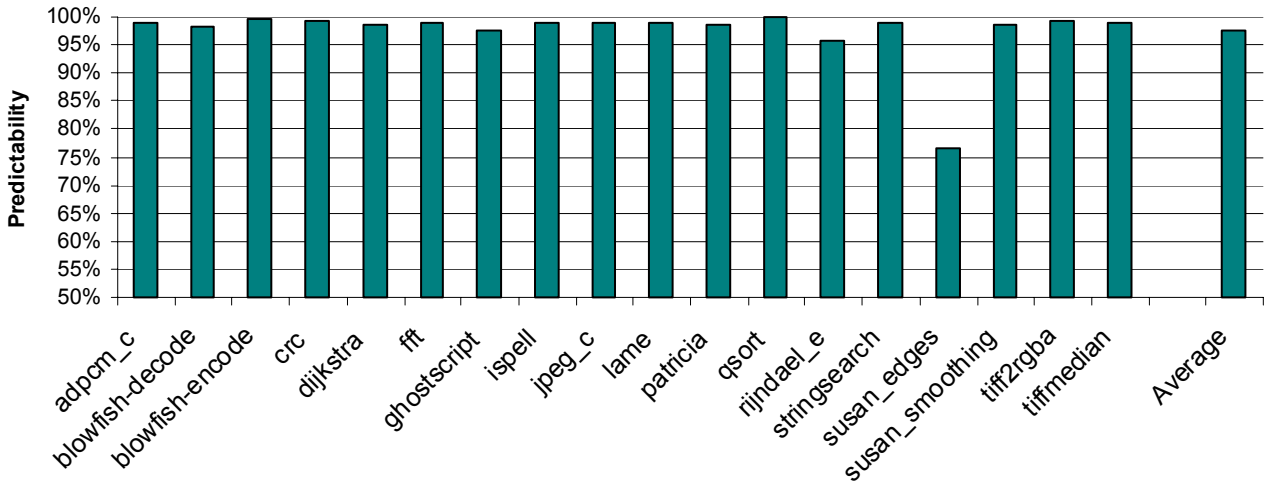

Figure 2: Scoreboard stall predictability

To predict the stalls, we use a small PC-indexed table where a saturating counter is associated with every table entry. We refer to this table as the *scoreboard access filter* or *SAF*. Figure 3 shows the schematic of SAF-enhanced pipeline. When an instruction causes a pipeline stall, we update SAF by incrementing the corresponding counter; otherwise, we decrement the counter. We use saturating counters to detect the most common behavior rather than following the last behavior as our study shows that this maintains performance more effectively.

We probe SAF prior to accessing the scoreboard. We do not allow the instruction to access the scoreboard (and to check resource and data availability) if the corresponding SAF counter entry has a counter saturated to the maximum. We refer to this as gating scoreboard access. Once an instruction is predicted to stall the pipeline we avoid accessing the scoreboard for a pre-decided number of cycles. We refer to this parameter as the gating period. Once the gating period is over we decrement the counter and restart regular scoreboard access.

The timing overhead associated with SAF would not impact the critical path. SAF is accessed using instruction PC which is available as early as fetch and at least two cycles before scoreboard has to be accessed.

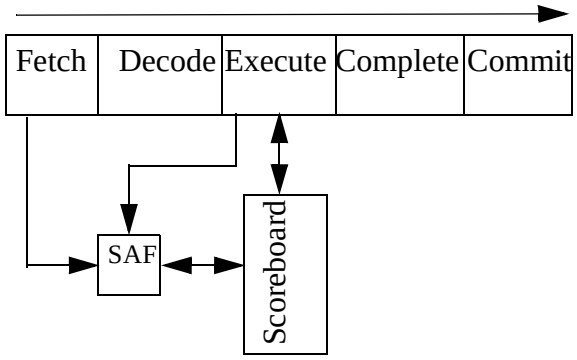

Figure 3: SAF-enhanced Pipeline

4. Methodology and Results

To evaluate SAF we measure both accuracy (*i.e.*, how often an access predicted as an EA turns out to be one) and coverage (*i.e.*, what percentage of EAs are identified by SAF).

To measure SAF accuracy and coverage we used a subset of MiBench benchmark suite compiled for ARM instruction set. All benchmarks were run for 100M instructions. We performed all simulations on a modified version of the Xtrem v1.0 tool which is an XScale simulator [6]. The configuration used for simulation is shown in Table 1.

Table 1: Configuration used for simulation

Issue	In-order
Functional Units	1 I-ALU, 1 F-ALU, 1 I-MUL/DIV, 1 F-MUL/DIV
BTB	128 entries
Main Memory Infinite	22 cycles
Inst/Data TLB	32 entries, fully associative
L1 - Instruction/Data Caches	32K, 32-way SA, 32-byte blocks, 1 cycle
L2 Cache	None

4.1. Accuracy and Coverage

In Figure 4 we report accuracy for 1-, 4- and 8-entry filters. Here we assume that SAF-filter uses a gating period of one, *i.e.*, it gates scoreboard accesses for one cycle upon predicting a stall. Later we report how gating scoreboard accesses for longer periods impacts results. As presented, by using a small predictor we are able to detect stalling instruction with a very high accuracy (*i.e.*, more than 99% on average). Also note that variations in the filter size do not impact our results. This is explained by the small footprint of scoreboard stalling instructions and the in-order execution process.

In Figure 5 we report coverage for 1-, 4- and 8-entry predictors. As presented coverage may vary from one application to another. While we detect about half of the EAs for *ispell*, we hardly detect any EAs for some (*e.g.*, *adpcm_c*). On average we detect more than 20% of the EAs.

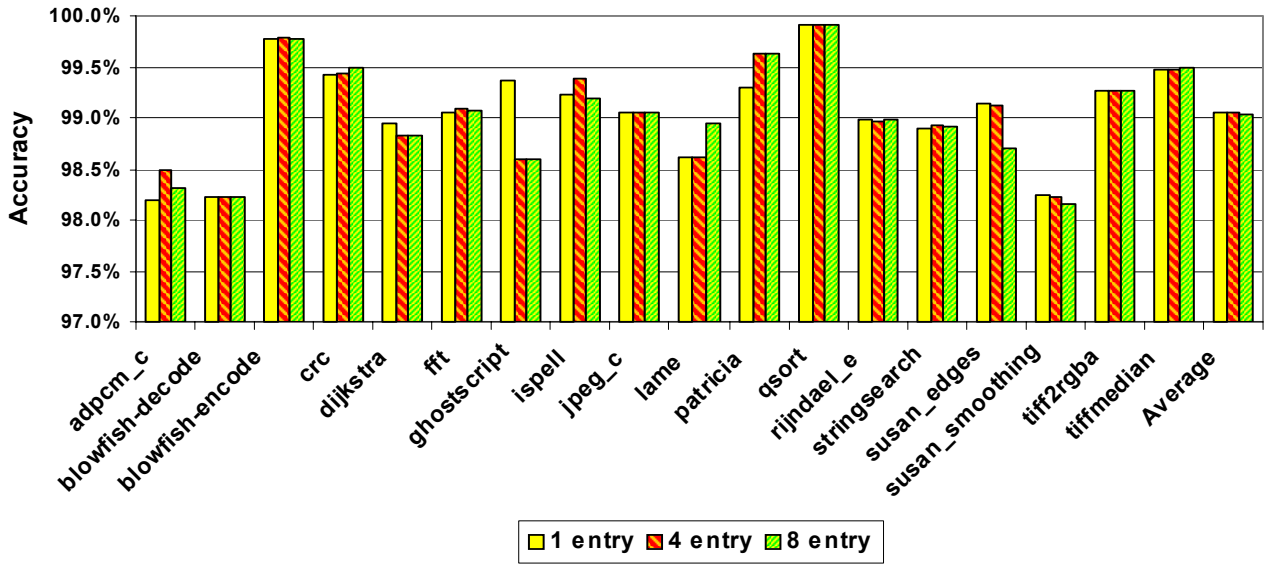

Figure 4: Accuracy of 1-, 4- and 8-entry scoreboard filters

As reported, a single entry table can perform as well as 4 and 8-entry filters. Therefore, in the rest of this study, we focus on a single entry predictor as it comes with minimum area and power overhead.

4.2. Sensitivity to Gating Period

One way to improve the coverage is to increase the gating period. To investigate this further, in this section we report how changing the gating period impacts coverage. In Figure 6 we report

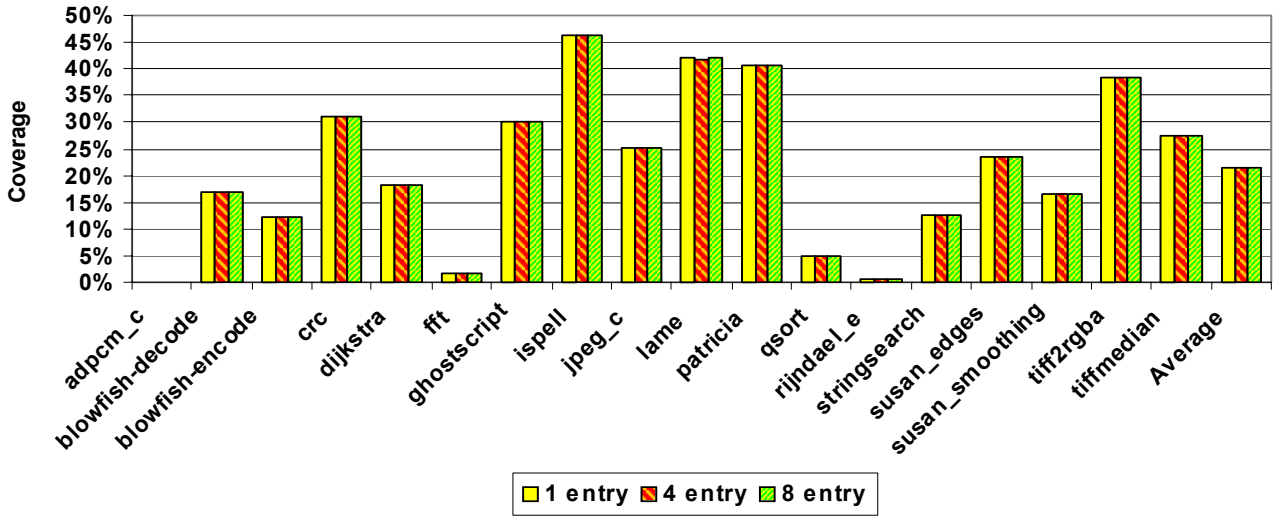

Figure 5: Coverage of 1-, 4- and 8-entry scoreboard filters

coverage when gating period is increased to two, four, eight and 16 cycles. As presented by increasing the gating period coverage is also improved. Average EA coverage is 28%, 35%, 39% and 41% for gating periods of two, four, eight and 16 cycles respectively. Coverage reaches the maximum of 87% for *ispell* and for a gating period of 16.

Increasing the gating period could result in imposing additional delay when the required data or resources become available within a gating period. This, in principle, can negatively impact performance.

To investigate this further, in Figure 7 we report performance loss for different gating periods. Average performance loss is less than 0.5% for all gating periods. Maximum performance loss is 0.1%, 0.3%, 0.5% and 1.5% for gating periods of two, four, eight and 16 cycles respectively. Our study shows that increasing the gating period to more than 16 cycles does not improve coverage considerably but results in much higher performance loss.

The scoreboard access reduction achieved by SAF should be viewed as an implementation-independent but indirect measurement of potential energy and power savings. Currently and as a part of our ongoing research we are investigating energy and power savings possible by using SAF by using a combination of circuit and microarchitectural simulations.

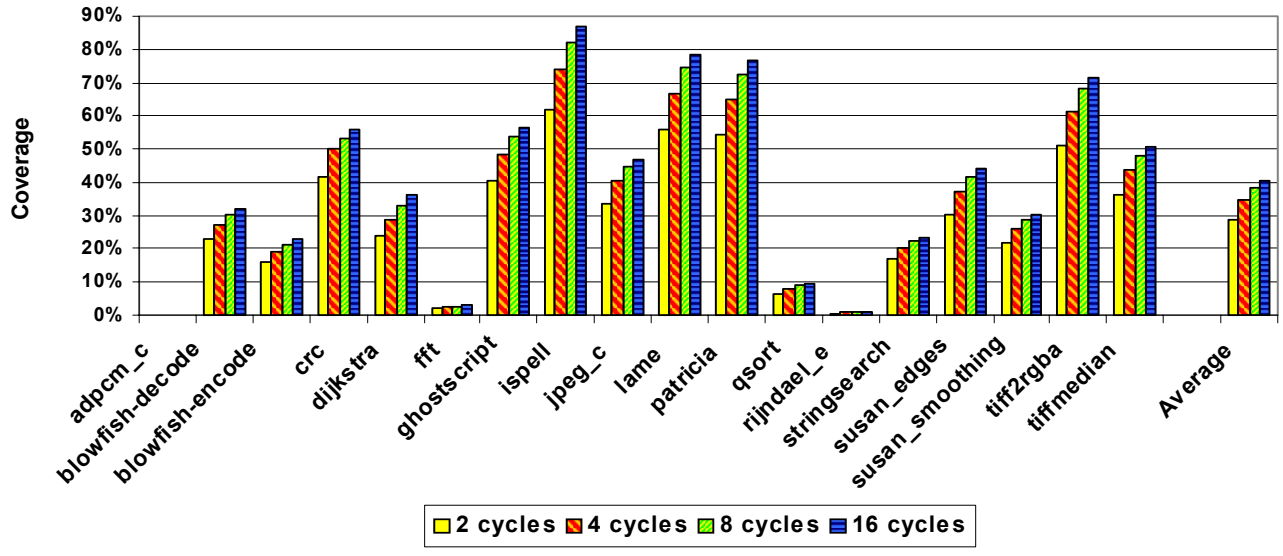

Figure 6: Coverage for gating periods of 2, 4, 8 and 16 cycles.

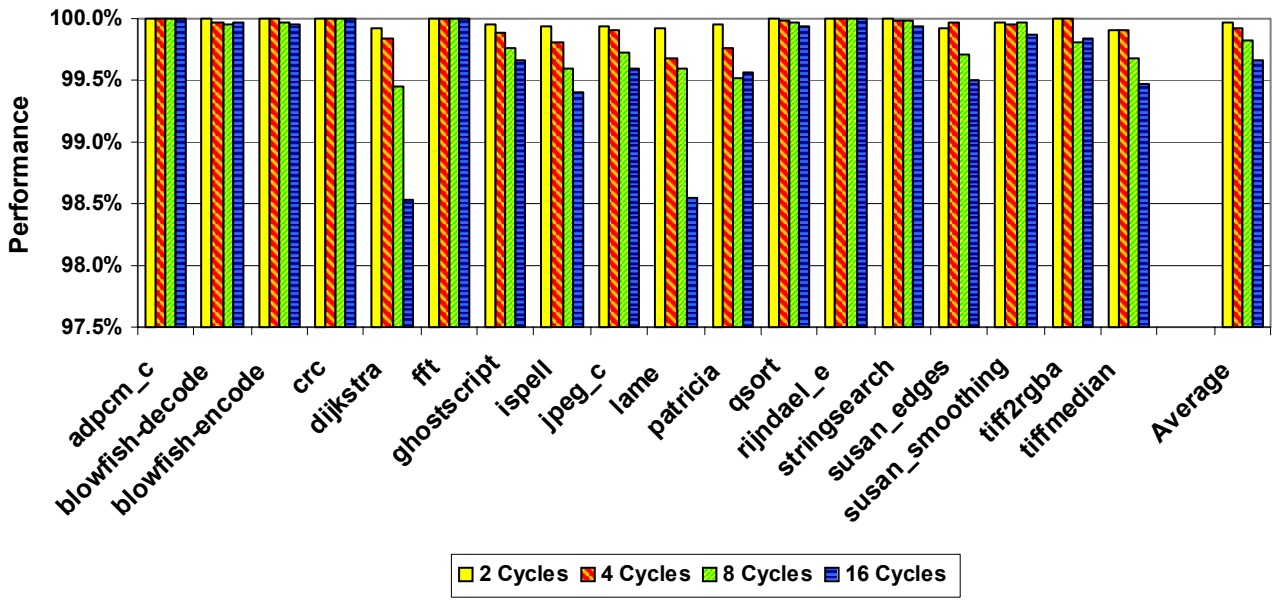

Figure 7: Relative performance for a SAF-enhanced processor for gating periods of 2, 4, 8 and 16 cycles.

We conclude from Figure 6 and Figure 7 that it is possible to achieve higher coverage by increasing the gating period. We also conclude that an 8-cycle gating period provides the best balance between coverage and performance loss.

5. Conclusion

In this work we presented a simple mechanism to eliminate unnecessary scoreboard accesses in Intel's XScale embedded processor.

We showed that a considerable number of scoreboard accesses are unnecessary as they do not contribute to performance. We studied such stalls and observed that they are mostly due to instruction source operand unavailability rather than resource shortage. We showed that instructions that stall the processor have a highly predictable behavior.

We use stall predictability to identify the stall periods and to avoid accessing the scoreboard during such periods. By doing so, on average, we remove about 40% (max: 85%) of extra scoreboard accesses while maintaining performance.

ACKNOWLEDGMENTS

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] *allnetDevices*, http://www.allnetdevices.com/wireless/news/2002/06/24/hps_ipaq.html.

[2] *PC WORLD*, http://www.pcworld.com/news/article0,aid,83923,00.asp.

[3]*Dell Axim X5*, http://www.dell.com/us/en/gen/topics/segtopic_axim.htm.

[4]*INTEGRITY RTOS*, http://www.ghs.com/download/whitepapers/integrity-rtos-xscale.pdf.

[5]S.K. Srinivasan, and M.N. Velev, Formal Verification of an Intel XScale Processor Model with Scoreboarding, Specialized Execution Pipelines, and Imprecise Data-Memory Exceptions, *Formal Methods and Models for Codesign (MEMOCODE '03)*, June 2003, pp. 65-74

[6]Gilberto Contreras, Margaret Martonosi, Jinzhan Peng, Roy Ju, Guei-Yuan Lueh: XTREM: a power simulator for the Intel XScale® core. LCTES 2004: 115-125