A Mechanism for Verifying Data Speculation^{*}

Enric Moranco, José María Llabería, and Àngel Olivé

Computer Architecture Department, Universitat Politècnica de Catalunya, Spain
{enricm, llaberia, angel}@ac.upc.es

Abstract. High-performance processors use data-speculation to reduce the execution time of programs. Data-speculation depends on some kind of prediction, and allows the speculative execution of a chain of dependent instructions. On a misprediction, a recovery mechanism must re-issue the speculatively issued instructions. Some recovery mechanisms rely on keeping each instruction in the Issue Queue (IQ) until it is known that the instruction has used correct data. However, their authors either assume that the IQ and the Reorder Buffer (ROB) are unified, or do not detail how the instructions are removed from the IQ before reaching the ROB head entry.

We propose the Verification Issue Queue (VIQ), a mechanism fed with a verification flow graph; the VIQ decides if an instruction can either be removed from the IQ or must be re-issued; the VIQ also allows decoupling the IQ from the ROB. Our evaluations, in the context of load address prediction, show that the verification mechanism is crucial for exploiting the performance potential of data speculation, and that the kind of graph used by the VIQ has a performance impact similar to reducing first-level cache latency by one cycle.

1 Introduction

Data-speculative processors must use a recovery mechanism to guarantee that, for each misprediction, all the speculatively issued instructions dependent on the misprediction will be re-issued. The simplest recovery mechanism is similar to the one used in branch mispredictions: flushing the pipeline and re-fetching the instructions from the instruction cache. However, this mechanism is not feasible in current high-performance processors due to the large performance cost of re-fetching the instructions [1].

Other recovery mechanisms avoid *re-fetching* by keeping each issued instruction on a processor structure until it is known that the instruction has used correct operands. Several authors [2, 3] propose the use of specific structures. On the other hand, some processors [4, 5] keep the issued instructions in the Issue Queue until they are no longer speculative; on a misprediction, the instructions are re-issued when their operands become ready. In this work, we keep the speculatively issued instructions in the IQ.

^{*} This work was supported by the spanish government (grant CICYT TIC2001-0995-C02-01), and the CEPBA (European Center for Parallelism of Barcelona).

An implicit approach [6, 7] to verify an instruction is based on re-issuing the instruction after observing a new value for any of its operands, and checking when the instruction reaches the ROB head entry (i.e., it becomes the oldest instruction in flight). As the operands of the oldest instruction are known to be non speculative, then when the state of the oldest instruction is *completed* the instruction gets verified and it can be retired. However, this approach does not scale well because it requires the number of IQ entries to be equal to the number of ROB entries. Sazeides [7] presents a verification mechanism that uses the data flow graph also as a verification flow graph. However, he also unifies IQ and ROB and assumes that, on one cycle, verifications traverse all the verification flow graph, regardless of the number of graph levels.

This work presents the Verification Issue Queue (VIQ), a mechanism that decides if an instruction can be removed from IQ or must be re-issued; it also allows decoupling IQ from ROB. The VIQ is fed with a verification flow graph that can be built in several ways. We evaluate the VIQ in the context of load address prediction. We conclude that VIQ significantly outperforms the implicit mechanism, and that the verification flow graph used by VIQ has a performance impact similar to reducing first-level cache latency by one cycle. This paper is organized as follows: Section 2 describes the IQ used in this work, Section 3 presents the VIQ, Section 4 presents an evaluation of our proposal, and Section 5 concludes the work.

2 Issue Queue (IQ)

In dynamically-scheduled processors, instructions wait in the IQ for the availability of operands and functional units. To issue instructions out-of-order to the functional units, the IQ has two components: a) *wakeup logic* and b) *select logic*. The *wakeup logic* keeps monitoring the dependencies among the instructions in the IQ and, when the operands of a queued instruction become available, this logic will mark the instruction as ready. The *select logic* selects which ready instructions will be issued to the functional units on the next cycle. When latency of the selected instructions is elapsed, dependent instructions are woken up.

The IQ may be implemented using CAM cells [8] or bit-matrices [9]. In this study, the IQ is similar to the one described in [10]; it uses a **dependence matrix** for tracking dependencies among instructions. The matrix has as many rows as the number of instructions analysed simultaneously for scheduling, and as many columns as the number of physical registers (registers for short). Each column is connected to a latency counter that starts counting-down when the related instruction is issued. On each crosspoint, a logical circuit determines if there is an unresolved dependence between the instruction and the register. For each row, the outputs of these logical circuits are used to compute a *ready bit* that indicates if the instruction is ready to be selected by the *select logic*. *Ready bits* are evaluated every cycle.

On processors that do not perform data speculation, instructions can be removed from IQ as soon as they are issued. However, on data-speculative proces-

sors, the recovery mechanism influences on the removal policy. In this work, we assume that the recovery mechanism keeps the speculatively issued instructions in the IQ until verifying them.

Re-issue Mechanism. While an issued instruction waits in the IQ for its verification, it is made non-visible to the *select logic*. This is handled by using a *no-request bit* in each matrix row; this bit it is set when the instruction is issued.

When an instruction must be reissued because it has used a misspeculated operand, the instruction is made visible again to the *select logic*. Moreover, its destination-register state is set to non-available; this change delays the issuing of its directly dependent instructions until the instruction is re-issued and a new data value is computed.

Two control circuits perform previous operations: the *removal circuit* and the *register-scoreboard circuit*. Fig. 1 shows the interface between the circuits and other IQ elements. Every cycle, the *removal circuit* is notified of the instructions that must be removed/ re-issued, and the *register-scoreboard circuit* is notified of the misspeculated instructions. The *removal circuit* removes verified instructions from the IQ, and makes visible again the instructions that must be re-issued. The *register-scoreboard circuit* activates latency counters (for each issued instruction, and when data of the predicted load instruction becomes available) or unsets columns (for each misspeculated instruction). As *ready bits* are re-evaluated every cycle, misspeculated instructions will re-evaluate their ready bits, waiting for operand availability. Furthermore, all instructions directly dependent on the misspeculated instructions will be slept.

Fig. 1. Structure of the IQ.

3 Verification Issue Queue (VIQ)

In this section we explain how the VIQ decides whether an instruction can be removed from the IQ or must be re-issued. We will apply the VIQ in the context of address prediction. The effective addresses computed by load instructions will

be predicted by an address predictor; this will allow the speculative execution of load instructions and their dependent instructions.

The VIQ works as follows. When the address-check result of a predicted load instruction is known, the VIQ initiates the verification/invalidation of the chain of speculatively issued instructions dependent on the load. The VIQ first verifies/invalidates issued instructions directly dependent on the load instruction. These verified/invalidated instructions are used, in next cycle, to verify/invalidate issued instructions that directly depend on them. Thus, each cycle, issued predicted load instructions and verified/invalidated instructions are used to verify/invalidate issued instructions.

In Fig. 2 we present an example of the propagation of verifications and invalidations (we only show significant pipeline stages). Assume that the effective address of load instruction a) has been predicted, memory has been accessed, and instructions b) and c) have been executed speculatively. On cycle 1, instruction a) issues (IQ), then reads registers (R) and computes its effective address (@). Concurrently, the address prediction is checked and turns out to be wrong; after that, VIQ propagates re-issue signals to the dependent instructions: on cycles 4 and 5, VIQ signals IQ that instructions b) and c) must re-issue, and they re-issue on cycles 5 and 6 respectively. Instruction a) access memory non speculatively (m), VIQ propagates its verification and notifies IQ that the instructions can be removed from IQ. In section 3.2, we will show that the VIQ can be also used to perform a faster propagation of the verifications/invalidations. Now, we describe the components of VIQ.

Fig. 2. Example of verification propagation using the VIQ.

3.1 Basic Structure

VIQ: Elements and Input/Output Signals. The issued instructions wait in the VIQ until it is known whether its source operands have become valid or misspeculated. The VIQ has two components (Fig. 3): a) **Verify/Invalidate logic** and b) **Decision logic**. The **Verify/Invalidate logic** monitors if the source operands of the instructions are valid or misspeculated. When all the operands of an instruction become valid, or one of them is misspeculated, this logic marks the instruction as verified or invalidated respectively.

For each issued instruction, the **Decision logic** determines if: a) the instruction can be removed from the IQ, b) the instruction must be re-issued or c) for predicted load instructions, the data value has been misspeculated and the speculatively issued dependent instructions must be re-issued. The Decision logic considers: a) the instruction has been issued, b) verify/invalidate signals, c)

Fig. 3. Components of the VIQ.

instruction type (predicted load, other instruction), d) address-check result, and e) memorization elements related to the instruction (one bit per instruction). Decision-logic outputs are used in next cycle to propagate verify/invalidate information to directly dependent (in the verification-flow graph) issued instructions. In addition, these outputs are sent to the IQ in order to remove/re-issue instructions.

Structure of the Verify/Invalidate Logic. The core of this logic is a matrix structure (Fig. 4) with the same number of rows and columns as the dependence matrix of IQ. This matrix monitors if the source operands of the instructions are valid or misspeculated.

Fig. 4. Matrix structure of the VIQ.

The columns are wires that cross all rows and each row includes a bit for each column. Each column marks the register state (speculative, valid) and if the data value has been misspeculated. The state of a register is speculative until the Decision logic indicates that the producer instruction, which computes the data value, can be removed. The misspeculated signal is set only for one cycle, when Decision logic indicates that the instructions that have consumed misspeculated data value must be re-issued. Instructions are inserted in both the IQ and the VIQ at the same time. When an instruction is inserted in a row, the bits related to its source operands are set; this information is built at the Register Rename stage and inserted in VIQ after this stage.

Each crosspoint of the matrix contains a logical circuit that determines whether the source operand is valid or misspeculated (Fig. 4). For each row, the outputs of the circuits are used to compute the verify/invalidate signals. The verify signal is activated when all source operands of the instruction are valid. The invalidate signal is activated if any source operand of the instruction is misspeculated. Both signals are evaluated every cycle.

Predicted Load Instructions. A predicted load instruction establishes if its destination register is valid or misspeculated by using two pieces of information related to the same instance of the dynamic load instruction: a) address-check result and b) verify signal of the load instruction (its source operand is valid). Then, predicted load instructions must be tagged in order to take into account both pieces of information before the Decision Logic takes a decision. For this, we use the instruction type bit (predicted load instruction, other instruction). However, for incorrect address-check results, the chain of dependent instructions will be re-issued immediately since the computed address is used for initiating a new memory access after the address-check stage, and this chain of instructions have used misspeculated data values.

Decision Logic. Each entry of the matrix structure has a valid bit which indicates if the instruction has been issued and it is waiting for the verification/invalidation of its source operands. This bit is set each time the instruction is issued and it is unset (non-valid) when Decision logic decides that the instruction must be re-issued.

For each instruction, the decision logic uses verify/invalidate signals of the matrix structure, valid bits of the entries, address-check results, instruction type and memorization elements to generate remove, re-issue and misspeculate signals.

For a predicted load instruction, address-check result and verify signal of the source operand must be paired before deciding if data value is valid. Because both pieces of information may be generated at different time, a memorization element in Decision logic is needed. After pairing both pieces of information, on a correct address-check result, the remove signal is set. Otherwise, incorrect-address check result, the misspeculate signal is set only for one cycle, and in the next cycle the remove signal is set. For the other instruction type, verify/invalidate signals drive directly remove and re-issue&misspeculate signals respectively.

Fig. 5 shows the interface between VIQ components. At every cycle, the *removal circuit* and the *Register-Verify circuit* are notified of the removed/re-issued and the re-issued/misspeculated instructions respectively. The *removal circuit* removes an instruction when then remove signal is activated. It also sets the entry as non-valid when the re-issue signal is activated. The *register-verify circuit* sets the destination register state to valid on a remove signal, and activates the misspeculate column on a misspeculate signal.

Fig. 5. Structure of the VIQ.

3.2 Verification Flow Graphs

The VIQ uses a verification flow graph to propagate the verifications/invalidations through the instructions. We can build several kinds of verification-flow graphs. Depending on the graph used, verifications/invalidations will be propagated faster to the instructions. However, the VIQ structure is independent of the verification flow graph used, the only difference is the dependence information stored in the matrix structure of the VIQ. We have considered two kinds of verification flow graphs:

Serial Verification Flow Graph. In this case, the verification flow graph is identical to the data flow graph, but the latency of all instructions is one cycle.

Enhanced Verification Flow Graph. In this case, we use a new verification-flow graph with fewer levels than the serial verification-flow graph. In this graph, every instruction is connected directly to the predicted load instructions it depends on (directly or indirectly). The goal of this design is to reduce the number of cycles needed to verify/invalidate instructions that depend on a predicted load instruction: only one step will be required to verify/invalidate source operands of instructions directly/indirectly dependent on a predicted load.

To build the rows of the matrix structure we use a table named Collapsed Graph Table (CGT) with as many entries as architectural registers, and each entry has a bit vector with as many bits as physical registers. The CGT is indexed by the architectural-register identifier. It is updated only with information known in the rename stage, and is also read in this stage to build row information. Predicted load instructions set the bit that identifies its destination register in the CGT entry of its destination architectural register. Any other instruction reads the CGT entries related to its source architectural registers, applies the OR function to the bit vectors, and stores the resultant bit vector in the CGT entry of its destination architectural register.

To insert an instruction in the VIQ, each instruction reads the CGT entries associated to its source architectural registers, applies the OR-function to the obtained bit vectors, and stores the result in its assigned entry of the matrix structure of the VIQ.

4 Evaluation

4.1 Evaluation Environment

To evaluate our proposal we derive a simulator from the SimpleScalar 3.0 (Alpha ISA) cycle-by-cycle simulator [11]. We simulate a 4-way processor (like 21264: same pipeline, branch prediction and functional units) with a 128-entry ROB, and an 8-way processor (scaling by 2 resources of the previous one) with a 256-entry ROB. Memory hierarchy has separated first-level caches (64Kb, direct mapped, variable latency), unified second-level cache (1Mb, dm, 12 cycles), and 80-cycle latency main memory.

To predict effective addresses, we use a hybrid address predictor composed of a stride and a context component. On average, this predictor predicts correctly about 83% of the effective addresses and its accuracy is about 96%. In our data-speculative processors, branch resolutions are resolved non-speculatively. Our proposals are evaluated on the SPEC95-INT benchmarks, and we present average results. Detailed information on simulation intervals and individual results can be found in [12].

4.2 Results

First, we evaluate the effectiveness of the implicit verification versus the use of the VIQ. We compare the performance of the baseline processor (without address prediction) and two data-speculative processors: one uses the implicit verification and the other uses the VIQ with the serial verification. In these evaluations, we assume that the number of IQ entries is equal to the number of ROB entries. We observe that the effectiveness of address prediction depends heavily on the verification mechanism. In fact, in most cases, the performance of the processor with implicit verification is worse than the performance of the baseline processor. Comparing both data-speculative processors, the use of the VIQ improves the performance about 5% (4-way processors) and from 13% to 17% (8-way processors). The instructions that take advantage of the faster verification process are the mispredicted branch instructions, because they can be resolved before reaching the ROB head-entry. We conclude that the verification mechanism is critical for dealing with mispredicted branch instructions.

Decoupling IQ from ROB. After this, we apply the VIQ both to remove the instructions from the IQ and to perform the verification process. Each instruction is removed from the IQ one cycle after the VIQ notifies the verification. Fig. 6.a shows the impact of the IQ size on the performance of a data-speculative processor with serial verification.

We can observe that the IQ size needed to saturate the performance is larger in data-speculative processors than in baseline processors. This is due to the fact that data-speculative processors use some IQ entries to maintain the speculatively issued instructions until they become non speculative. Then, these entries

Fig. 6. a) IPC versus IQ size on baseline processors and data-speculative processors with serial verification, b) IPC versus IQ size on 8-way data speculative processors.

are not used by the scheduler to look ahead for independent non-issued instructions. For instance, in 4-way processors, using the 25-entry IQ, the performance is below saturation from 1.6% (2-cycle latency) to 1.9% (4-cycle latency). In 8-way processors, using the 50-entry IQ, the performance is below saturation from 2% to 2.3%.

Enhanced Verification Flow Graph. Finally, we use the enhanced verification flow graph to remove the instructions from the IQ. The use of the enhanced graph reduces the pressure on the IQ with respect to the use of the serial graph, since the enhanced mechanism frees some IQ entries earlier than the serial mechanism. Freeing IQ entries as soon as possible is useful because it may allow the insertion of younger instructions into the IQ to be advanced. Fig. 6.b shows the impact of the IQ size on the performance of 8-way address-speculative processors, as well as both the serial and the enhanced verification mechanisms. Our results show the performance effect of the enhanced mechanism compared with the serial mechanism; it is similar to reducing the first-level cache latency by one cycle. For instance, the graph related to the 3-cycle latency, serial verification and 8-way processor, and the graph related to the 4-cycle latency, enhanced verification and 8-way processor, are almost overlapped. Furthermore, to achieve a target performance, processors with enhanced verification need fewer IQ entries than processors with serial verification. The behaviour in 4-way processors is similar to the behaviour in 8-way processors.

5 Conclusions

In this work we propose the detailed design of a verification mechanism: the Verification Issue Queue. This mechanism signals when an instruction can be removed from the Issue Queue or when it must be re-issued. Our evaluations show that the verification mechanism is crucial to exploit the performance potential of

data-speculative processors. We also evaluate the use of two possible verification flow graphs (serial and enhanced) by the VIQ. Our results, in the context of load address prediction, show the importance of removing the instructions from the IQ as soon as possible: using the enhanced verification instead the serial verification has a performance impact similar to reducing first-level cache latency by one cycle. Future work must be developed to design verification mechanisms more scalable than the VIQ. Although the VIQ allows the IQ to be decoupled from the Reorder Buffer, some IQ and VIQ structures are proportional to the number of physical registers.

References

1. G. Reinman and B. Calder, "Predictive techniques for aggressive load speculation," in *International Symposium on Microarchitecture*, 1998, pp. 127–137.
2. T. Sato, "Quantitative evaluation of pipelining and decoupling a dynamic instruction scheduling mechanism," *Journal of System Architecture*, vol. 46, no. 13, pp. 1231–1252, 2000.
3. E. Morancho, J. M. Llabería, and À. Olivé, "Recovery mechanism for latency misprediction," in *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pp. 118–128.
4. R. E. Kessler, "The alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, 1999.
5. K. Dieffendorff, "Hal makes sparcs fly," *Microprocessor Report*, vol. 13, no. 5, Nov 1999.
6. T. Sato, "Data dependence speculation using data address prediction and its enhancement with instruction reissue," in *24 th. EUROMICRO Conference Volume 1 (EUROMICRO'98)*. IEEE Computer Society, Aug. 1998, pp. 285–292.
7. Y. Sazeides, "An analysis of value predictability and its application to a superscalar processor," Ph.D. Thesis, University of Wisconsin-Madison, 1999.
8. S. Palacharla, "Complexity-effective superscalar processors," Ph.D. Thesis, University of Wisconsin-Madison, Department Computer Science, 1998.
9. M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita, and S. ichiro Mori, "A high-speed dynamic instruction scheduling scheme for superscalar processors," in *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*. IEEE Computer Society, 2001, pp. 225–236.
10. J. Farrell and T. C. Fischer, "Issue logic for a 600-mhz out-of-order execution microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 707–712, 1998.
11. D. Burger, T. Austin, and S. Bennet, "Evaluating future microprocessors: The simplescalar tool set," University of Wisconsin-Madison, Department Computer Science, Tech. Rep. CS-TR-96-1308, 1996.
12. E. Morancho, "Address prediction and recovery mechanisms," Ph.D. Thesis, Dept. of Computer Architecture, Universidad Politècnica de Catalunya, May 2002.