

Effects of Speculation on Performance and Issue Queue Design

Tali Moreshet and R. Iris Bahar

Abstract—Current trends in microprocessor designs indicate increasing pipeline depth in order to keep up with higher clock frequencies and increased architectural complexity. Speculatively issued instructions are particularly sensitive to increases in pipeline depth. In this brief, we use load hit speculation as an example, and evaluate its cost effectiveness as pipeline depth increases. Our results indicate that as pipeline depth increases, speculation is more essential for performance but can drastically alter the utilization of pipeline resources, particularly the issue queue. We propose an alternative, more cost-effective design that takes into consideration the different issue queue utilization demands without degrading overall processor performance.

Index Terms—Architecture, critical-path, high-performance, microprocessors, performance-tradeoffs.

I. INTRODUCTION

Current day superscalar, out-of-order processors achieve high performance by exploiting available instruction level parallelism (ILP) in the program. In order to increase parallelism further, speculative techniques have been implemented. Speculation techniques allow instructions to execute earlier, based on the predictions, before actual information about dependencies or latencies is available. Among these techniques are branch prediction, value and address prediction [1], memory dependence prediction [2], and latency prediction [2], [3]. Specifically, load-hit speculation predicts the access time of load instructions. This allows instructions dependent on a load to issue earlier, without having to wait for actual hit status.

For any form of speculation, some means of recovery is required in case of a misprediction. On a branch misprediction, the processor pipeline is flushed and correct path instructions are fetched from the instruction cache. With other forms of speculation, a misprediction implies the correct instructions were issued but with the wrong data, or in the wrong order. Although these mispredictions may be handled in a similar manner as branch mispredictions, this can be costly in terms of performance [4]. Instead, these instructions can be retained in the issue queue and reissued later with correct data. The issue queue is responsible for tracking data dependencies among all instructions in the queue, determining when instructions are ready for issue, and arbitrating among all the ready instructions. Each cycle, all entries in the issue queue are informed of instructions completing execution so that their dependent instructions may update their ready status and bid for issue. The arbiter must prioritize the bidding instructions and determine which ones to grant permission to issue. The grant signals are then broadcast across the length of the issue queue so that all instructions may update their bid status and granted instructions may drive information to appropriate execution units. This bid/grant process is on the critical path in the design because it limits the rate at which instructions can begin execution.

Manuscript received October 3, 2003; revised March 8, 2004. This work was supported in part by the National Science Foundation under Grant CCR-0311180 and in part by an equipment grant from Sun Microsystems.

The authors are with the Division on Engineering, Brown University, Providence, RI 02906 USA (e-mail: tali@lems.brown.edu; iris@lems.brown.edu; iris_bahar@brown.edu).

Digital Object Identifier 10.1109/TVLSI.2004.834226

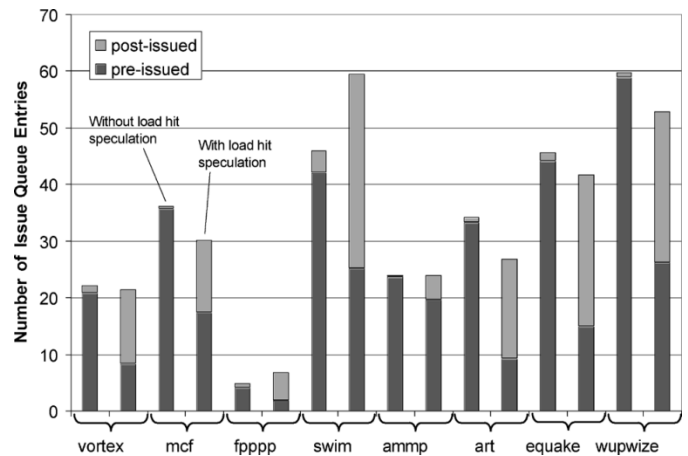


Fig. 1. Number of pending and post-issue instructions for deepest pipeline without (left bar in each pair) and with (right bar in each pair) speculation.

Current trends in microprocessor designs show increasing pipeline depth in order to keep up with higher clock frequencies and increased architectural complexity. By employing speculation techniques, deeper pipelines will increase the number of cycles between the speculation and verification stages. In addition, as pipelines get deeper, a larger percentage of the issue queue will be needed to hold post-issue instructions—most of which will never be needed, assuming low misprediction rates. This limits the queue's effectiveness in exposing available instruction level parallelism.

In this brief, we use load hit speculation as an example speculation technique that uses the issue queue to hold post-issue instructions for misprediction recovery. The Alpha 21 264 [5], as well as the Pentium4 [6] use load hit speculation. Load hit speculation predicts the access time of load instructions in order to allow instructions dependent on a load to issue earlier, without having to wait for actual hit status. One problem associated with speculating the latency of load instructions is the scheduling of instructions dependent on these loads. In general, there is a nonzero delay between the time an instruction is issued to the time it can begin execution. This delay is due to register file access and the moving of data across buses. For instance, the Pentium4 processor takes two cycles to move data across buses and four cycles to access the register file. Early issue of instructions dependent on a load is problematic since load instructions have a nondeterministic latency due to their unknown hit/miss status. The *load resolution loop* is the delay between the issue of a load instruction until its hit/miss information (also referred to as the hit signal) is passed back to the load's dependent instructions. This loop delay increases as the delay between instruction issue and execute increases. If the load misses, all post-issue instructions dependent on the load and issued in its *speculative window* must be reissued or *replayed*.

In general, a larger issue queue will expose more ILP; however, as pipelines get deeper, a larger percentage of the issue queue will hold post-issue instructions that will never be reissued, thereby limiting the utilization of the issue queue. Fig. 1 compares occupancy rates with (first bar for each benchmark) and without (second bar for each benchmark) load hit speculation for our deepest simulated pipeline. The simulation model is described in Section III. Notice that the portion of the issue queue holding post-issue instructions grows from less than 5% to over 55% for floating point benchmarks. Moreover, technology scaling increases the gap between the relative speeds of gates and wires, decreasing the distance that a signal can travel in one clock cycle [7]. Thus, any benefits that come from having a larger issue queue may

TABLE I
BASELINE PROCESSOR CONFIGURATION

Parameter	Configuration
Inst. Window	64-entry LSQ, 256-entry ROB 64-entry ISQ
Machine Width	8-wide fetch, issue, commit
Number of FUs	8 Int add (1), 2 Int mult/div (3/20)
Latency in ()	4 Load/Store (3), 8 FP add (2) 2 FP mult/div/sqrt (4/12/24)
L1 Icache	16KB 2-way; 32B line; 1 cycle
L1 Dcache	64KB 2-way; 32B line; 3 cycle
L2 Cache	128KB 4-way; 64B line; 12 cycle
Memory	100-cycle latency; 16 bit-wide
issue→exec. latency	variable from 1 to 9 cycles
feedback delay	variable from 1 to 9 cycles

come at the expense of a slower clock rate for the scheduler logic [8]. As an alternative, we propose a more cost-effective *Dual Issue Queue* structure that separates the post-issue instructions from the rest of the pending preissued instructions in the queue. Each subqueue is smaller, allowing a faster clock and better overall utilization of the issue queue.

II. RELATED WORK

Several works have addressed reducing the complexity of the issue queue structure [9]–[11], [3]. Of the more relevant works, Raasch *et al.* [10] proposed a dynamic scheme where the issue queue is divided into segments, according to the expected time the instructions will be ready for issue. Only instructions residing in the lowest segment can be issued. When a load misses, its dependency chains are stalled in the current issue queue segments. Lebeck *et al.* [11] proposed an alternative scheme for dealing with instructions dependent on loads that miss. Following a load miss, the chain of its dependent instructions is moved from the issue window to a *waiting instruction buffer* (WIB) and moved back to the issue queue once the load completes. This scheme may require a single instruction to be moved from the issue queue to the WIB several times before it actually issues. In addition, the WIB must be the size of the reorder buffer (active list) in order to accommodate all possible instructions.

While the works of [10] and [11] show promising results, they do not include any analysis on the effects of deeper pipelines or speculation in the study. Other works deal specifically with deeper pipeline issues. For instance, the work of [4] focused on redesigning the register file to better accommodate for deeper pipelines. The limits of deep pipelines were addressed in [12] and [13], but did not specifically deal with the effects of misspeculation, other than for branch mispredictions. The impact of load hit missprediction on performance has been specifically addressed in [3]. In that work, Morancho *et al.* evaluated the recovery mechanism of load hit missprediction and proposed a structure called the *recovery buffer* to store issued instructions dependent on misspredicted loads. We provide a general approach that can be used with multiple types of speculation with various verification delays simultaneously.

III. LOAD HIT SPECULATION MODEL

We used a modified version of the SIMPLESCALAR [14] tool suite for our study. In particular, we implement a separate reorder buffer (ROB) and issue queue (ISQ). We assume that issued instructions are kept in the issue queue until it is known that they will not need to be reissued. Table I shows the complete configuration of the processor.

Other modifications to SIMPLESCALAR include a cycle delay between the issue and added execute stage that can be varied from 1–9 cycles, in order to increase the pipeline depth specifically between the issue and writeback stages. Also, a variable wire latency was added for the feedback of hit/miss information from the cache to the consuming,

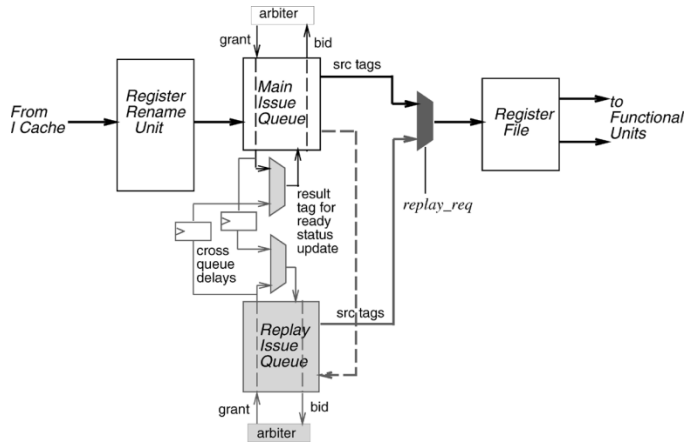


Fig. 2. Proposed dual issue queue scheme.

post-issue instructions. Consuming instructions may be issued such that the producers' results will be ready by the time execution begins. Load instructions are speculated to be ready after the time it takes to access the level 1 data cache. Once it is discovered that a load missed in the cache, instructions that were issued in a misspredicted load's speculative window and are dependent, directly or indirectly, on the load are replayed. Therefore, issue queue entries are released only when instructions reach writeback, for all instructions. Simulations are executed on a subset of the SPEC95 and SPEC2000 integer and floating point benchmarks [15], [16] (a total of 32 benchmarks). DL1 miss rates for these benchmarks varied from 0% to 23%.

IV. DUAL ISSUE QUEUE SCHEME

According to Palacharla *et al.* [9], the issue queue logic and data bypasses (i.e., wire delay) are likely to be the key limiting factors of all pipeline delays. In order to issue dependent instructions in consecutive cycles, the issue queue bid/grant (wakeup/select) logic must be accomplished in a single cycle. This makes the issue queue latency the processor critical path which determines the clock frequency. A larger issue queue comes at the expense of a slower clock [8].

Our goals are to reduce the size of the main issue queue without hurting overall performance, and leave a larger fraction of the issue queue for preissued instruction, thus exposing more ILP. Our new issue queue design consists of two parts: the main issue queue (MIQ), and the replay issue queue (RIQ). The RIQ is effectively a temporary place holder for issued instructions that may need to be replayed. The MIQ and RIQ are similar in structure, minimizing the design effort for the two queues. The pipeline for our new issue queue scheme is shown in Fig. 2. The added logic is shown in gray. Initially, dispatched instructions are placed in the MIQ. The MIQ is searched for ready instructions, and these are given a chance to bid for an issue slot. Instructions that have already issued and updated the ready status of their dependent instructions are then moved from the MIQ to the RIQ if there are empty slots available. After instructions are issued, they are not likely to be accessed again from the issue queue, since most load instructions hit in the cache, and their dependent instructions will not be reissued.

During the issue stage, instructions may be selected for issue either from the RIQ or the MIQ, but not both. To facilitate this, instructions from both queues are allowed to update their ready status every cycle, but only one queue is allowed to bid for issue resources at a time. In our simulation model, the processor gives priority in selecting instructions to be issued to the RIQ; however, the only time instructions in the RIQ can bid and be granted an issue slot is after a load hit missprediction. Since the RIQ holds ready instructions only after a missed load

reaches the writeback stage, we do not need to check the RIQ for ready instructions every cycle.

Once an instruction is granted an issue slot, the ready status of its dependent instructions needs to be updated. If the issued instruction comes from the RIQ, its dependent instructions may reside in either the RIQ or MIQ. Since the RIQ may be physically located at a distance from the MIQ, broadcasting the grant signals from the RIQ to the MIQ may incur a longer cross-queue latency, preventing these dependent instructions in the MIQ from bidding for an issue slot in the following cycle. If the issued instruction resides in the MIQ, its dependent instructions will most likely reside in the MIQ as well. These updates are part of the bid/grant critical path loop and will not be delayed an extra cycle. However, if dependent instructions from a load miss reside in both the RIQ and MIQ (due to space limitations in the RIQ), then broadcasting the grant signals from the MIQ to the RIQ will incur a longer cross-queue latency as well.

As illustrated in Fig. 2, updating dependent instructions in the MIQ requires adding a multiplexor along the bid/grant loop since the issued instruction can come from either the MIQ or RIQ (see the two lightly shaded multiplexors in the figure). Although this multiplexor does not come for free, the delay impact may be minimized by incorporating the MUX function into the latch used to control the driving of the result tags. Furthermore, the select lines for the MUX are not on the critical path since they are set according to the actions occurring in the previous cycle. The extra MUX added to the source tag output bus path (the dark shaded multiplexer in Fig. 2) also adds extra delay; however, combining it with a pipeline latch will make this extra logic unlikely to have a significant effect on the issue to execute latency. Notice that our approach does not introduce any pipeline bubbles.

It is useful to compare our dual issue queue design to the recovery buffer scheme proposed in [3] which was designed to address similar issues. They proposed removing post issued instructions from the main issue and placing them in a *recovery buffer*. The recovery buffer consists of two levels of instruction buffers with shift registers and two dependency matrices. The sizes of the buffers are determined assuming a specific verification delay. Although issue logic in the recovery buffer is simplified, the scheme does not scale well with longer delays. For example, assuming a nine cycle verification delay and an 8-wide issue processor, the scheme in [3] would need 72 buffer entries for the first level, and 72 buffer entries for *each* mispredicted load in the second level buffer in order to accommodate for the worst case scenario. This is a very inefficient way to use registers since it is unlikely that all 72 instruction slots would hold valid instructions dependent on a single load. In addition, this scheme is designed for a specific verification delay, which would make it unsuitable to handling a combination of speculation techniques. For instance, load hit speculation and load-store dependency prediction are often both implemented in high-end processors, yet their verification delays may be quite different. The scheme of [3] could not handle both types simultaneously. Notice that our scheme does not have these limitations.

V. RESULTS

To evaluate the effectiveness of our scheme, one must consider impact on IPC as well as clock speed. Assuming the bid/grant loop is on the processor's critical path, the processor clock speed is determined by the main issue queue. The bold bars in Fig. 3 show the change in IPC using either a dual issue queue scheme (with various sizes of MIQ and RIQ) or a standard unified issue queue. Comparisons are made relative to a standard 64-entry issue queue. If IPC alone is considered, we see that, in general, IPC improves when the issue queue size is increased. Using both MIQ and RIQ, the number of preissued instructions is allowed to increase with pipeline depth, and as a result so does the

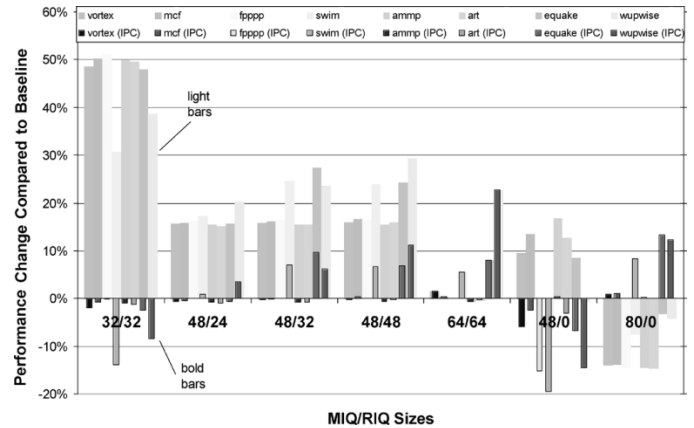


Fig. 3. Relative IPC (and performance) of the dual issue queue scheme with variable MIQ and RIQ sizes vs. a 64-entry issue queue (and one with a slower clock). Issue to execute latency is nine cycles.

exposure of available ILP. The figure shows that the MIQ can be substantially smaller than 64 entries in order to obtain comparable IPC to that of the baseline. Although smaller, the MIQ issues more than 99% of all instructions, on average. A 48-entry MIQ and a 32-entry RIQ configuration is sufficient in terms of IPC, and increasing the RIQ size to 48 entries gives an additional improvement. The better performing benchmarks are floating point benchmarks, characterized by having more parallel streams of dependent instructions. They also tend to benefit the most from having a larger issue queue. Benchmarks with large DL1 miss rates did not perform as well, and may benefit from a load hit predictor such as [17] to avoid some of the misprediction penalty.

We have shown that a smaller main issue queue can be used to achieve comparable IPC, however, performance improvements may be even greater if we also consider the effect on the clock. By extrapolating from results in [8] we estimate that a standard 64-entry issue queue will have a 34% and 14% slower clock than a 32-entry and a 48-entry issue queue, respectively, so clock rate becomes the dominating factor over relative IPC. The lighter, wider bars in Fig. 3 show the relative performance of the dual issue queue scheme when taking the slower clock into account. Now the 32-entry MIQ, 32-entry RIQ scheme (32/32) seems more attractive in terms of performance. However, if the issue queue is small enough, then its delay will not determine the processor clock rate, since some other part of the pipeline will be on the critical path. When compared to a slower clock 64-entry standard issue queue, our 48-entry MIQ, 32-entry RIQ scheme has a performance advantage of over 15% across all benchmarks.

The last two clusters of bars in Fig. 3 show the performance of a unified 48-entry and 80-entry issue queue. It can be seen that the IPC degradation of the 48-entry issue queue is so large, that in some cases even its faster clock does not compensate. For the 80-entry issue queue, on the other hand, although the IPC increase is greater than that of a 48/32 dual issue queue, the slower clock of the larger queue hides these increases, and the relative performance suffers.

VI. CONCLUSION

Speculation is an important method for increasing performance by enabling more instruction-level parallelism. However, because post-issue instructions must remain in the issue queue longer to allow for quick recovery from a misprediction, as pipeline depth increases, speculation increases the percentage of post-issue instructions in the issue queue, limiting ILP. We propose a new issue queue scheme that addresses the utilization concerns without compromising performance.

Our dual issue queue dedicates a separate structure to post-issue instructions, thus, allowing more ILP to be exposed even with a smaller main issue queue. In addition, a smaller main issue queue can reduce the pressure on the bid/grant critical path, allowing the processor to run with a faster clock. The dual issue queue does not depend on the verification latency and is flexible for usage with other forms of speculation such as load address, value or memory dependence prediction, and scales well with deepening pipelines.

REFERENCES

- [1] B. Calder and G. Reinman, "A comparative survey of load speculation architectures," *J. Instruction-Level Parallelism*, vol. 2, May 2000.
- [2] A. Yoaz, M. Erez, R. Ronen, and S. Jourdan, "Speculation techniques for improving load related instruction scheduling," in *Proc. Int. Symp. Computer Architecture*, May 1999.
- [3] E. Moranco, J. M. Llaberia, and A. Olive, "Recovery mechanism for latency misprediction," in *Proc. Int. Conf. Parallel Architectures Compilation Techniques*, Barcelona, Spain, Sept. 2001.
- [4] E. Borch, E. Tune, S. Manne, and J. Emer, "Loose loops sink chips," in *Proc. Int. Symp. High-Performance Computer Architecture*, Boston, MA, Feb. 2002, pp. 299–310.
- [5] *Alpha 21 264 Microprocessor Hardware Reference Manual*, Compaq Computer Corporation, 1999.
- [6] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The microarchitecture of the pentium 4 processor," *Intel Technol. J.*, vol. Q1, 2001.
- [7] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," in *Proc. Int. Symp. on Comput. Architecture*, Vancouver, BC, Canada, June 2000, pp. 248–259.
- [8] D. Ernst and T. Austin, "Efficient dynamic scheduling through tag elimination," in *Proc. Int. Symp. Computer Architecture*, Anchorage, AK, May 2002, pp. 37–45.
- [9] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-effective super-scalar processors," in *Proc. Int. Symp. Computer Architecture*, Denver, CO, June 1997, pp. 206–218.
- [10] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, "A scalable instruction queue design using dependence chains," in *Proc. Int. Symp. Computer Architecture*, Anchorage, AK, May 2002, pp. 318–329.
- [11] A. R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg, "A large, fast instruction window for tolerating cache misses," in *Proc. International Symposium on Computer Architecture*, Anchorage, AK, May 2002, pp. 59–66.
- [12] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," in *Proc. Int. Symp. Computer Architecture*, Anchorage, AK, May 2002, pp. 25–34.
- [13] A. Hartstein and T. R. Puzak, "The optimum pipeline depth for a micro-processor," in *Proc. Int. Symp. Computer Architecture*, Anchorage, AK, May 2002, pp. 7–13.
- [14] D. Burger and T. Austin, "The SimpleScalar Tool Set," Technical Report, Univ. Wisconsin, WI, 3.0 ed., 1999.
- [15] Spec; Standard Performance Evaluation Corporation (1996). [Online]. Available: <http://www.specbench.org>
- [16] J. L. Henning, "SPEC CPU2000: Measuring cpu performance in the new millennium," *IEEE Computer*, pp. 28–35, July 2000.
- [17] J. Peir, S. Lai, S. Lu, J. Stark, and K. Lai, "Bloom filtering cache misses for accurate data speculation and prefetching," in *Proc. Int. Conf. Supercomputing*, New York, NY, June 2002.

Correction to "A Digitally Programmable Delay Element: Design and Analysis"

M. Maymandi-Nejad and M. Sachdev

In [1], there appeared a typing error in (10). The correct equation should be as follows:

$$t_d = \frac{A_1}{(V_g - V_1)^2} \quad (10)$$

REFERENCES

- [1] M. Maymandi-Nejad and M. Sachdev, "A digitally programmable delay element: Design and analysis," *IEEE Trans. VLSI Syst.*, vol. 11, pp. 871–878, Oct. 2003.

Manuscript received April 2, 2004.

The authors are with the Electrical and Computer Engineering Department, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: maymandi@vlsi.uwaterloo.ca).

Digital Object Identifier 10.1109/TVLSI.2004.837019