Counteracting Bank Misprediction in Sliced First-Level Caches*

E.F. Torres[1], P. Ibañez[1], V. Viñals[1], and J.M. Llabería[2]

[1] DIIS, Universidad de Zaragoza
[2] DAC, Universidad Politécnica de Catalunya

Abstract. Future processors having sliced memory pipelines will rely on bank prediction to schedule memory instructions to a first-level cache split into banks. In a deeply pipelined processor, even a small bank misprediction rate may degrade performance severely. The goal of this paper is to counteract the bank misprediction penalty, so that in spite of such bank misprediction, performance suffers little. Our contribution is twofold: a new recovery scheme for latency misprediction, and two policies for selectively replicating loads to all banks. The proposals have been evaluated for 4 and 8-way superscalar processors and a wide range of pipeline depths. The best combination of our mechanisms improves IPC of an 8-way baseline processor up to 11%, removing up to two thirds of the bank misprediction penalty.

1 Introduction

Current superscalar processors have the potential for exploiting high degrees of ILP, demanding a large memory bandwidth. So, to overcome the potential bottleneck of a single cache port, concurrent accesses to non-blocking first-level caches are required [21]. An ideal implementation should not involve latency increase, because programs often have load instructions at the head of critical chains of dependent instructions and so, increasing the load-use delay could be very harmful to performance.

Several alternatives have been used in commercial processors to support such bandwidth demands. *Truly multiported* caches [17], *Mirror caches* [7] and *Virtual-multiporting* [9, 14] caches have been widely studied and have drawbacks in terms of area, latency or scalability. *Multibanking* uses an interconnection component placed between address units and fast single-ported cache banks [4, 11, 13]. Multibanking scales easily to multiple ports, but the interconnection component (either a crossbar network [11] or a bank scheduler [4, 13]) may increase the latency of loads noticeably.

The *sliced memory pipeline* reduces the load latency of multibanked schemes by removing the interconnection component from the memory pipeline [5, 18, 22, 23]. Because the Instruction Issue Queue (IQ) does not know memory addresses yet and there is only one path to each bank, this design relies on *bank prediction* to schedule memory instructions to banks. A hardware loop between IQ and

* This work was supported in part by the Ministry of Education of Spain under grant TIC 2001-0995-C02-02 and the Diputación General de Aragón (BOA 58,14/05/03).

the address computation stage is responsible for checking the bank prediction (a *loose* loop according to Borch et al. [2]).

Speculation can be used to manage such a loose loop (load speculation). The IQ assumes the bank prediction to be correct and speculatively schedules dependent work at the proper time. If predictions are mostly correct, performance increases. However, at each misprediction the speculative work is lost and a recovery action is needed.

The aim of this papers is to counteracting the bad side effects of bank mispredictions in first-level sliced caches. We first introduce a new recovery scheme for reducing the recovery penalty. Secondly, we propose two policies for a selective replication of loads, so that the load-use delay and the number of load misspeculations are reduced.

We evaluate deeply pipelined 4 and 8-way out-of-order processors, considering that only a few logic levels will fit within the processor cycle [1]. Pentium 4 is an example, with 20 stages up to branch check, and 4 between IQ and Ex [10].

Section 2 details the processor and memory pipelines, offering an alternative for distributing the buffering needed for disambiguation. After detailing the simulation environment in Section 3, in Section 4 we isolate and analyze a number of factors that contribute to the bank misprediction penalty. Next, we propose two mechanisms for counteracting the bank misprediction penalty: Section 5 describes a new recovery mechanism and Section 6 introduces two mechanisms for spreading loads into banks in a selective way.

2 Baseline processor and memory system

Processor. Figure 1a shows the execution pipeline of the superscalar processor used as baseline. The front-end pipeline has eight stages from Fetch to IQ.

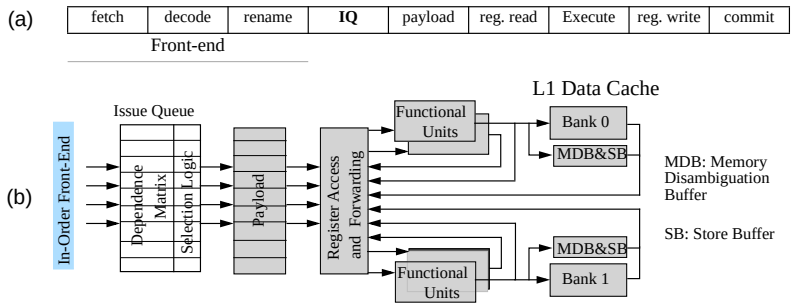

Fig. 1. Execution pipeline (a). Out-of-Order execution core with two cache banks (b).

The IQ has a Dependence Matrix and a Selection Logic [8], see Figure 1b. The Dependence Matrix encloses the dependency information among instructions and uses wired-OR logic to find ready instructions (wake-up phase). The Selection Logic issues the oldest instructions among the ready ones (select phase). After issuing, instruction payload is read, operands are read from the register file, and execution starts.

If the trend towards rising frequency holds, the number of stages between IQ and Ex will continue increasing [1, 2, 10]. So, we will take this number as a parameter in all our experiments and analyze its impact on performance.

The processor can issue speculatively while waiting for the resolution of all latency predictions, namely bank prediction, cache hit prediction and disambiguation buffer non-full prediction. Each prediction has its own resolution delay. IQ assumes the cache hit latency for loads, and speculatively issues their dependent instructions during a set of cycles called the *Speculative Window*. The Speculative Window starts at the first cycle in which the direct dependents can issue and ends either when the last-prediction acknowledgement reaches IQ, or when a misprediction is notified to the Recovery stage.

IQ is used as the Recovery stage. The recovery mechanism used as a baseline is described by Morancho et al. for latency prediction in [16]. It starts recovery as soon as a misprediction notification reaches IQ. It is selective, meaning that only the (already issued) instructions that depend on a mispredicted load will be reexecuted. With *Dependent Set* we refer to the dependent instructions issued during the Speculative Window. After every issue cycle, the instructions belonging to the Dependent Set are identified and the non-speculative instructions are removed from IQ. The load itself and the Dependent Set remain in IQ until the last cycle of the Speculative Window ends. Therefore, the IQ pressure increase is only determined by the longest resolution delay.

Sliced memory pipeline. The cache is sliced into banks. Each bank is only tied to one address computation unit. We assume that bank check is made concurrently with address computation by evaluating the expression A+B=K without carry propagation [6]. On a bank misprediction IQ knows about the right bank number during the cycle following bank check, see Figure 2. Then, the mispredicted instruction becomes ready to be issued again to the right bank one cycle later. We model the resolution delay for a bank prediction with a duration of up to 6 cycles, which is the case shown in Figure 2.

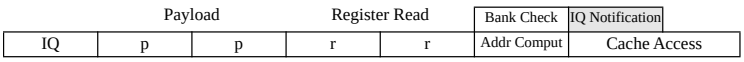

Fig. 2. Bank check delay.

Bank predictor. All memory instructions need a bank prediction. We assume that banks are line-interleaved. Therefore, the less significant bits of the line number are our prediction target. We have chosen to predict each address bit separately: 1 bit for 2 banks and 2 bits for 4 banks. As a bank predictor we use the *enhanced skewed binary predictor*, proposed by Michaud et al. to perform branch prediction [15]. We opt for a global predictor because it eases yielding several predictions per cycle [19]. Every bit predictor is sized identically: 1K entries for the three required tables and a history length of 10. Since we are interested in recognizing the hard-to-predict loads, we further add a 2-bit confidence saturating counter to every entry in all tables.

Each individual execution of a load has been classified according to confidence (high or low) and bank prediction outcome (right or wrong). Table 1 shows the

average percentages reached by the predictor for 2 and 4 banks (see workload details in Section 3).

Table 1. Average percentages reached by the bank predictor

	High confidence		Low confidence	
	right	wrong	right	wrong
2 banks	67.6	3.3	18.5	10.6
4 banks	53.7	2.6	22.8	20.9

Memory data flow. We assume memory disambiguation based on the Alpha 21264 approach [12], where loads place their addresses into a load buffer and stores place address and data at once into a store buffer. In this centralized model, the buffer entries are allocated in-order at decode, filled out-of-order at execution, and finally released at commit.

In a sliced memory system the accesses handled by the different cache banks reference disjoint memory regions. Consequently, the disambiguation unit could be distributed acting locally at each bank. Nevertheless, the entries of such distributed buffers cannot be allocated in fetch order because the bank number is unknown at decode time.

Zyuban&Kogge have suggested a distributed disambiguation mechanism for their multicluster architecture [23]. They solve the allocation problem partially by allocating an entry to every store in the buffer of every cluster at decode time. The main drawback of this approach is that the buffer of every cluster requires almost the same size of a centralized buffer.

We propose reducing the size of buffers in accordance with the number of banks and allocating buffer entries *after* the bank check (out-of-program order). A deadlock situation might arise when an allocation demand is rejected because all buffer entries have been already allocated.

The Buffer-Full condition is managed by adding a new prediction to the memory loop: it is predicted that there will be room and the memory instruction is speculatively issued. If the buffer is eventually full, a latency misprediction occurs. The faulting instruction is reissued from IQ when a signal indicates that the buffer is no longer full. On the other hand, in order to prevent deadlock it is sufficient to warrant that the oldest memory instruction in IQ can progress. This can be easily achieved by allocating the last free entry of a given buffer only if it is requested by the oldest memory instruction in IQ.

As an example, the considered 8-way processor with four banks and up to 128 in-flight loads/stores does not show performance losses when reducing the number of entries of every load buffer to 32 and of every store buffer to 16.

3 Simulation environment

We evaluate 4 and 8-wide issue processors having 8 stages before IQ and 1 to 4 stages between IQ and Ex. We have modified SimpleScalar 3.0c [3] in order to model a Reorder Buffer and a separate Issue Queue, looking carefully at all situations requiring latency misprediction recovery. We assume an oracle pre-

(a)

	4-issue	8-issue
fetch and decode width	4	8
branch predictor: hybrid (bimodal, gshare)	16 bits	
reorder buffer entries	128	256
load/store in-flight instructions	64	128
integer IQ entries	25	64
integer units	4	8
FP IQ entries	15	32
FP units	2	4
L1 I-cache	64KB, 2-way	

	4-issue	8-issue
L1 D-cache	2 cycles	
banks	2	4
ports per bank	1 r/w	
bank size	16 KB, 1-way	
line size	32 B	
LD buffer entries	32	
ST buffer entries	16	
L2 Unified Cache	1MB, 4-way, 12 cycles	
main memory	80 cycles	

(b)

Bench.	Data set
bzip2	program-ref
crafty	ref
eon	rushmeier-ref
gcc	166-ref
gzip	program-ref
mcf	ref
parser	ref
perl	diffmail-ref
twolf	ref
vortex	one-ref
vpr	route-ref

Fig. 3. Microarchitectural parameters (a). Used SPECint 2K programs (b).

dictor for store-load independency. The microarchitectural parameters used are summarized in Figure 3a.

We are interested in integer code because it exhibits difficult-to-predict address streams. As workload we use all the integer benchmarks of SPECint 2K but 254.gap which could not be executed within our framework (see Figure 3b). We use the Alpha binaries compiled by C. Weaver (www.simplescalar.com), simulating a contiguous run of 100M instructions from the points suggested by Sherwood et al. after a warming-up of 200M instructions [20].

4 Performance impact of bank misprediction

Figure 4a shows execution of a well-predicted load and two dependent instructions. We can see a load-use delay of 2 cycles (cycles *2:3*) and a 4-cycle Speculative Window (cycles *4:7*) inside which the Dependent Set (*add*, *sub*) is issued.

Figure 4b shows a bank misprediction. When IQ is notified a recovery action is performed during a full processor cycle (cycle *7*, box with *N*). Recovery implies stalling issue because the results of both the load and the Dependent Set must be tagged as not available in the Matrix Dependence. In the same cycle, the load itself and the Dependent Set become visible to the issue logic and they will be executed again as their operands become available (cycles *8*, *11*, and *12* for *ld*, *add* and *sub*, respectively).

When comparing the timings of the right and wrong bank predictions, the following differences appear: the load-use delay has 7 added cycles (cycles *4:10*), the work started by the Dependent Set during the Speculative Window has been lost, and finally, a cycle is spent for the recovery action (cycle *7*). Summarizing, the bank misprediction penalty is due to a number of factors, namely *the load-use delay increase*, *work loss*, and *recovery penalty*. We group the last two factors under the term *misspeculation penalty*. Notice that, as the resolution delay increases (cycles *2:7*) both the load-use delay and the work lost may also increase.

To quantify this penalty, Figure 5 shows IPC for processors having from 1 to 4 stages between IQ and Ex. First, we look at the bars labelled BASE and OBP

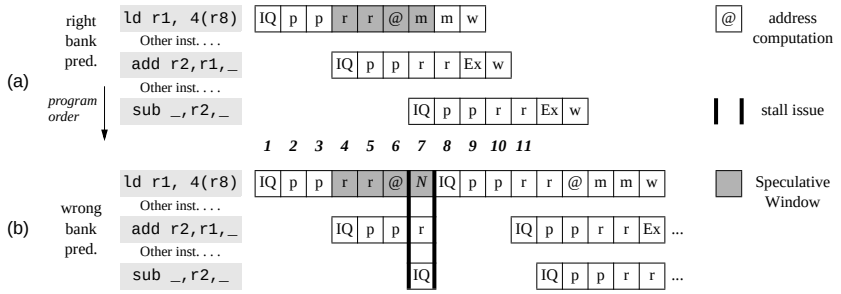

Fig. 4. Right and wrong bank predictions, assuming 4 cycles between IQ and Ex. For the sake of clarity only one instruction per cycle is issued. We show the stages of the instructions to be nullified until the cycle where IQ is notified of the misprediction.

(Oracle Bank Predictor). The BASE processor has the realistic bank predictor introduced above, while OBP has an oracle that always predicts the correct bank. So the IPC difference between OBP and the BASE processor (OBP-BASE) is the bank misprediction penalty (and the room for improvement). The IPC for OBP worsens with the number of IQ-Ex stages because the branch penalty increases.

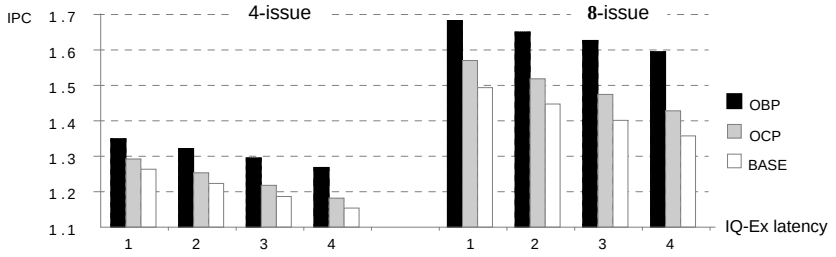

Fig. 5. Harmonic IPC mean with an Oracle Bank Predictor (OBP), an Oracle Confidence Predictor (OCP), and the real predictor (BASE).

Compared with OBP, the 4-way BASE processors experience an IPC loss raging from 6.4% to 9.0% as the number of IQ-Ex stages increases from 1 to 4. Eight-way BASE processors have more IPC loss, raging from 11.2% to 15.0%.

Another interesting point is breaking the misprediction penalty down into load-use delay increase and misspeculation penalty. For that we simulate the real bank predictor again, but now if the prediction is wrong the dependent instructions are not issued (OCP, Oracle Confidence Predictor). Thus, the misspeculation penalty is removed and so (OBP-OCP) difference is the load-use delay increase, whereas (OCP-BASE) difference is the misspeculation penalty.

Results show that both degrading factors are significant. However, load latency weighs more as the number of IQ-Ex stages increases; specifically, its contribution ranges from two thirds to three quarters as the number of IQ-Ex stages increases from 1 to 4. This is true for both 4-way and 8-way processors. In Section 6 we show how to reduce the overall misprediction penalty by replicating some loads, and in Section 5 we introduce a new recovery mechanism for reducing the recovery penalty.

5 Chained Recovery

Chained Recovery (CR) reduces the recovery penalty by not stalling issue as the baseline recovery does. Instead, the recovery process may require now several cycles because, in a given cycle, instructions that depend on previously nullified ones may be issued. In this Section the CR concept is applied to bank misprediction, but notice that it can also be used to recover from any kind of latency misprediction.

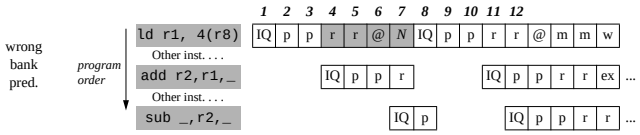

Fig. 6. Bank misprediction and Chained Recovery.

Figure 6 shows how CR acts on the code of Figure 4b. In the first recovery cycle (cycle *7*), the results of the mispredicted load and the Dependent Set are tagged as not available. However, in the same cycle the Issue Logic selects an instruction (*sub*) which depend on another one that is being nullified (*add*). Next, in cycle *8*, CR identifies which instructions issued in the previous cycle depends on the mispredicted load. Finally (cycle *9*), the results of the instructions identified by CR in cycle *8* are tagged as not available.

From now on, in every cycle (cycles *10*, *11*, ...) the two actions stated above are made at once: *a)* the results of the instructions identified as dependents in the previous cycle are tagged as not available, and *b)* those instructions issued in the previous cycle and depending on nullified instructions are identified. The recovery process will eventually finish in the cycle where no issued instruction depends on any nullified instruction.

Adding a bit vector to the baseline recovery mechanism described in [16] is enough to implement CR. This bit vector check whether the instructions issued in a cycle depend or not on instructions nullified in the previous cycle.

Evaluation. Figure 7 shows results again for the OCP and BASE processors considered in Section 4. The central bar in each bar group represents a processor enhanced with CR. As expected, the misspeculation penalty paid for CR is quite small -remember that such a penalty is the (OCP-CR) difference. So, the work lost during the Speculative Window and the issue slots wasted during the chained recovery have almost no adverse effects on performance.

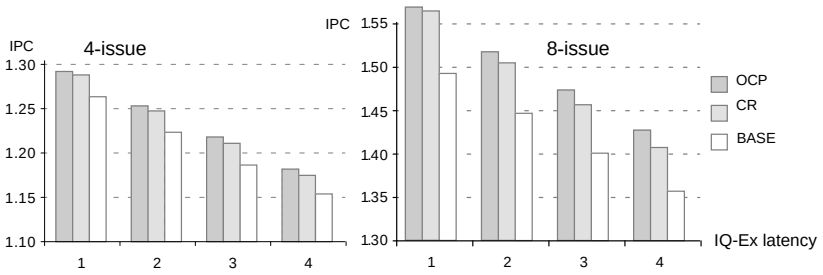

Fig. 7. Harmonic IPC mean for processors with Oracle Confidence Predictor (OCP), Chained Recovery (CR), and the actual predictor (BASE).

Summarizing, an 8-way processor with CR outperforms the baseline from 4.8% to 3.7% as the number of IQ-Ex stages grows from 1 to 4. In 4-way processors the benefits are lower, ranging from 1.9% to 1.8%.

6 Load Replication

As shown in Section 4, bank mispredictions increase the load-use delay and the effect is more pronounced as the number of the IQ-Ex stages increase. To reduce this penalty, instead of re-issuing from IQ, Neefs et al. propose using a dedicated network *after* the bank check. The network is out of critical timing paths and re-routes the bank-mispredicted instructions towards the correct bank [18].

Our point is quite different. We consider taking conservative actions *before* knowing about the outcome of the bank check. By conservative actions we understand issuing a load to several banks in limited cases. Next we show two approaches to this idea.

The first approach is *Replication Using Free Memory Slots* (RF). It consists in making use of idle memory ports. For that, after the selection phase in IQ the oldest load is identified among the selected instructions. Then, several instances of that load are issued to all free memory issue ports, if any. As usual, the dependent instructions can be speculatively scheduled. This alternative is similar to that proposed by Yoaz et al. However, since they assume a separate IQ for the memory instruction stream, the replication policy has no interaction with the integer IQ and the sliced memory system they propose is evaluated with an analytical model, but it does not include load replication [22].

The second approach consists in adding confidence to the bank prediction and whenever it is low, switch to a conservative dispatch policy that discards the prediction. We call this approach *Replication in Dispatch* (RD), and it is accomplished by tagging low-confidence loads in order to be issued to all the banks from the IQ. All load instances are not necessarily issued in the same cycle. Each port scheduler use instruction age to select, among the ready instructions, the one to be issued. Thus, RD does not disturb the issuing of older instructions with ready operands. Besides, when IQ is notified of the bank check, the load instances pending issuing are nullified. Because a replicated load has low bank confidence, we choose not to wake up the dependent instructions speculatively. The latest load instance or an early bank check will wake them up.

RD could produce a performance loss. The low-confidence loads replicated by RD account for a 43.7% (29%) of all loads for 4 (2) banks (see Table 1), and so, younger instructions could be stalled by lack of issue slots. Moreover, the low-confidence loads that were actually well predicted see their dependent instructions delayed until the last instance is issued. On the other hand, the low-confidence loads that were actually mispredicted (e.g. 21% for 4 banks) win the replication benefit: load-use delay decrease and misspeculation penalty removal. Summing up, the results below will show that the overall effect is clearly positive.

Evaluation. As said in Section 4, counteracting bank misprediction effects offers good opportunities for improving performance, and these opportunities grow as the number of IQ-Ex stages increases; see the gap between OBP and BASE bars in Figure 8. The three bars between OBP and BASE report the performance

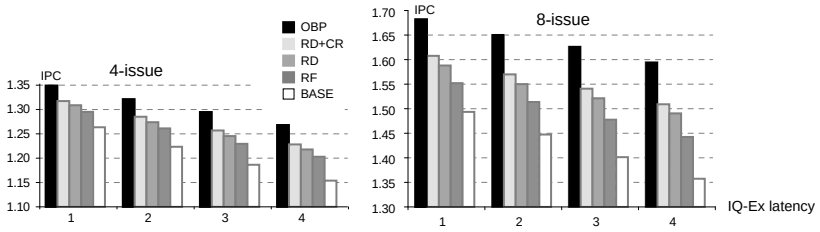

Fig. 8. Harmonic IPC mean. OBP means Oracle Bank Predictor. Replication in Dispatch (RD), Replication Using Free Memory Slots (RF) and Chained Recovery (CR).

of a selection of three enhancements of increasing complexity: RF, RD, and the combination of RD and Chained Recovery (RD+CR). Lets focus on the processors having 4 cycles from IQ to Ex and move then from the simplest alternative to the most complete. First we consider RF, which improves IPC 6.3% (4.3%) for an 8 (4)-way processor, respectively. Considering that RF adds a very small complexity in control, these are quite good numbers. Second, we consider the RD approach. We add confidence to the bank predictor by investing 6Kbits in storage, but IPC now rises to 9.8% (5.6%) for an 8 (4)-way processor, respectively. Third, if we add CR on top of RD, we are simultaneously decreasing the misspeculation penalty and the average load-use delay, achieving then 11.1% (6.5%) IPC increase for the 8 (4)-way processor, respectively (see the RD+CR bar). This extra performance comes only from the slight increase in control complexity added by CR.

As a final remark, it's important to realize that for all IQ-Ex numbers and for all issue widths, RD+CR achieves two thirds of the ideal performance given by OBP. This makes RD+CR very sound across important microarchitectural parameters and therefore an appealing cost/performance design point.

7 Conclusions

Current trends in microarchitectural design, such as issue width increase and deeper pipelining, have a negative influence on processors having sliced memory pipelines, which is increasingly dependent on the bank misprediction rate. In this paper we have introduced two complementary approaches to reduce the overall bank misprediction penalty.

The first one is *Chained Recovery*, a new recovery mechanism intended to reduce the recovery penalty after a misspeculation. We have shown that CR is very effective, achieving in the worst case 98.6% of the IPC given by an Oracle Confidence Predictor.

The second approach is a set of two specific policies to issue a load to several banks in limited cases. *Replication Using Free Memory Slots* replicate a load to free banks when there are free issue slots at issue time. It achieves a maximum 6.3% increase in performance adding only a modest control complexity. On the other hand, *Replication in Dispatch* replicates to all banks those loads with a low-confidence prediction. The increase in performance achieves 9.8% in the deepest pipelined 8-way processor, with an affordable increase in the predictor size (6K bits for our bank predictor).

Finally, if we put Chained Recovery and Replication in Dispatch together, we eliminate about two thirds of the IPC lost in bank mispredictions for all the simulated processors.

References

1. V. Agarwal et al.: Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. Proc. of 27th ISCA (2000) 248–259
2. E. Borch, E. Tune, S. Manne, and J. Emer: Loose Loops Sink Chips. Proc. of 8th HPCA (2002) 299–310
3. D.C. Burger and T.M. Austin: The SimpleScalar Tool Set, Version 2.0. UW Madison Computer Science Technical Report #1342 (1997)
4. B. Case: Intel Reveals Pentium Implementation Details. MPR vol 7, n 4 (1993) 1–9
5. S. Cho, P. Yew, and G. Lee: A High-Bandwidth Memory Pipeline for Wide Issue Processors. IEEE Trans. on Computers, vol. 50, no. 7 (2001) 709–723
6. J. Cortadella and J.M. Llabería: Evaluation of A+B=K Conditions without Carry Propagation. IEEE Trans. on Computers, vol. 41, no. 11 (1992) 1484–1488
7. J. Edmondson, P. Rubinfeld, and V. Rajagopalan: Superscalar Instruction Execution in the 21164 Alpha Microprocessor. IEEE Micro, vol. 15, no. 2 (1995) 33–43
8. J.A. Farrell and T.C. Fisher: Issue Logic for a 600 Mhz Out-of-Order Execution Microprocessor. IEEE J. of Solid-State Circuits, vol. 33, no. 5 (1998) 707–712
9. L. Gwennap: IBM Regains Performance Lead with Power2. MPR,vol.7,no.13(1993)
10. G. Hinton et al.:The Microarchitecture of the Pentium 4 Processor. ITJ Q1(2001)
11. P. Hsu: Design of the R8000 Microprocessor. IEEE Micro, vol.14 (1994) 23–33
12. R. Kessler: The Alpha 21264 Microprocessor. IEEE Micro,vol.19, no.2(1999)24–36
13. A. Kumar: The HP PA-8000 RISC CPU. IEEE Micro, vol. 17, no.2 (1997) 27–32
14. M. Matson et al.: Circuit Implementation of a 600MHz Superscalar RISC Microprocessor. Proc. of ICCD (1998) 104–110
15. P. Michaud, A. Seznec, and R. Uhlig: Trading Conflict and Capacity Aliasing in Conditional Branch Predictors. Proc. of 24th ISCA (1997) 292–303
16. E. Morancho, J.M. LLabería, and A. Olivé: Recovery Mechanism for Latency Misprediction. Proc. of PACT (2001) 118–128
17. S. Naffziger et al.: The implementation of the Itanium 2 Microprocessor. IEEE J. Solid State Circuits, vol. 37, no. 11 (2002) 1448–1460
18. H. Neefs, H. Vandierendonck, and K. De Bosschere: A Technique for High Bandwidth and Deterministic Low Latency Load/Store Accesses to Multiple Cache Banks. Proc. of 6th HPCA (2000) 313–324
19. A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides: Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor. Proc. of 29th ISCA (2002) 295–306
20. T. Sherwood, E. Perelman, G. Hamerly, and B. Calder: Automatically Characterizing Large Scale Program Behaviour. Proc. of ASPLOS (2002)
21. G.S. Sohi and M. Franklin: High-Bandwidth Memory Systems for Superscalar Processors. Proc. 4th ASPLOS (1991) 53–62
22. A. Yoaz, E. Mattan, R. Ronen, and S. Jourdan.: Speculation Techniques for Improving Load Related Instruction Scheduling. Proc. of 26th ISCA (1999) 42–53
23. V. Zyuban and P.M. Kogge: Inherently Lower-Power High-Performance Superscalar Architectures. IEEE Trans. on Computers, vol. 50, no. 3 (2001) 268–285