Available online at www.sciencedirect.com

ScienceDirect

Journal of Systems Architecture 53 (2007) 170–183

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

A comparison of two policies for issuing instructions speculatively

Enric Morancho *, José María Llabería, Àngel Olivé

Computer Architecture Department, Universitat Politécnica de Catalunya, Spain
Members of the European Network of Excellence on High Performance Embedded Architectures and Compilers (HiPEAC)

Received 13 October 2005; received in revised form 7 June 2006; accepted 11 September 2006
Available online 15 November 2006

Abstract

Value speculation is a speculative technique proposed to reduce the execution time of programs. It relies on a predictor, a checker and a recovery mechanism. The predictor predicts the result of an instruction in order to issue speculatively its dependent instructions, the checker checks the prediction after issuing the predicted instruction, and the recovery mechanism deals with mispredictions in order to maintain program correctness.

Previous works on value speculation have considered that the instructions dependent on a predicted instruction can be issued before issuing the predicted instruction (*non-delayed* issue policy). In this work we propose delaying the issue time of the instructions dependent on a value-predicted instruction until issuing the value-predicted instruction (*delayed* issue policy). Although the potential performance benefits of the *delayed* issue policy are smaller than that of the *non-delayed* issue policy, the recovery mechanism required by the *delayed* issue policy is simpler than the recovery mechanism required by the *non-delayed* issue policy.

We have evaluated both issue policies in the context of load-value prediction by means of address prediction in order to determine in which scenarios the performance of the *delayed* issue policy is competitive with that of the *non-delayed* issue policy. Our results show that the *delayed* policy is a cost-effective alternative to the *non-delayed* policy, especially for realistic issue-queue sizes.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Speculative execution; Value speculation; Recovery mechanism; Address prediction

1. Introduction

Value speculation is a speculative technique that has been proposed to reduce the execution time of programs [1,7]. Value speculation relies on predicting an instruction result and forwarding the prediction to its dependent instructions in order to speculatively issue them. Later, the prediction must be checked. Prediction check is performed after issuing the predicted instruction. On a correct prediction, the speculative results turn out to be valid; otherwise, a recovery mechanism must re-issue the speculatively issued instructions in order to

* Corresponding author. Tel.: +34 93 401 7188; fax: +34 93 401 7055.
E-mail addresses: enricm@ac.upc.edu (E. Morancho), llaberia@ac.upc.edu (J.M. Llabería), angel@ac.upc.edu (À. Olivé).

maintain program correctness because they have consumed an incorrect input value.

Previous works on value speculation (e.g., [3–6]) allow issuing instructions dependent on the predicted instruction before issuing the predicted instruction (through this paper, this issue policy is named *non-delayed*).

In this work we propose delaying the issue time of the instructions dependent on a value-predicted instruction until issuing the value-predicted instruction (*delayed* policy). Although the *delayed* policy has smaller potential performance benefits than the *non-delayed* one, the *delayed* alternative allows using a simpler recovery mechanism than the *non-delayed* one. While the *non-delayed* issue policy requires a generic recovery mechanism, the *delayed* issue policy may also use a simpler recovery mechanism specific for a speculative technique named latency prediction (Section 2.2). Consequently, the *delayed* policy may be a cost-effective alternative to the *non-delayed* one.

Our evaluations are focused in the scope of load-value prediction by means of address prediction and speculative memory accesses. We consider load-value prediction using both issue policies (*non-delayed* issue policy and *delayed* issue policy). In both cases, processor front-end predicts the effective addresses computed by load instructions and triggers speculative memory accesses. The models differ in the issue time of the dependent instructions. In the first case, instructions dependent on a predicted load instruction may be speculatively issued as soon as the dependent instructions enter in the Issue Queue (IQ), that is, may be issued before issuing the predicted load instruction they depend on. In the second case, instructions dependent on the predicted load instruction are speculatively issued after issuing the predicted load instruction.

Our results show that, in certain scenarios, the performance of the *delayed* policy combined with a recovery mechanism specific for latency mispredictions is competitive with the performance of the *non-delayed* policy with the generic recovery mechanism.

The contributions of this work are: (a) we propose delaying the issue time of the instructions dependent on a value-predicted instruction until issuing the value-predicted instruction in order to allow the usage of a simpler recovery mechanism, (b) we evaluate this proposal in the scope of load-value prediction by means of address prediction.

This paper is organized as follows. Section 2 describes some background for this work. Section 3 details the characteristics of the issue policies considered in this work. Section 4 describes the evaluated processor models. Section 5 presents the evaluation of the issue policies. Section 6 reports some related works and Section 7 summarizes the conclusions of this work.

2. Background

This section describes some concepts used through this paper.

2.1. Value speculation

Value speculation is a speculative technique that relies on predicting an instruction result and forwarding the prediction to its dependent instructions in order to speculatively issue them [1,7].

Predictions are performed early in the pipeline (concurrently with instruction fetch or with instruction decode) in order to allow propagating them to the dependent instructions before these instructions are inserted into the IQ.

Instructions with ready operands (either speculatively or non-speculatively), compete to be issued. Consequently, an instruction with speculative operands may be issued before issuing the predicted instruction (we refer to this issue policy as *non-delayed*).

When a predicted instruction is issued, its prediction is checked. On a correct prediction, the speculatively issued instructions dependent on the prediction had consumed correct input data and their results are considered as valid. However, on a wrong prediction, they had consumed incorrect input data; then, a recovery mechanism must reissue these instructions.

2.2. Latency prediction

Latency prediction is a speculative technique applied by several superscalar processors (for instance, Alpha 21264 processor [8] and Pentium 4 processor [9]) to deal with instructions which exact latency is unknown at issue time. For instance, the latency of a load instruction depends on the memory-hierarchy level where data is located and on resource constraints. Consequently, the exact latency of a load instruction is known several cycles after issuing the load instruction.

To schedule the instructions dependent on such instructions the scheduler has two options. On one hand, the scheduler may delay the scheduling of the dependent instructions until knowing the exact latency of the unknown-latency instruction; however, this option increases the effective latency of the unknown-latency instruction and decreases processor performance [10,11]. On the other hand, the scheduler may predict the latency of the unknown-latency instruction, issue it, and speculatively schedule its dependent instructions accordingly. Several cycles after issuing the latency-predicted instruction, its latency prediction is checked. On a correct latency prediction, the issued dependent instructions had consumed a correct input data; however, on an incorrect latency prediction, the issued dependent instructions had consumed an incorrect input data and the recovery mechanism must re-issue them.

To schedule instructions dependent on load instructions, Alpha 21264 and Pentium 4 predict whether a load instruction hits first level cache. Then, the scheduler schedules the dependent instructions according to this prediction.

There are two implicit characteristics of latency prediction that are relevant to this work: (a) the instructions dependent on the latency-predicted instruction are issued after issuing the latency-predicted instruction (that is, it uses the *delayed* issue policy) and (b) the number of cycles from issuing a latency-predicted instruction until knowing the correctness of the latency prediction is fixed.

2.3. Recovery mechanisms

Processors that implement speculative execution must deal with wrong predictions. The mechanism responsible for maintaining program correctness despite mispredictions is the recovery mechanism. There are several alternatives to implement the recovery mechanism varying performance impact and hardware cost.

2.3.1. Re-fetch versus re-issue

On *re-fetch* mechanisms, the instructions younger than the mispredicted instruction are flushed-out from the pipeline and the fetch unit is redirected to the next instruction after the mispredicted instruction.

On *re-issue* mechanisms, the instructions are not read again from memory because the mechanism keeps enough information to re-execute them.

In the scope of branch prediction, a wrong prediction introduces instructions from a wrong execution path in the pipeline. After detecting the misprediction, these instructions must be discarded and a new execution path must be fetched. Consequently, branch mispredictions are dealt with *re-fetch* recovery mechanisms. Some researchers have improved the recovery mechanisms for branch mispredictions by considering the existence of control-independent instructions (e.g. [12,13]).

In the scope of value speculation, both *re-fetch* and *re-issue* recovery mechanisms may deal with value mispredictions because, unless branch mispredictions are involved, the execution path remains unaltered. However, due to the large performance cost of re-fetching (and re-decoding,...) the instructions younger than the mispredicted instruction, *re-issue* recovery mechanisms are the selected choice to deal with value mispredictions [14].

re-issue mechanisms are classified considering which storage structure keeps the speculatively issued instructions: the IQ or a specific storage structure.

Some proposed recovery mechanisms keep the speculatively issued instructions in the IQ both in the scope of value speculation [15–18] and in the scope of latency prediction [8]. However, keeping these instructions in the IQ reduces its effective size and may harm processor performance.

Some researchers have proposed recovery mechanisms specific for the scope of latency prediction. Instead of keeping the speculatively issued instructions in IQ entries, these mechanisms keep these instructions in devoted structures (the Recovery Buffer [19] or the Slice Processing Unit [20]). Moreover, recovering from latency mispredictions requires a simpler management than recovering from general value mispredictions.

2.3.2. Selective versus non-selective

The *re-issue* recovery mechanisms can be classified either as *selective* or as *non-selective*. On *selective* recovery mechanisms, only the speculatively issued instructions dependent on the misprediction are re-issued; on *non-selective* mechanisms, some independent instructions may also be re-issued.

The *selective* recovery mechanisms must use a dependence tracking mechanism to detect the instructions to be re-issued. Next subsection describes several alternatives to implement it.

In our evaluations we use *selective re-issue* recovery mechanisms.

2.4. Verification mechanisms

Processors that implement value speculation use a verification mechanism for propagating to its dependent instructions that an operand has become either speculative or non-speculative. This mechanism influences on:

– branch-resolution delay because we use non-speculative branch resolution [21], that is, the operands of a branch instruction must be verified before resolving the branch instruction,
– the effective capacity of the IQ on processors that keep the speculatively issued instructions in the IQ until verifying their operands,
– the re-issue time of the instructions dependent on a misprediction.

2.4.1. Keeping the speculatively issued instructions in the IQ

In the literature, there are several alternatives to perform the verification process. Sato [16] assumed an RRU-based processor [22] (the ROB and the IQ are grouped into the same structure: the RUU). He used a mechanism where no specific hardware is devoted to propagating the verifications because instructions are implicitly verified as soon as they reach the ROB's head entry; we name this mechanism *implicit verification on commit*. The same approach was used in [18] in the context of memory communication (by predicting store-load dependences). Rychlik et al. [23] proposed a mechanism that uses the data-flow graph to propagate verifications serially, that is, the verification of an instruction is propagated only to its directly dependent instructions. Sazeides [17] also used the data-flow graph to propagate verifications; he assumed that they can traverse the flow graph on a single cycle, regardless the number of graph levels.

Morancho et al. [24] proposed a verification mechanism named Verification Issue Queue (VIQ). The VIQ allows the processor verifying the instructions before reaching the head entry of the ROB. The VIQ is fed with a verification-flow graph. Attending to the verification-flow graph used, they consider *serial* verification and *enhanced* verification (on a single cycle, the verification of an instruction is propagated to its directly and to its indirectly dependent instructions).

Fig. 1 depicts a simplified scheme of the communication between the IQ and the VIQ. After being renamed, all instructions are inserted in both structures. Each cycle, the IQ notifies to the VIQ which instructions have been issued; the VIQ also receives the prediction checks. The VIQ propagates check results through the verification flow graph. As the graph is being traversed, the VIQ notifies which instructions can be extracted from the IQ and which ones must be re-issued due to be dependent on a misprediction. The hardware cost of the VIQ is similar to the hardware cost of the IQ because the VIQ operates at the same frequency that the IQ and the VIQ is implemented using a matrix that has the same structure that the matrix that implements the IQ [24].

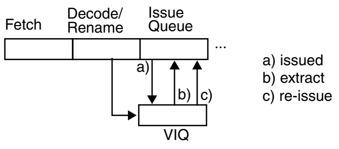


Fig. 1. Communication between the Issue Queue and the Verification Issue Queue.

2.4.2. Keeping the speculatively issued instructions in a devoted structure: the Recovery Buffer

Fig. 2 depicts the placement of the Recovery Buffer (RB) in the processor pipeline. As soon as an instruction is issued, it is extracted from the IQ and it is inserted in the RB. After a fixed number of cycles, issued instructions are classified either as not dependent on mispredictions or as dependent on mispredictions. This classification is performed in issue-order using a register scoreboard. Instructions not dependent on mispredictions are discarded; instructions dependent on mispredictions are re-issued from the RB.

The RB is implemented using a two-level buffer structure. The first level is a FIFO buffer; its size depends on misprediction-detection latency. In [19], the RB was proposed to deal with L1 misses: while instructions not dependent on L1 misses were extracted from the RB, issued instructions dependent on L1 misses were re-issued from the RB.

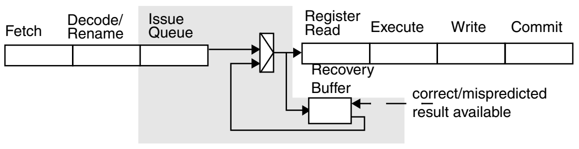


Fig. 2. Placement of the Recovery Buffer in the processor pipeline.

The verification is performed by a register scoreboard as instructions leave the first-level buffer.

3. Policies for issuing instructions speculatively

In this work we consider two policies for issuing instructions dependent on a instruction which result has been predicted: *non-delayed* and *delayed*. In the first case, a dependent instruction can be speculatively issued as soon as it enters into the IQ. This policy has been used by previous works on value speculation (e.g., [3,5,6]). In the second case, the speculative issue is delayed until issuing the predicted instruction it depends on. This policy is also used by the speculative technique named latency prediction.

As our evaluation focuses on load-value prediction by means of address prediction, Fig. 3 shows the execution of a value-predicted load instruction and a dependent instruction using both alternatives. In both examples we assume that the speculative memory access has been completed before inserting its dependent instruction in the IQ. Then, in the examples, the dependent instruction reads the speculative data from register file.

Fig. 3(a) shows the execution of a value-predicted load instruction and a dependent instruction using the *non-delayed* policy. The instruction dependent on the value-predicted load instruction is speculatively issued as soon as it is inserted in the IQ because the speculative memory access has been completed. The value-predicted load instruction is issued when its source operand is available.

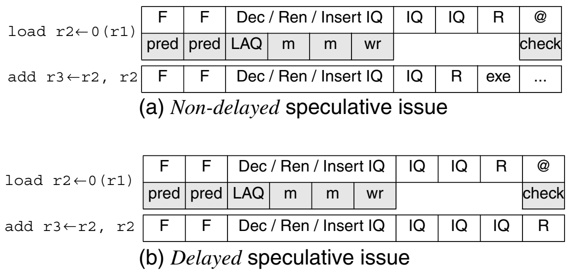


Fig. 3. Execution of a predicted load instruction and the speculative issue of a dependent instruction: (a) *non-delayed* issue policy (b) *delayed* issue policy. We assume a two-cycle first-level cache latency. Actions related to the speculative execution have been shaded. Legend: F (fetch), Dec. (decode), Ren. (rename), IQ (issue queue), R (register read), @ (effective-address computation), pred. (effective-address prediction), LAQ (load-address queue), m (memory access), wr (write), check (address-prediction check).

Fig. 3(b) shows the execution of a value-predicted load instruction and a dependent instruction using the *delayed* policy. The dependent instruction is issued after issuing the value-predicted load instruction with a load-use delay of just one cycle.

In these examples, the *delayed* policy delays the issuing of the dependent instructions two cycles with respect to the *non-delayed* policy.

3.1. Implications of the issue policy on the issue logic and the recovery mechanism

In both cases, we use a *selective re-issue* recovery mechanism to deal with mispredictions. However, the implementations of the recovery mechanisms will differ.

3.1.1. Non-delayed issue policy

The recovery mechanism for the value-speculative processor with the *non-delayed* issue policy is a generic recovery mechanism for value speculation. This mechanism keeps the speculatively issued instructions in the IQ until they become non speculative.

The implementation of the *non-delayed* issue policy with the generic recovery mechanism requires several modifications to the issue logic. After predicting an instruction result, processor front-end must insert the predicted instruction into the wakeup logic with its output register tagged as available. This allows issuing its dependent instructions as soon as they enter into the IQ.

As the speculatively issued instructions are kept into the IQ, the IQ must identify which entries correspond to instructions issued speculatively. These entries can not be visible to the select logic of the IQ and can not be freed until verifying the instruction.

An IQ entry can be freed only when the verification mechanism allows it. In Section 2.4 we have shown several possible implementations; however, as we will show in Section 4.2.1, the performance benefits of value prediction heavily depend on the verification mechanism. We assume the VIQ verification mechanism, which is implemented using an structure similar to the IQ.

3.1.2. Delayed issue policy

As the *delayed* issue policy is also used by the speculative technique named latency prediction, value-speculative processors with *delayed* issue policy may use a recovery mechanism specific for latency prediction: the Recovery Buffer [19]. This mechanism keeps the speculatively issued instructions in a dedicated structure. Moreover, the verification process is performed using a register scoreboard.

The implementation of the *delayed* issue policy also requires some modifications to the issue logic.

After predicting an instruction result, processor front-end must notify the new latency of the predicted instruction. This allows the wake-up logic to wake-up its dependent instructions according to the predicted latency. As soon as an instruction is issued, it can be extracted from the IQ.

The IQ must maintain a set of dependence vectors to track the speculatively issued instructions dependent on a predicted instruction. These vectors are needed to stop issuing instructions dependent on a mispredicted instruction when a misprediction is detected.

Although the *delayed* policy is more restrictive than the *non-delayed* policy, we are interested in evaluating if the *delayed* policy is a cost-effective alternative to the *non-delayed* policy. Our interest is initiated because value speculation with the *delayed* issue policy may use a simpler recovery mechanism than value speculation with the *non-delayed* issue policy.

4. Processor models

This section describes the main characteristics of the processor models used in this work: a non-value-speculative processor and a value-speculative processor.

4.1. Non-value-speculative processor

Our non-value-speculative processor model (for short, non-speculative processor) is similar to existing superscalar processors. Its block organization is depicted in Fig. 4 (solid lines and hollow boxes).

Fig. 5 shows the processor pipeline related to both non-speculative and speculative processors. Some stages may take several processor cycles.

Table 1 details the characteristics of the processors used in this work: a 4-way processor (resembles the Alpha 21264 processor) and a 8-way processor. In our evaluations we used oracle memory disambiguation. In the literature [25] have been proposed memory-dependence predictors that exhibit nearly optimal performance in processors similar to the ones described in Table 1.

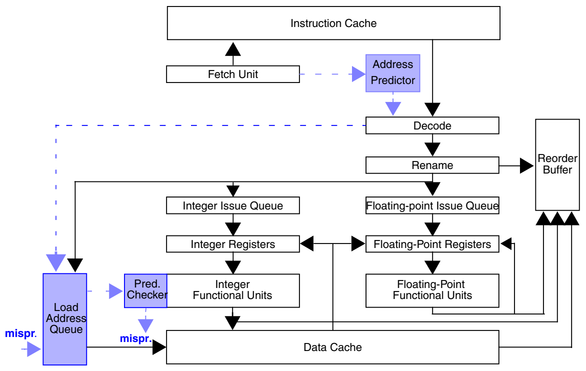


Fig. 4. Block organization of the non-speculative processor (hollow boxes and solid lines) and the speculative processor.

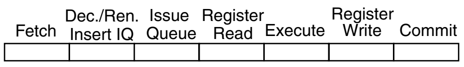


Fig. 5. Processor pipeline.

4.2. Value-speculative processors

The block organization of our value-speculative processor model (for short, speculative processor) is also depicted in Fig. 4 (all lines and boxes). Its main differences with respect to the non-speculative organization are detailed in the following list:

– Concurrently with instruction fetch, the address predictor accesses prediction tables to obtain address predictions for the load instructions being fetched. We assume an address predictor with the same latency as the fetch engine and immediate update of the prediction tables.

  All the evaluations of value-speculative processors performed in this work use an Hybrid Address Predictor made up of a conventional Stride-Address Predictor, a Context-Address Predictor [3] and a bimodal selector. We use a configuration of the previous address predictor with large prediction tables: the Stride-Address Predictor has a 16 K-entry Address Table and the Context-Address Predictor has 16 K-entry Value History Table and Value Prediction Table. Using this configuration, we stress all the mechanisms specific to address prediction and value speculation.

  Table 2 presents the percent of committed load instructions, the coverage (percent of loads correctly predicted among all load instructions) and the accuracy (percent of correct predictions among all predictions) obtained by the address predictor on SPEC95-INT benchmarks.

Table 1
Processor configurations used in this work

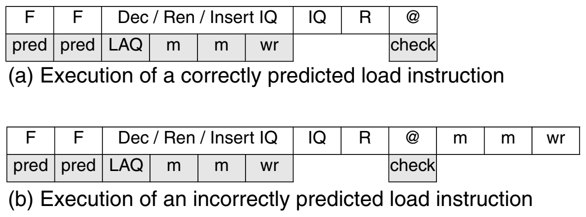
	4 way	8 way
Fetch/decode width	4 instrs./cycle	8 instrs./cycle
Fetch-engine latency	2 cycles	
Branch prediction	1 target/cycle 2^{15}-entry local predictor, 2^{15}-entry global predictor, 32-entry RAS, 1024-entry, 4-way BTB Penalty: 6 cyc. + fetch-engine latency	2 targets/cycle
Reorder Buffer	128 entries	256 entries
Load-Address Queue	64 entries	128 entries
Issue width	4 INT + 2 FP	8 INT + 4 FP
Functional units and latencies		
INT ALUs (1 cycle)	4 units (2 @ adders)	8 units (4 @ adders)
INT mul/div (3/20 c.)	1 u.	2 u.
FP ALUS (4 c.)	1 u.	2 u.
FP mul/div (4/12 c.)	1 u.	2 u.
RW-Memory ports	2 u.	4 u.
First-level caches	Separated, 64 KB, direct-mapped, write back, write allocate, 2-cycle hit	
Second-level cache	Unified, 1 MB direct-mapped, 12-cycle hit latency	
Main memory	80-cycle latency	
Load-latency prediction	Oracle	
Memory disambiguation	Oracle	
Commit width	8 instrs./cycle	16 instrs./cycle

Table 2
Benchmark characterization: percent of committed load instructions, coverage and accuracy obtained by the address predictor on SPEC95-INT benchmarks

Benchmarks	% Loads	Coverage	Accuracy
go	28.8	59.2	91.6
m88ksim	24.6	96.3	97.8
gcc	26.8	83.3	95.0
compress	18.7	76.2	96.2
li	25.5	85.0	93.8
ijpeg	18.1	75.2	96.0
perl	22.5	99.0	99.3
vortex	25.6	90.7	95.5
Average	23.8	83.1	96.0

– Each predicted load instruction is inserted in the Load-Address Queue (LAQ) during the decode stage. Moreover, the predicted address and the mapping for the destination register must be recorded in the related LAQ entry. After that, the speculative memory access can be issued from the LAQ. We assume that predicted effective addresses are inserted into the LAQ during the first decode stage.
– The number of cache accesses that can be performed every cycle remains unchanged with respect to the non-speculative processor. An arbiter prioritizes memory access initiated from the IQ respect memory accesses initiated from the LAQ.
– A functional unit must check the correctness of each prediction. The functional unit that performs this checking obtains the predicted effective address from the LAQ. This checking can be performed concurrently with the effective-address computation [26].
– The recovery mechanism depends on the issue policy. Speculative processors using the *non-delayed* issue policy (for short, *non-delayed* speculative processors) implement a generic recovery mechanism. Speculative processors using the *delayed* issue policy (for short, *delayed* speculative processors) implement a recovery mechanism specific for latency misprediction.

The processor pipeline associated to the speculative processor is the same as that of the non-speculative processor. Fig. 6 shows the execution of predicted load instructions; we have assumed that both the fetch-engine and the cache-access latencies are two cycles, and the availability of a free data-cache port for the predicted memory access.



F	F	Dec / Ren / Insert IQ				IQ	R	@
pred	pred	LAQ	m	m	wr			check

(a) Execution of a correctly predicted load instruction

F	F	Dec / Ren / Insert IQ				IQ	R	@	m	m	wr
pred	pred	LAQ	m	m	wr			check			

(b) Execution of an incorrectly predicted load instruction

Fig. 6. Execution of a predicted load instruction: (a) correctly predicted and (b) incorrectly predicted. Actions related to the speculative execution have been shaded.

Fig. 6(a) depicts the execution of a correctly predicted load instruction. Fig. 6(b) describes the execution of a incorrectly predicted load instruction. After the misprediction has been detected, memory is accesses non-speculatively using the computed effective address.

4.2.1. *Influence of the verification mechanism in non-delayed speculative processors*

As a previous evaluation, we evaluate the sensitivity of *non-delayed* speculative processors to the verification mechanism (note that in *delayed* speculative processors, the verification is performed by a register scoreboard).

Fig. 7 shows the average performance of the non-speculative processor and two *non-delayed* speculative processors: the first one uses the implicit verification [16], the second one uses the VIQ with the *serial* verification [24]. In these previous evaluations we assume that the number of IQ entries is equal to the number of ROB entries.

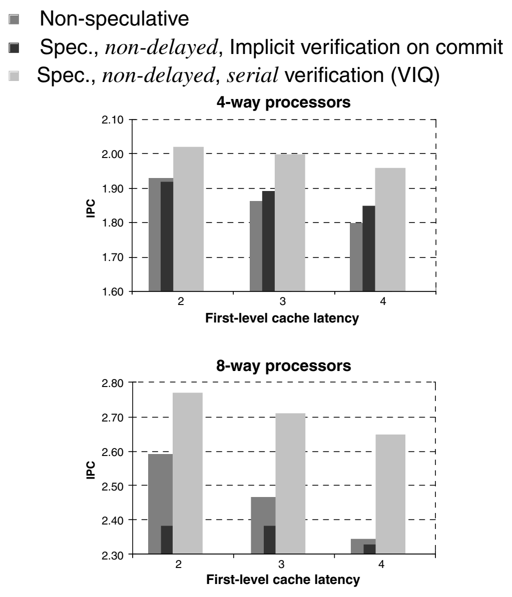


Fig. 7. IPC versus first-level cache latency in the non-speculative processor and two *non-delayed* speculative processors (implicit and *serial* verification).

We observe that the effectiveness of address prediction and value speculation heavily depends on the verification mechanism. In fact, in most cases, the performance of the processor with implicit verification is worse than the performance of the non-speculative processor.

Comparing both speculative processors, the use of the VIQ improves the performance around 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's head-entry.

Moreover, current processors decouple the IQ from the ROB. In this case, the verification mechanism may be also used to decide when an instruction can be extracted from the IQ. In this work, the VIQ will be used in the *non-delayed* speculative processors to allow both extracting instructions from the IQ and determining that the outcome of a branch instruction is non-speculative in order to, eventually, initiate the branch-misprediction recovery.

5. Evaluation

To perform the evaluations we have derived a simulator from the SimpleScalar 3.0 (Alpha ISA) cycle-by-cycle simulator [27]. As a benchmark suite we use the SPEC95-INT benchmarks and we simulate a representative execution interval [28]. We present harmonic average results.

5.1. *Delayed speculative processors versus non-speculative processors*

First, Fig. 8 shows the impact of the IQ size on the performance of non-speculative processors and *delayed* speculative processors; every graph connects results for the same data-cache latency.

We observe that the evaluated speculative processors outperform the non-speculative processors. For instance, in 4-way processors with a 25-entry IQ, the improvement ranges from 4% (2-cycle latency) to 10% (4-cycle latency); in 8-way processors with a 50-entry IQ, the improvement ranges from 6% (2-cycle latency) to 13% (4-cycle latency).

Furthermore, the performance of the evaluated speculative processors is less sensitive to small IQ

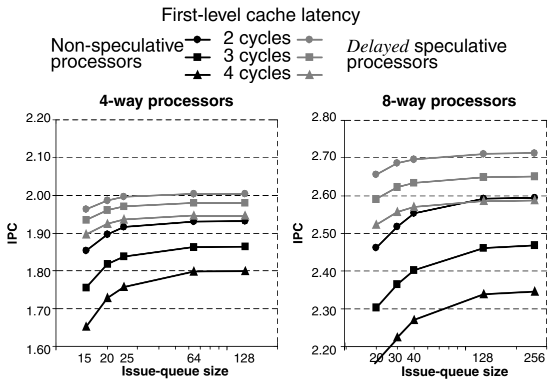


Fig. 8. IPC versus IQ size in non-speculative processors and *delayed* speculative processors.

sizes. For instance, in 4-way processors with a 20-entry IQ, non-speculative processors are below saturation from 2% (2-cycle latency) to 4% (4-cycle latency). However, speculative processors are below saturation around 1%. Consequently, the smaller the IQ size, the larger the performance impact of address prediction with *delayed* issue policy.

Finally, the evaluated speculative processors tolerate cache latency better than non-speculative processors. For instance, in 4-way processors with a 25-entry IQ, the performance degradation observed when cache latency increases from two to four cycles is 8% (non-speculative processors) and 3% (speculative processors); in 8-way processors with a 50-entry IQ, the degradation is 11% and 5% respectively.

5.2. *Delayed versus non-delayed issue policy*

Now, we compare the performance results obtained using both issue policies. Fig. 9 shows the average performance of non-speculative and both *delayed* and *non-delayed* speculative processors. The horizontal axis stands for the first-level-cache latency, both data and instruction caches, and each bar is related to a processor model (non-speculative, *serial* verification *non-delayed* issue, *enhanced*-verification *non-delayed* issue and *delayed* issue), vertical axis stands for the IPC. Each bar also stacks results for several IQ sizes (Appendix A presents individual results for each benchmark).

For a cache latency and issue-width, all evaluated speculative processors outperform the non-speculative processors, independently of the IQ size. Then, value speculation produces more performance benefits than enlarging the IQ of a non-speculative processor.

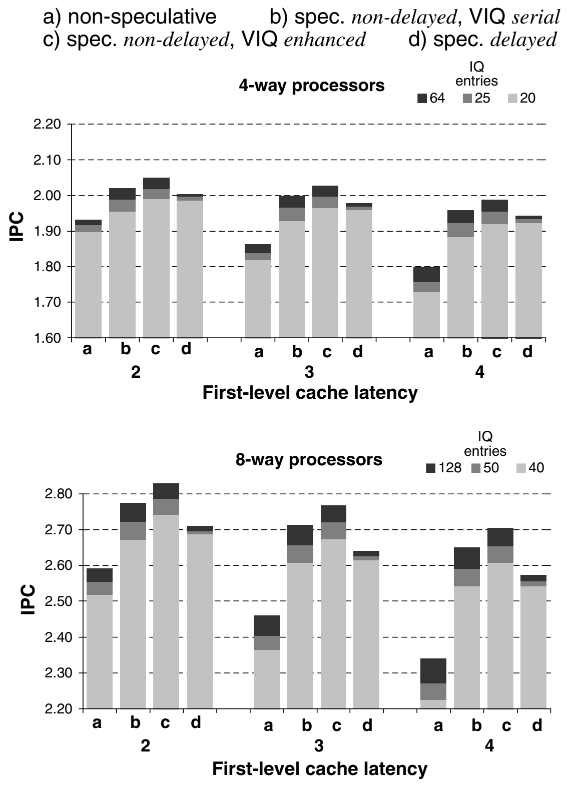


Fig. 9. Performance of non-speculative and several speculative processors.

The IQ is one of the most critical structures of superscalar processors. Consequently, we are interested in finding out the sensitivity of the evaluated processors to the IQ size. Our results show that while speculative processors with *non-delayed* issue are able to exploit large IQ sizes, the performance of processors with *delayed* issue almost saturates using relatively small IQs. For instance, in 4-way processors with a 20-entry IQ, while the performance of *delayed* -issue speculative processors is 1% below saturation, the performance of *non-delayed* speculative processors is around 3% below saturation. For 8-way processors with a 40-entry IQ, the results are similar.

This behavior can be explained because the *delayed* speculative processor extracts the instructions from the IQ as soon as they are issued. However, a *non-delayed* speculative processor retains the speculatively issued instructions into the IQ until they become non speculative. Then, increasing IQ size allows a *non-delayed* processor to reduce the performance degradation due to keeping in the IQ the speculatively issued instructions waiting for verification.

Comparing all speculative processors, our conclusions depend on the IQ size. In 4-way processors, for small IQs (20 entries), the *delayed* processors outperform about 2% *non-delayed* processors with *serial* verification, and perform like *non-delayed* processors with enhanced verification. For medium IQs (25 entries) *delayed* processors outperform *non-delayed* processors with *serial* verification about 0.5%, and present a performance degradation about 1% with respect to the *non-delayed* speculative processors with enhanced verification. Finally, for large IQs (64 entries), performance of *delayed* processors degrades about 0.5% and 2% with respect to *non-delayed* processors with *serial* and *enhanced* verification respectively. As the IQ size increases, the performance of *non-delayed* processors is less sensitive to keeping issued instructions in the IQ.

In 8-way processors, for small IQs (40 entries), *delayed* processors achieve similar performances to *non-delayed* processors with *serial* verification, and about 2% degradation with respect to the *non-delayed* processors with *enhanced* verification. For medium IQs (50 entries), performance degradation is about 1% (*serial*) and 3% (*enhanced*). For large IQs (128 entries), degradation reaches 2% and 4% respectively.

The performance difference between both issue policies is due to two factors. First, using the *delayed* policy, the instructions dependent on a predicted load instruction are issued after issuing the effective-address computation of the predicted load instruction; however, this restriction is not present using the *non-delayed* policy. Second, the recovery mechanism used by *delayed* speculative processors suffers an one-cycle penalty on each address misprediction.

From these evaluations we conclude that the *delayed* policy combined with the recovery mechanism based on the Recovery Buffer is an attractive alternative to the *non-delayed* policy, especially for realistic IQ sizes.

6. Related works

Several works have presented evaluations of the potential performance of address prediction and speculative execution. Reinman and Calder [5] presented an evaluation of speculation techniques that can be applied to load instructions: dependence prediction, address prediction, value prediction and memory renaming. They evaluated these techniques over a micro-architecture that fetches up to eight instructions per cycle, issues up to 16 instructions per cycle, couples the IQ and the reorder buffer into a 512-entry RUU, and has a first-level cache latency of four cycles. They considered the *non-delayed* issue policy and two recovery mechanisms: re-fetch, and selective re-issue. In their evaluations, they varied the address-predictor model (Last-Address Predictor, Stride Address Predictor, Context Address Predictor and Hybrid Address Predictor), the confidence estimator (conservative, conventional and oracle) and the recovery mechanism (re-fetch coupled with the conservative estimator, and re-issue coupled with the conventional estimator); however, the authors do not present the implementation of the recovery mechanisms. Their results show that, using the re-fetch recovery mechanism, the impact of address prediction is small (limited by a 1.04 speed-up). However, using the re-issue recovery mechanism, the impact is closer (up to 1.10 speed-up) to that of the oracle predictor (1.13).

Black et al. [4] evaluated the performance impact of address prediction and speculative execution. They assumed a realistic hybrid address predictor, the *non-delayed* issue policy and the re-fetch recovery mechanism. Their results showed the large performance degradation due to address mispredictions. Although the potential performance of address prediction and speculative execution, they concluded that its effectiveness will not be feasible until overcoming the effects of address mispredictions (by improving address prediction or by using selective recovery mechanisms).

Bekerman et al. [3] evaluated the performance impact of address prediction and speculative execution; they used a prediction model named Global-Correlated Context-Address Predictor. They focused on a prediction-table configuration and analyzed the impact of pipelining address prediction (prediction tables are not updated immediately). They assumed the *non-delayed* issue policy and a selective re-issue recovery mechanism. Their results showed that pipelining address prediction affects the performance impact of address prediction, but this impact is still significant (in SPEC95-INT benchmarks, speed-up decreases from 1.22 to 1.17).

Some works have proposed recovery mechanisms that can be applied to speculative processors with the *non-delayed* speculative issue. Akkary and Driscoll [2] proposed a two-level IQ. After decoding the instructions, they are sent to the rename stage (and

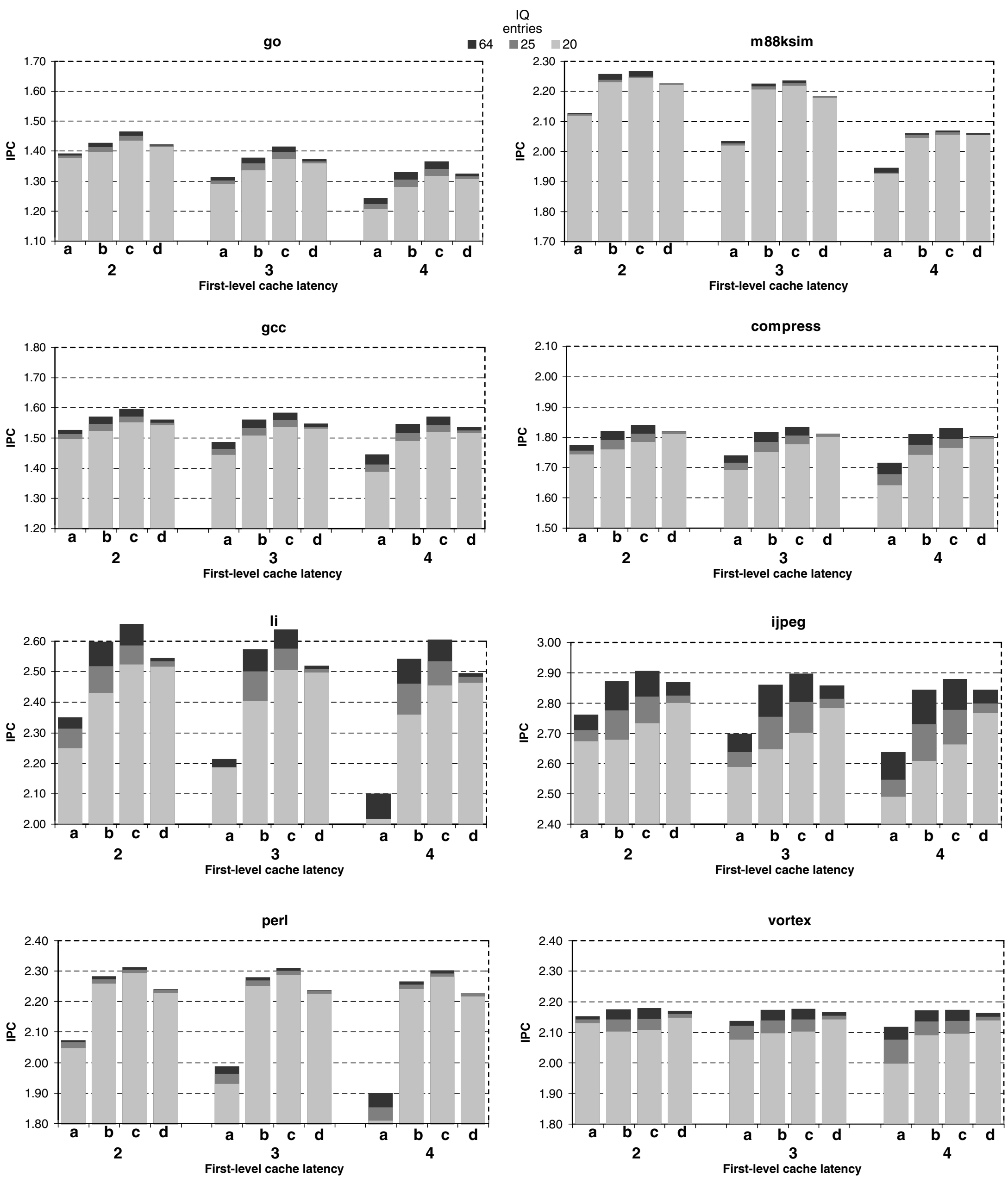


Fig. 10. Performance of 4-way non-speculative and several speculative processors on each SPEC95-INT benchmark (a) non-speculative, (b) spec. *non-delayed*, VIQ *serial*, (c) spec. *non-delayed*, VIQ *enhanced* and (d) spec. *delayed*).

after that, they are inserted into the IQ) and they are inserted into a trace buffer. Instructions are removed from the IQ as soon as they have been issued, but they remain in the trace buffer until they have been committed. On a misprediction, the trace buffer detects which instructions depend on the misprediction, groups them, and sends these instruction blocks to the rename stage.

Sato [6] also proposed a two-level structure: the scheduling window and the instruction buffer. The scheduling window is equivalent to the IQ; it holds non-issued instructions. The instruction buffer is equivalent to the RUU; it holds all the fetched instructions that have not been committed. On a misprediction, instructions are re-issued from the instruction buffer. There is a key difference between

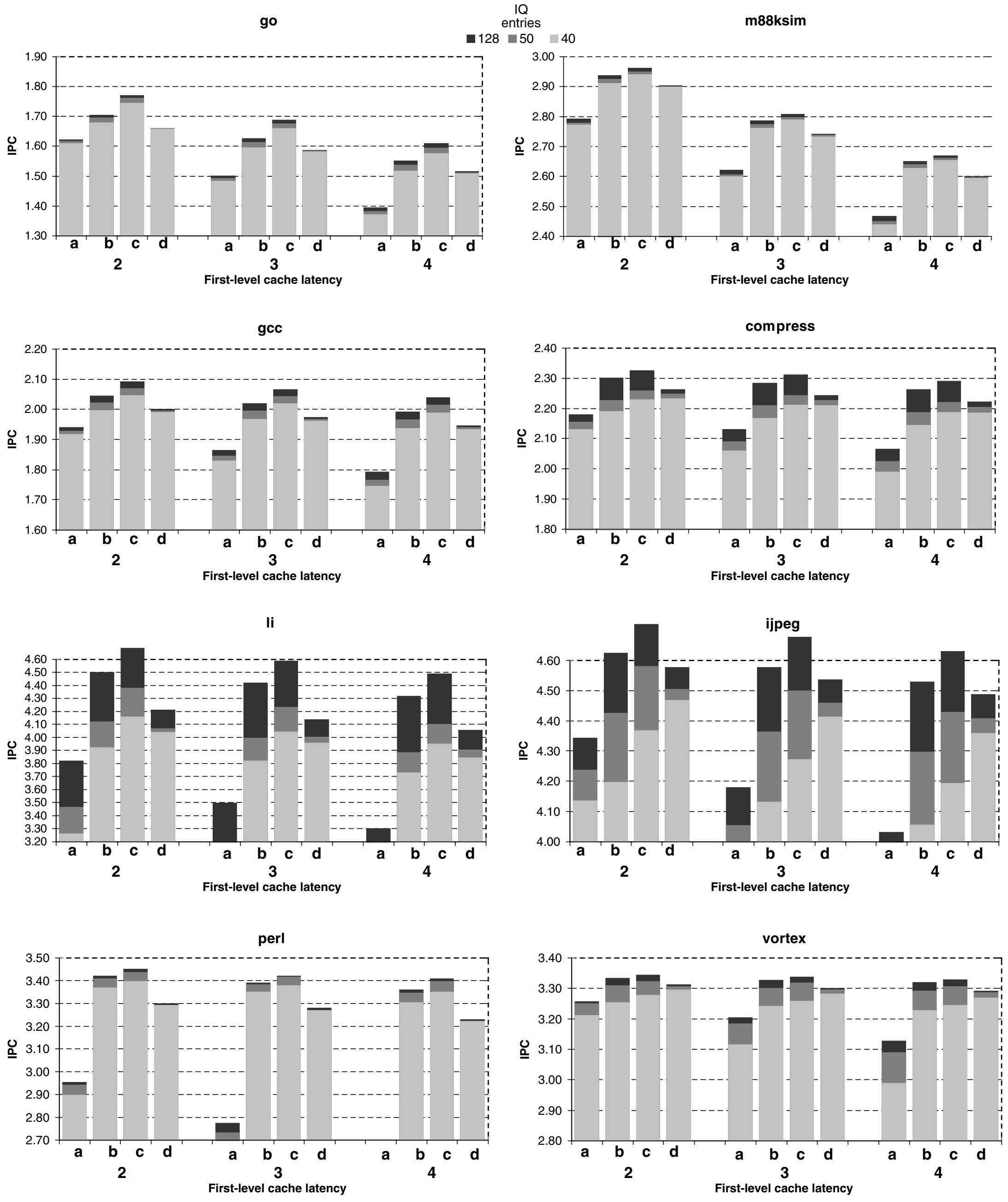


Fig. 11. Performance of 8-way non-speculative and several speculative processors on each SPEC95-INT benchmark (a) non-speculative, (b) spec. *non-delayed*, VIQ *serial*, (c) spec. *non-delayed*, VIQ *enhanced* and (d) spec. *delayed*).

the Recovery Buffer and this scheme. While the Recovery Buffer is a simpler structure because it records a valid scheduling of the instructions, Sato's proposal must replicate the scheduling logic in the scheduling window and the instruction buffer. Moreover, due to the large size of the instruction buffer, its select logic and its wake-up logic are pipelined.

7. Conclusions

We have evaluated two policies for issuing instructions speculatively in value-speculative processors: *non-delayed* and *delayed*. Although the *non-delayed* policy has larger potential performance benefits than the *delayed* policy, the *delayed* policy may be a cost-effective alternative to the *non-delayed* one. While the first policy needs a conventional recovery mechanism that puts additional pressure on the IQ, the second policy may use a recovery mechanism (the Recovery Buffer) that puts no additional pressure on the IQ.

We have evaluated both policies in the scope of load-value prediction by means of address prediction. Our results show that, depending on the number of IQ entries of the processor, value-speculative processors with *delayed* speculative issue (and the recovery mechanism based on the Recovery Buffer) is a cost-effective alternative to value-speculative processors with *non-delayed* speculative issue. The best scenario for the *delayed* policy is processors with a realistic number of IQ entries.

Acknowledgement

This work has been supported by the Ministry of Education and Science of Spain under the contract TIN-2004-07739-C02-01.

Appendix A. Detailed results

Figs. 10 and 11 show individual results in 4-way and 8-way processors respectively. We observe that, almost for all benchmarks, given a cache latency, all the speculative processors outperform all the non-speculative processors, independently on the IQ size.

For the smallest evaluated IQ sizes (20-entry in 4-way processors and 40-entry in 8-way processors), the performance of *delayed* speculative processors are competitive with the performance of *non-delayed* speculative processors with serial verification. The only exceptions are benchmarks *perl* (4-way and 8-way processors) and *m88ksim* (8-way processors). Both the two highest coverage and accuracy metrics of our address predictor are achieved in these benchmarks; consequently, the *delayed* issue policy is losing many chances of executing speculatively instructions dependent on correct predictions. Comparing the *delayed* processors versus the *non-delayed* processors with *enhanced* verification, *delayed* processors show performance degradations also in benchmarks *go*, *gcc* and *li* (8-way processors); degradation is about 5%, 3% and 3% respectively.

In benchmarks with higher IPC (*li* and *ijpeg*), we observe the effect of increasing IQ size on the performance of the evaluated processor models. As the *delayed* issue policy allows the processor to extract instructions from the IQ as soon they are issued, enlarging the IQ in speculative *delayed* processors presents smaller performance benefits than in the remaining processor models (non-speculative, and speculatives with the *non-delayed* issue policy).

References

[1] F. Gabbay, A. Mendelson, Speculative execution based on value prediction, Tech. Rep. EE Department TR-1080, Technion–Israel Institute of Technology, Israel, 1996.

[2] H. Akkary, M.A. Driscoll, A dynamic multithreading processor, in: MICRO 31: Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 226–236.

[3] M. Bekerman, S. Jourdan, R. Ronen, G. Kirshenboim, L. Rappoport, A. Yoaz, U. Weiser, Correlated load-address predictors, in: ISCA '99: Proceedings of the 26th Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, 1999, pp. 54–63.

[4] B. Black, B. Mueller, S. Postal, R. Rakvic, N. Utamaphethai, J.P. Shen, Load execution latency reduction, in: ICS '98: Proceedings of the 12th International Conference on Supercomputing, ACM Press, New York, NY, USA, 1998, pp. 29–36.

[5] G. Reinman, B. Calder, Predictive techniques for aggressive load speculation, in: MICRO 31: Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 127–137.

[6] T. Sato, Quantitative evaluation of pipelining and decoupling a dynamic instruction scheduling mechanism, Journal of Systems Architecture 46 (13) (2000) 1231–1252.

[7] M.H. Lipasti, C.B. Wilkerson, J.P. Shen, Value locality and load value prediction, in: ASPLOS-VII: Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems, ACM Press, New York, NY, USA, 1996, pp. 138–147.

[8] R.E. Kessler, The alpha 21264 microprocessor, IEEE Micro 19 (2) (1999) 24–36.

[9] G. Hinton, D. Sager, M. Upton, D. Boggs, et al., The microarchitecture of the pentium® 4 processor, Intel Technology Journal 5 (1) (2001) 1–12.

[10] J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, K. Lai, Bloom filtering cache misses for accurate data speculation and prefetching, in: ICS '02: Proceedings of the 16th International Conference on Supercomputing, ACM Press, New York, NY, USA, 2002, pp. 189–198.

[11] A. Yoaz, M. Erez, R. Ronen, S. Jourdan, Speculation techniques for improving load related instruction scheduling, in: ISCA '99: Proceedings of the 26th Annual International Symposium on Computer Architecture, IEEE Computer Society, Washington, DC, USA, 1999, pp. 42–53.

[12] E. Rotenberg, J. Smith, Control independence in trace processors, in: MICRO 32: Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture, IEEE Computer Society, Washington, DC, USA, 1999, pp. 4–15.

[13] A. Gandhi, H. Akkary, S.T. Srinivasan, Reducing branch misprediction penalty via selective branch recovery, in:

HPCA '04: Proceedings of the 10th International Symposium on High Performance Computer Architecture, IEEE Computer Society, Washington, DC, USA, pp. 254–263.
[14] M.H. Lipasti, Value locality and speculative execution, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1998.
[15] K. Diefendorff, Hal makes Sparcs fly, Microprocessor Report 13 (15) (1999) 5–13.
[16] T. Sato, Data dependence speculation using data address prediction and its enhancement with instruction reissue, in: EUROMICRO '98: Proceedings of the 24th Conference on EUROMICRO, IEEE Computer Society, Washington, DC, USA, 1998, pp. 285–292.
[17] Y.T. Sazeides, An analysis of value predictability and its application to a superscalar processor, Ph.D. thesis, University of Wisconsin-Madison, 1999.
[18] G.S. Tyson, T.M. Austin, Improving the accuracy and performance of memory communication through renaming, in: MICRO 30: Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, IEEE Computer Society, Washington, DC, USA, 1997, pp. 218–227.
[19] E. Morancho, J.M. Llabería, Àngel Olivé, Recovery mechanism for latency misprediction, in: PACT '01: Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques, IEEE Computer Society, Washington, DC, USA, 2001, pp. 118–128.
[20] S.T. Srinivasan, R. Rajwar, H. Akkary, A. Gandhi, M. Upton, Continual flow pipelines, in: ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM Press, New York, NY, USA, 2004, pp. 107–119.
[21] A. Sodani, G.S. Sohi, Understanding the differences between value prediction and instruction reuse, in: MICRO 31: Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 205–215.
[22] G.S. Sohi, S. Vajapeyam, Instruction issue logic for high-performance, interruptable pipelined processors, in: ISCA '87: Proceedings of the 14th Annual International Symposium on Computer Architecture, ACM Press, New York, NY, USA, 1987, pp. 27–34.
[23] B. Rychlik, J. Faistl, B. Krug, J.P. Shen, Efficacy and performance impact of value prediction, in: PACT '98: Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques, IEEE Computer Society, Washington, DC, USA, 1998, pp. 148–154.
[24] E. Morancho, J.M. Llabería, Àngel Olivé, A mechanism for verifying data speculation, in: Proceedings of the Euro-Par 2004 Parallel Processing, 2004, pp. 525–534.
[25] G.Z. Chrysos, J.S. Emer, Memory dependence prediction using store sets, in: ISCA '98: Proceedings of the 25th Annual International Symposium on Computer Architecture, IEEE Computer Society Press, Washington, DC, USA, 1998, pp. 142–153.
[26] J. Cortadella, J.M. Llabería, Evaluation of $a + b = k$ conditions without carry propagation, IEEE Transactions on Computers 41 (11) (1992) 1484–1488.
[27] D. Burger, T.M. Austin, S. Bennett, Evaluating future microprocessors: The simplescalar tool set, Tech. Rep. CS-TR-1996-1308, University of Wisconsin-Madison, 1996.
[28] E. Morancho, Address prediction and recovery mechanisms, Ph.D. thesis, Computer Architecture Department, Universitat Politécnica de Catalunya, 2002.