Memory Address Prediction for Data Speculation *

José González and Antonio González

Departament d'Arquitectura de Computadors
Universistat Politècnica de Catalunya
C. Jordi Girona 1-3, 08034 Barcelona (Spain)
Email: {joseg,antonio}@ac.upc,es
Tel:+ 34 4 4016988
Fax: + 34 4 4017055

Abstract. Data speculation refers to the execution of an instruction before some logically preceding instructions on which it is data dependent. Data speculation implies some form of prediction of the data required by the speculative executed instruction and a recovering mechanism in case of misspeculation. This paper shows that load/store instructions are very good candidates for speculative execution since their effective address is highly predictable. We propose a novel technique called Memory Address Prediction (MAP) that implements speculative execution of load/store instructions in an out-of-order processor. The cost of this mechanism is mainly the addition of an address prediction table since the misprediction recovery hardware is already present in many current microprocessors for other purposes. The mechanism is evaluated for the SPEC95 benchmark suite showing significant performance gains.

1 Introduction

The performance of current processors heavily rely on the exploitation of Instruction Level Parallelism (ILP). The effectiveness of this technique is limited by the necessity to obey the data dependences existing among instructions.

There have been very few proposals trying to overcome the limitations imposed by having to obey data dependences. In the same way as control dependent instructions can be speculative executed before the branches on which they depend, data dependent instructions can do so. What is needed for the latter is the ability to predict the value required by the data dependent instruction and a recovery mechanism for misspeculated instructions.

In this paper, we propose a data speculation mechanism that is based on the observation that the source operands of load/store instructions are highly predictable.

* This work has been supported by the Spanish Ministry of Education under grant CYCIT TIC 429/95 and the Direcció General de Recerca of the Generalitat de Catalunya under grant 1996FI-03039-APDT.

The proposed Memory Address Prediction mechanism (MAP) identifies which load/store instructions are highly predictable and issues them speculatively, as well as those instructions that depend on them. In case of misprediction, the misspeculated instructions are re-executed. The mechanism is evaluated for an out-of-order processor with two different memory ordering techniques: a conventional total disambiguation scheme and a more aggressive partial disambiguation mechanism.

2 Data speculation and related work

Data prefetching schemes [2] have certain similarities with the MAP in the sense that they predict the effective address of future load/store operations and bring the data into cache if not yet present. However, data prefetching does not execute any instruction speculatively.

Data speculation is a family of techniques that try to avoid the ordering imposed by data dependences. Data speculation allows the speculative execution of some instructions before some other instructions on which they are data dependent. Data speculation could be applied to values that flow either through memory or registers.

The most remarkable proposals on data speculation are: [1][3][5][9] [10][11]. The main differences between the mechanism proposed in this paper and previous work are:

- The mechanism proposed is this work predicts the address of memory instructions as [1] [3][5]. However in our proposal the instructions that depend on the predicted load are issued speculatively meanwhile such previous proposals do not perform speculative execution of those instructions.
- In [11] the effective address of load instructions is predicted and the load and subsequent dependent instructions are speculatively issued as is done in our work. But in our proposal the effective address of store instructions is also predicted. Besides, in [11] a perfect memory disambiguation scheme is considered, whereas in this paper we study the performance of address prediction for two realistic memory disambiguation schemes, where the prediction of the effective address of stores plays an important role in order to achieve an significant performance improvement.
- Regarding [9][10] the difference with our method is that their predict the result of an operation whereas we speculate with load/store instructions predicting their effective address. In addition, we will show in this paper that memory addresses are more predictable than memory values.

3 Motivation

In [6] it is shown that even with unlimited resources and perfect control speculation, the performance of current architectures would not be much higher than a hundred IPC (instructions per cycle) for many programs, and in some cases it

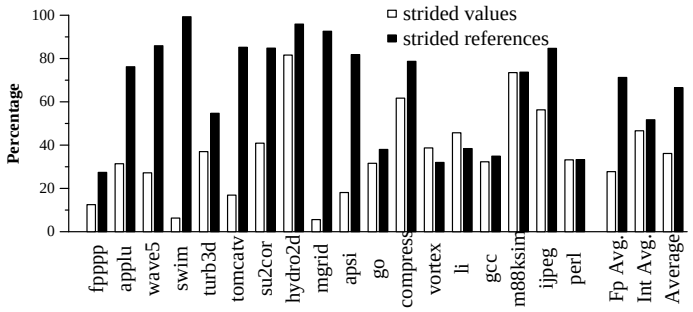

Fig. 1. Percentage of strided references compared with percentage of strided values.

would be just a few tens of IPC. Data speculation allows to go beyond this barrier. Data speculation is a powerful technique to increase the ILP of a program.

The mechanism proposed in this paper is based on the observation that the effective address of many load/store instructions are highly predictable. In particular, it tries to identify those load/store instructions whose effective address in successive executions is either the same or equal to the previous one plus a constant value. We will refer to both as strided references.

To validate our hypothesis on the predictability of load/store effective address we have run all the SPEC95 benchmarks and measured the percentage of strided references. The experimental framework is described in section 5.1. For the results in this section, each benchmark was run until the first billion load instructions or until completion if it happened earlier. Strided references are captured by means of a history table with 1K entries. This table is direct-mapped, non-tagged and it is indexed with the ten least-significant bits of the load/store instruction address. For each entry, the table stores the last effective address and the last observed stride.

Figure 1 shows the percentage of dynamically executed load/store instructions that exhibit strided references. It can be seen that strided references are very common in the SPEC95 suite: In average, they represent about 70% of all memory references. In both integer and floating point applications strided references are very frequent although the percentage is higher in floating-point applications. For comparison, Figure 1 also shows the performance of the load value prediction scheme proposed in [9], extended to account for strided values for both load and store instructions. Although strided values are quite frequent they are much less common than strided references. We conclude that memory addresses are more predictable than memory values.

4 Memory Address Prediction

The fact that the effective address of memory references is highly predictable can be used to speedup processor performance in different ways (e.g. prefetching

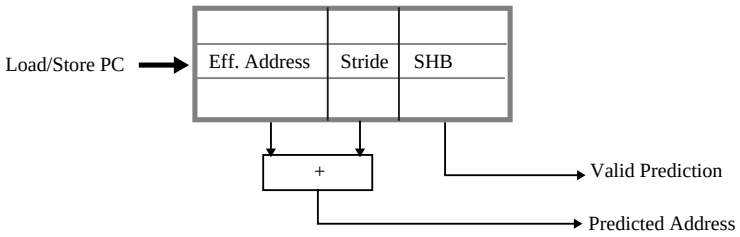

Fig. 2. Memory History Table (MHT)

mechanisms). We propose to use memory address prediction to speculate on the effective address of unresolved load/store instructions and execute them speculatively as well as the instructions that depend on them.

We have investigated the use of memory address prediction in the context of an out-of-order execution processor with in-order retirement to support precise exceptions [6]. Two different memory ordering mechanisms have been considered:

- Total disambiguation: In this scheme, a load can be issued as soon as there are not preceding stores with unknown effective address and the load operands are available. A store is issued as soon as its effective address can be computed and all previous loads and stores have already been issued.
- Partial disambiguation: This memory ordering scheme is based on the mechanism implemented in the HP PA8000 [4] and the ARB proposed in [7]. In this scheme a load or store can be issued as soon as its operands are ready, without being fully disambiguated with previous references. When a store founds that a load has already been performed to the same address from a succeeding instruction with no intervening stores in between, a recovery action is initiated.

In both schemes data can be forwarded from a previous resolved store to a subsequent dependent load.

The MAP mechanism is implemented by means of a table called Memory History Table (MHT), as it is shown in Figure 2. It is a 1024-entry, direct-mapped table that is indexed with the least-significant bits of the instruction address and does not contain tags. Each entry stores the following information:

- Effective Address: This is the last effective address seen by that load/store instruction.
- Stride: This field contains the last stride observed for that load/store instruction. The length of this field is 4 bytes.
- Stride History Bits (SHB): This field is used to assign confidence to the prediction. It is implemented by means of a two-bit up-down saturated counter. The prediction is determined by the most-significant bit of this field.

The MAP works as follows: At the decode stage, the corresponding MHT entry is read and the predicted effective address is computed. If the SHB most

Functional Unit	Number	Latency	Repeat Rate
Simple Integer	1	1	1
Complex Integer	1	9 multiply	9
		67 Divide	64
Effective Address	2	1	1
Simple FP	1	2	1
FP Multiplication	1	2	1
FP Divide and SQRT	1	21 divide	21
		35 SQRT	35

Table 1. Functional Units and instruction latency.

significant bit is set, then a correct prediction is assumed and the instruction is considered to be ready-to-issue to the load/store buffer.

Speculatively executed load/stores must be verified. This is done by issuing them to the address computation unit when their operands are ready. The computed effective address is compared against the predicted effective address and in case of a mismatch a recovering action is initiated. At this time the SHB field is updated and in case of a misprediction, the stride field is set to the new value.

The recovery for mispredicted loads and stores is implemented in a different way. For loads, we assume the recovering mechanism proposed in [9]. When a load is mispredicted, all subsequent instructions that depend on such load are re-executed. This mechanism cannot be used for stores because the destination of a store is not a register but a memory location, and in consequence, dependent instructions are not known until the correct address is calculated. For mispredicted stores we assume the same recovery mechanism used to recover from mispredicted branches (a pipeline flush).

Since the store misprediction penalty is higher than that of loads, a store instruction is executed speculatively only when its operands are unknown when it is decoded.

5 Performance evaluation

This section studies the effectiveness of the MAP in the context of a superscalar out-of-order execution processor for the two memory ordering schemes discussed in the previous section.

5.1 Experimental Framework

We have developed a simulator of a superscalar processor with out-of-order execution that resembles some of the latest microprocessors. The execution of an instruction consists of the typical stages: fetch, decode, issue, execute, write-back and graduate (or commit). Branch Prediction is performed by means of a

	No MAP		MAP	
	Total disambiguation	Partial disambiguation	Total disambiguation	Partial disambiguation
104.hydro2d	1.14	1.21 (6.1%)	1.21 (6.1%)	1.26 (10.5%)
107.mgrid	1.68	1.68 (0%)	1.82 (8.3%)	1.82 (8.3%)
110.applu	1.20	1.20 (0%)	1.22 (1.7%)	1.23 (2.5%)
146.wave5	0.95	0.99 (4.2%)	1.02 (7.4%)	1.02 (7.4%)
124.m88ksim	0.91	0.91 (0%)	1.14 (25.3%)	1.18 (29.7%)
129.compress	1.15	1.28 (11.3%)	1.31 (13.9%)	1.31 (13.9%)
130.li	0.92	1.02 (10.9%)	1.03 (12.0%)	1.06 (15.2%)
134.perl	0.91	1.03 (13.2%)	0.98 (7.7%)	1.06 (16.5 %)
Avg. improvement		5.7%	10.3%	13.0%

Table 2. Instruction completion rates. In brackets it is shown the percentage improvement over the total disambiguation scheme with no memory address prediction.

2048 entry Branch History Table with a 2 bit up-down saturated counter per entry.

The results presented in this section assume the following configuration. The size of the reorder buffer is 32 entries. There are two separate physical register files for integer and FP data. The size of both files is 64 registers. The processor has a lookup-free data cache [8] that allows up to 16 pending misses to different cache lines. The cache size is 8Kb, and it is direct-mapped with 32-byte line size. Cache hit latency is 2 cycles and the penalty for a cache miss is 20 cycles. An infinite L2 cache is assumed. Table 1 shows the number of functional unit and their latency.

Our experimentation methodology is trace-driven simulation. The object code, previously compiled with full optimization for a DEC AlphaStation 600 5/266 with and Alpha AXP 21164 processor, is instrumented using the Atom tool [12]. Because of the detail at which simulation is carried out the simulator is slow, so we have simulated 50 million of instructions for each benchmark after skipping the first 100 million of instruction. Eight SPEC'95 benchmarks (4 FP and 4 Integer) has been selected for this study.

5.2 Results

Table 2 shows the IPC (instructions committed per cycle) achieved by the two memory disambiguation schemes with and without the MAP mechanism proposed in this paper. It is shown in brackets the percentage improvement over the total disambiguation scheme without memory address prediction. In average, the improvement achieved by MAP is around 10

The improvement due to MAP depends on a number of issues in addition to the percentage of strided accesses. It depends on the accuracy of the memory address predictor. The data in Figure 3a represents the percentage of memory ref-

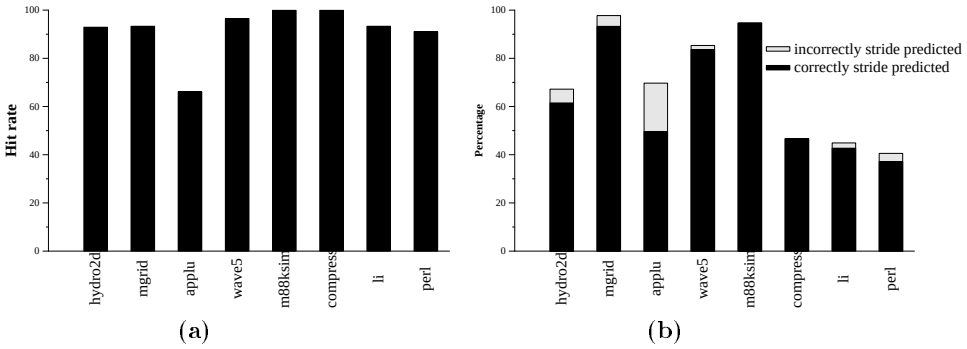

Fig. 3. Percentage of: a) correctly predicted loads and stores. b) correctly predicted loads

erences that are correctly predicted either as strided references or as not strided references. Obviously, a better prediction will imply more accurate speculation and less number of mispredictions. The performance also depends highly on the percentage of memory references that are predicted to be strided references and the number of these predictions that are correct. This is shown in Figure 3b for loads. A higher percentage of strided references means more speculation. On the other hand, a higher number of misprediction implies a higher penalization due to recovering.

For instance, 110.applu is the program that experiences the lowest improvement with the MAP. In Figure 1, it can be observed that it has a quite high number of strided accesses but the predictor exhibits a rather low hit ratio (Figure 3a). In particular, this translates in a high percentage of loads that are predicted to be strided but they are not actually, as it can be seen in Figure 3b.

On the other hand, 124.m88ksim shows the highest improvement. This benchmark has a high percentage of strided accesses, most of them predicted correctly as it is shown in Figure 3b.

We can conclude that memory address prediction for data speculation can be an interesting mechanism to be included in future microprocessors as a way to overcome the ordering imposed by true dependences. Its hardware cost is not negligible mainly because of the large memory history table that it requires but it may be affordable for next generation machines.

The objective of this paper is to demonstrate the potential benefits of a new technique for data speculation. Detailed evaluation of different prediction schemes and their associated hardware cost is not the aim of this paper.

6 Conclusions

We have presented a novel technique for data speculation based on the observation that the effective address of many load/store instructions is highly predictable. By predicting their address, these instructions and the instructions

dependent on them can be executed speculatively before their source operands are known, allowing the processor to go beyond the limits imposed by having to obey the true dependences.

We have observed an average performance gain of about 10%, and in some cases it was much higher (up to 30%). We have also shown that memory addresses are more predictable than memory values. Thus speculation based on memory addresses may be more effective than speculating on memory values. However, both techniques could be combined to performed a more aggressive speculation. We plan to investigate this issue as an extension to this work. We have also shown that data speculation based on memory address prediction is more effective than speculation based on partial memory disambiguation and both types of speculation can be combined to obtain an average gain of about 13%.

References

1. T.M. Austin, G.S. Sohi Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency. Proc. of Int. Symp. on Microarchitecture, pp 82-92, 1995.
2. T-F. Chen and J-L. Baer: A Performance Study of Software and Hardware Data Prefetching Schemes Proc of the Int. Symp. Computer Architecture, pp. 223-232, 1994.
3. R.J. Eickemeyer and S. Vassiliadis: A Load Instruction Unit for Pipelined Processors. IBM Journal of Research and Developement, 37(4), pp. 547-564, July 1993
4. M. Franklin and G.S. Sohi: ARB: A Hardware Mechanims for Dynamic Reordering of Memory References IEEE Transactions on Computers, 45(6), pp. 552-571, May 1996.
5. M.Golden and T.N. Mudge:Hardware Support for Hiding Cache Latency. Technical Report #CSE-TR-152.93. University of Michigan, 1993.
6. J.L Hennessy and D.A. Patterson: Computer Architecture. A Quantitative Approach. Second Edition. Morgan Kaufmann Publishers, San Francisco 1996.
7. D. Hunt: Advanced Performance Features of the 64-bit PA-8000 Proc. of the CompCon'95, pp. 123-128, 1995.
8. D. Kroft: Lockup-free Instruction Fetch/Prefetch Cache Organization Proc. of the Int. Symp. on Computer Architecture, pp. 81-87, May 1981.
9. M.H. Lipasti, C.B. Wilkerson and J.P. Shen: Value Locality and Load Value Prediction Proc. of the 7th. ACM Conf. on Architectural Support for Programming Languages and Operating Systems, Oct. 1996.
10. M.H. Lipasti and J.P. Shen: Exceeding the Dataflow Limit via Value Prediction. Proc. of Int. Symp. on Microarchitecture, 1996
11. Y. Sazeides, S. Vassiliadis and J.E. Smith: The Performance Potential of Data Dependence Speculation & Collapsing. Proc. of Int. Symp. on Microarchitecture, December 1996.
12. A. Srivastava and A. Eustace: ATOM: A system for building customized program analysis tools Proc of the 1994 Conf. on Programming Languages Design and Implementation, 1994.