Correlation Between State and Control Signals in Out-of-Order Issue Logic

Kenneth Barr Kenneth Conley Serhii Zhak
Advanced VLSI Computer Architecture -- MIT 6.893 Fall 2000
{kbarr, conley, zhak}@mit.edu

Abstract

Current out-of-order control logic is optimized for performance and does not take advantage of energy-saving techniques. Key out-of-order logic structures, including register renaming logic and superscalar issue logic, will be examined for correlation between state and asserted control signals. By exploiting correlation, predictive or memoization techniques could be implemented in the issue logic to eliminate redundant work and increase energy efficiency. We suggest hardware structures to take advantage of the correlation between these signals.

1 Introduction

2 Related Work

In dynamically scheduled superscalar processors, the size (and thus delay and power consumption) of the issue logic depends quadratically on the product of instruction issue width and instruction window size. This delay is mainly due to the associative search required by the issue logic [3].

A recent paper [6] modeled the power consumption of an out-of-order superscalar processor at the level of functional blocks. It found that the renaming table, instruction queue, and reorder buffer, all necessary for extracting instruction level parallelism, are responsible for an average of 53% of the total power consumed by a processor. In addition, each of these three units considered individually consumes more energy than any other part of the processor (e.g., cache, branch predictor, functional units, and I/O). To address the power consumption of the instruction queue, the authors proposed a dynamically resized queue which saved 57% of the power consumed by the queue and reduced the total power consumption of the processor by about 15%.

Hiraki [7] proposed a decoded instruction buffer that memoized control signals for execution with power savings of 40% and Wang [16] proposed a new encoding scheme for storing control signals in microprogrammed control units and claimed 4.8%-16.5% reduction in switching activity.

Our study, motivated by such measurements and proposals, will investigate the possibility of bypassing broadcast and select mechanisms in the issue/renaming logic. Since it is typically designed with performance in mind [5], this logic may consume lots of unnecessary power.

Potential for modified renaming schemes has been shown in [14], but we hope to suggest a scheme that works without compiler assistance. Jourdan [8] proposed physical register reuse in a value-identity detection scheme to improve performance, but we hope to reuse *mappings* of logical registers to physical registers in order to reduce energy, perhaps expanding on Vajapeyam's renamed trace cache [15]. We also plan to consider the potential of hardware memoization in the control unit. This technique has been shown to both reduce power and improve performance of various microprocessor structures [4] [11], avoiding penalties associated with misprediction.

Research has also been done at the circuit level for saving power in control logic [9], but these results are not applicable to our research.

3 Methodology and Results

The Simplescalar microarchitectural simulator [1] has been extended to provide detailed per-instruction statistics that explore trends in register renaming. In addition Simplescalar pipetrace output is analyzed with custom parsers.

Through the use of an Oracle in the issue stage of the pipeline, we attempt to quantify the upper bounds of correlation between processor state and asserted control signals. As we propose modifications to baseline architecture, it will become necessary to consider the effect of misspeculating control signals.

3.1 Pipetracing

Two types of profile statistics were collected using Simplescalar. We termed these "vertical" and "horizontal" statistics, imagining the familiar

Patterson and Hennesey [12] diagram of a pipelined processor.

Profile statistics were collected using the SpecINT95 benchmark suite with scaled-down datasets [table]. Statistics were collected from the perl, ijpeg, and m88ksim benchmarks. Attempts were made to collect statistics from the other benchmarks, but Simplescalar would not provide pipetraces for the requested instruction counts. For both types of profile statistics, cache misses, TLB misses, and branch mispredicts were ignored as we would not want to memoize this worst-case behavior.

Due to the volume of data we had to process, Simplescalar's pipetrace facility had to be re-written to produce binary pipetraces, and the output had to be separated into multiple traces of one million instructions. Furthermore, Simplescalar was also inconsistent in outputting pipetraces for long instruction counts. Summary data from the separate traces was then aggregated in the final statistics. Although the data from the traces were not necessarily independent, as the same PC may have been present in multiple traces, the traces were sufficiently long to reflect what a processor would view before retraining its data set.

3.1.1 Horizontal Statistics

The first type of profile statistic studied the consistency of an individual instruction's path through the stages of the pipeline. We define a consistent trace as one in which a given instruction spends the same amount of time in each stage of the pipeline every time it is dynamically executed. Figure 1 shows that instructions are consistent 65%-80% of the time. This figure grows when restricting the definition of consistency to measure only the time it takes an instruction to move from Fetch to Dispatch stages of the pipeline.

3.1.2 Vertical Statistics

Our vertical statistics examine the correlation between a cycle's fetch program counter (fetch_PC) and the instructions dispatched to execution in that cycle. The highest frequency correlation determines a lower bound on the success rate of an issue logic prediction mechanism. Data from this profile statistic was also used as a data set for prediction logic discussed later section 3.3.3.

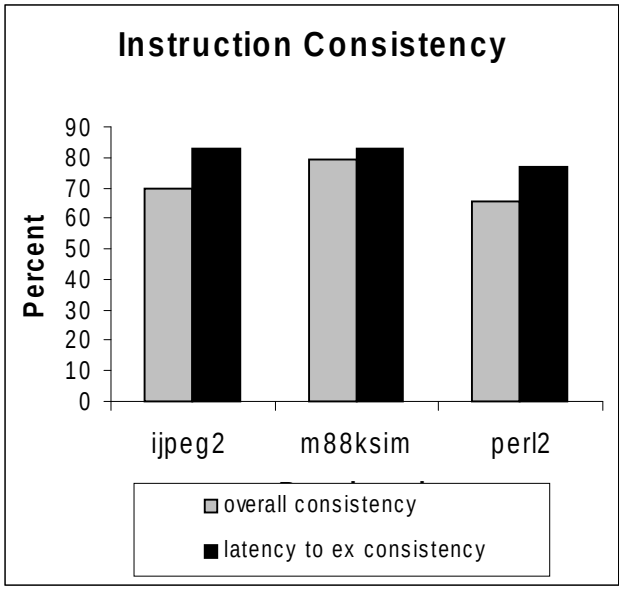

Figure 1

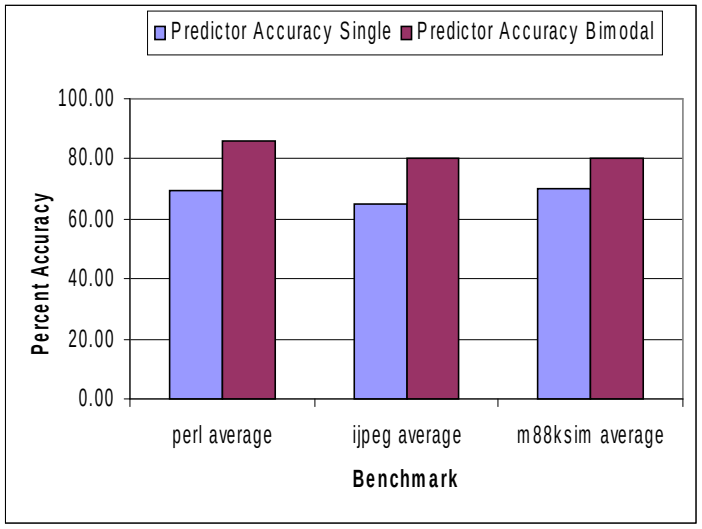

Figure 2

3.2 Renaming

The Simplescalar RUU scheme [13] ties renaming to all aspects of out-of-order execution. Unfortunately, this makes it difficult to quantify the effects of register renaming or to see how renaming effects other aspects of out-of-order control.

We hope to add a decoupled, parameterized renaming structure or modify the existing Simplescalar scheme. Tests with varying rename structures and policies may reveal helpful trends in asserted logic. Intuition suggests that fewer choices of physical registers would lead to higher correlation at the expense of

performance. Perhaps performance would take less of a hit if alternative schemes for determining a mapping were implemented.

Currently we extend Simplescalar to manage a "free list" of physical registers. Also, we have added extensive per-instruction renaming statistics including minimum, maximum, and average length of time a register stays renamed; number of dynamic instances of an instruction; and the number of times an register is renamed to its initial renaming.

Initial results were compiled by examining these statistics after running 500K instructions of a Perl benchmark on a default Simplescalar configuration (with 16 RUU_stations). Figure 3 shows that every instruction needs a physical register for at least 3 cycles, but 80% relinquish their mapping within 7 cycles. This gives hope that a scheme could quickly reassign a mapping.

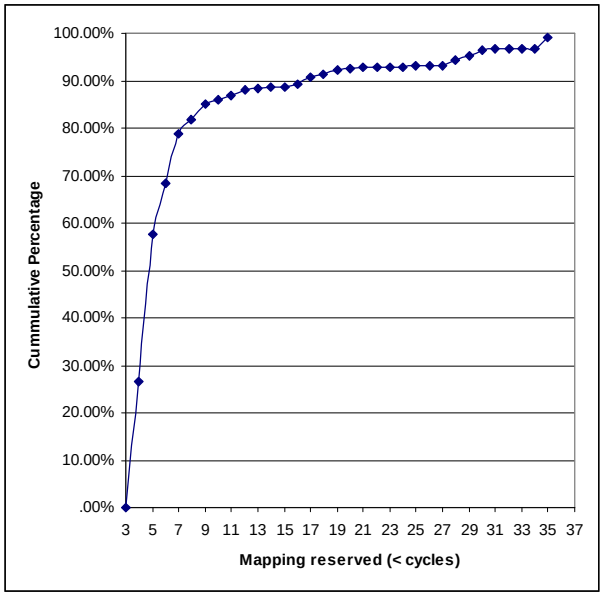

Figure 3

Figure 4 quantifies this hope. We run all 8 spec benchmarks for 50 Million instructions. Just before a logical register is assigned an RUU, we check to see if its original RUU is available. In 59%-77% of the cases, it is.

To finalize the upper bound we will *force* this remapping to occur. We anticipate each benchmark's bar to converge around 60%. As remapping occurs more frequently (the short bar approaches the limit shown by the tall bar), we may be stealing a mapping desired by a future instruction (but as we have shown, mappings are reserved for a very short period of time, so the impact should be minor).

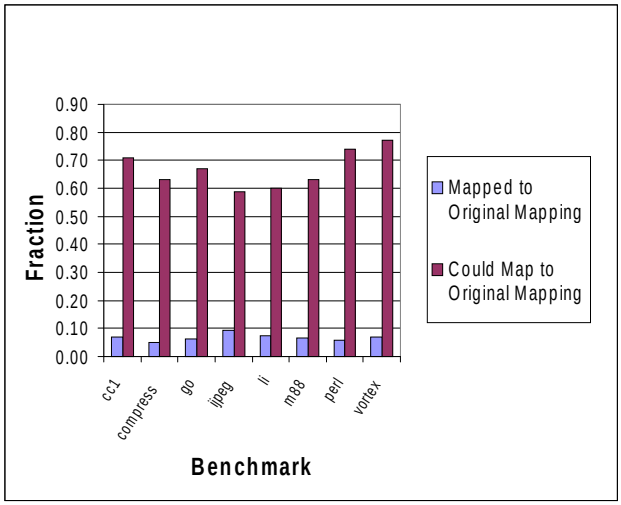

Figure 4

3.3 Issue Oracle

We have to create an "oracle" that acts as ideal issue control prediction logic. The oracle uses the issue statistics collected from previous execution of the instruction stream to predict the instructions to dispatch as a function of the current fetch PC. We hope that the use of the fetch PC to make issue logic predictions will allow the oracle to make predictions along both directions of a branch. The predictions use the most frequent instruction window associated with a given fetch PC. Predictions based on other elements of the processor state, including load-store queue state, functional unit state, and cache misses, have been considered but not yet implemented. The oracle also does not include results from the renaming research.

The oracle is implemented within Simplescalar and makes use of its existing RUU scheme. Using the RUU logic ensures that the oracle does not interfere with the Writeback and Commit stages and also allows the oracle to check data dependencies. This oracle implementation does not correspond well to an actual hardware implementation, as power-saving issue prediction logic would most likely turn off the other OOO logic. Nevertheless, it allows analysis of the success rate of issue prediction logic.

Performance counters have been put in Simplescalar to measure the success and stall rates of the oracle prediction logic. If the oracle cannot issue the predicted instructions because of data dependencies, the oracle stalls the front-end of the processor until

data dependencies are met. If a predicted instruction is quashed due to mis-speculation, the oracle attempts to execute for one more cycle before failing. The oracle prediction logic can fail in several other situations. During startup, if the instruction it wishes to issue was already issued, the oracle will not issue any instructions.

Also, if the predictions fail to account for all of the instructions currently in the dispatch queue, dependencies will quickly force the oracle to fail. In both situations, the oracle is turned off and the normal issue logic is allowed to execute for several cycles.

4 Application[1]

One must keep the Figure 5 (*todo: add cc1 data*) in mind along with Amdahl's Law when considering changes to the architecture based on prediction. When an instruction repeats more than 10 times, it is like to repeat *many* times; these are the instances during which we hope to save power. However, almost half the instructions in our benchmarks only execute once. This represents a potential for disaster if we are not careful to make this common case no worse than it is currently

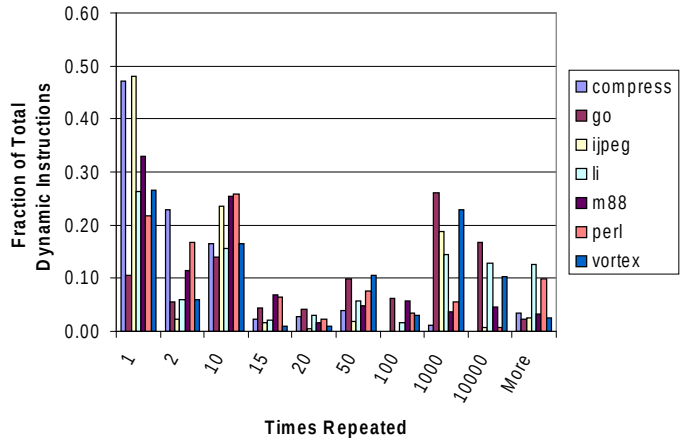

Figure 5

4.1 Suggested hardware structures

5 Conclusions

[1] Or, "whoop-dee-do. What does it all *mean*, Basil?"

References

[1] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. *Technical Report CS-TR-97-1342, University of Wisconsin, Madison*. June 1997.

[2] G. Cai. Architectural Level Power/Performance Optimization and Dynamic Power Estimation. In *Proceedings of the CoolChips tutorial, 32nd Annual International Syumposium on Microarchitecture*, 1999.

[3] R. Canal and A. Gonzalez. A Low-Complexity Issue Logic. In *Proceedings of the 14th International Conference on Supercomputing (ICS-00)*, May 8-11, 2000.

[4] D. Citron, D. Feitelson and L. Rudolph. Accelerating Multi-Media Processing by Implementing Memoing in Multiplication and Division Units. *ASPLOS VIII*, October 1998.

[5] J. Farrell and T. Fischer. Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor. *IEEE Journal of Solid State Circuits*. May 1998.

[6] D. Folegnani and A. Gonzalez. Reducing Power Consumption of the Issue Logic. *ISCA-2000 workshop?*

[7] M. Hiraki, R. Bajwa, H. Kojima, D. Gorny, K. Nitta, A. Shridhar, K. Sasaki, and K. Seki. Stage-Skip Pipeline: A Low Power Processor Architecture Using a Decoded Instruction Buffer. *ISLPED*, 1996.

[8] S. Jourdan, R. Ronen, M. Bekerman, B. Shomar, and A. Yoaz. A novel renaming scheme to exploit value temporal locality through physical register reuse and unification. In *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, 1998.

[9] U. Ko, A. Hill, and P. Balsara. Design Techniques for High-Performance, Energy-Efficient Control Logic. *ISLPED*, 1996.

[10] P. Landman and J. Rabaey. Activity-Sensitive Architectural Power Analysis for the Control Path. *ISLPED*, 1996.

[11] A. Ma, M. Zhang, and K. Asanović. Way Memoization to Reduce Fetch Energy in Instruction Caches. *MIT*, 2000.

[12] Patterson and Hennessy. Computer Architecure; A Quantative Approach. Morgan Kaufman, 1996.

[13] G. Sohi. Instruction Issue Logic for High-Performance, Interruptible, Multiple Functional Unit, Pipelined Computers. *IEEE Transactions on Computer*s, March 1990.

[14] E. Sprangle and Y. Patt. Facilitating superscalar processing via a combined static/dynamic register renaming scheme. In *Proceedings of the 27th annual international symposium on microarchitecture*, 1994.

[15] S. Vajapeyam and T. Mitra. Improving Superscalar Dispatch and Issue by Exploiting Dynamic Code Sequences. In Proceedings of the 24th Annual International Symposium on Computer Architecture. June 1997.

[16] C. Wang and K. Roy. An Activity-Driven Encoding Scheme for Power Optimization in Microprogrammed Control Unit. *IEEE Transactions on VLSI Systems, Vol 7, No. 1*, March 1999.

[new] B. Black and J.P. Shen. Turboscalar: A High Frequency High IPC Microarchitecture. ???

Code

All tables, figures, and performance results included in this paper were generated by code produced by the authors. The renaming code, extensions to the Simplescalar simulator, lives on CAG LCS machines in ~kbarr/6.893/ken. The pipetrace parser is in ~conley/893.