Memory Bank Predictors

Stefan Bieschewski[1], Joan-Manuel Parcerisa[1], and Antonio González[1,2]

[1]*Departament d'Arquitectura de Computadors*
Universitat Politècnica de Catalunya
Barcelona, Spain

[2]*Intel Barcelona Research Center*
Intel Labs, Universitat Politècnica de Catalunya
Barcelona, Spain

{sbiesche,jmanel,antonio}@ac.upc.es

Abstract

Cache memories are commonly implemented through multiple memory banks to improve bandwidth and latency. The early knowledge of the data cache bank that an instruction will access can help to improve the performance in several ways. One scenario that is likely to become increasingly important is clustered microprocessors with a distributed cache. This work presents a study of different cache bank predictors. We show that effective bank predictors can be implemented with relatively low cost. For instance, a predictor of approximately 4 Kbytes is shown to achieve an average hit rate of 78% for SPECint2000 when used to predict accesses to an 8-bank cache memory in a contemporary superscalar processor. We also show how a predictor can be used to reduce the communication latency caused by memory accesses in a clustered microarchitecture with a distributed cache design.

Keywords: Memory bank prediction, clustered microarchitectures.

1. Introduction

The quest for higher levels of parallelism in processor architecture has brought in the past decade growing levels of complexity - and longer access times - to many processor components. Moreover, with current trends towards deeper pipelines and faster clock speeds, the time in clock cycles to access on-chip memory structures is quickly increasing. At the same time, as process technologies continue to shrink, wire delays become dominant, compared to gate delays, which makes access latencies grow even faster [1].

One way to cope with these challenges is to partition data path components. Partitioning is the basis of clustered microarchitectures. By partitioning subsystems and placing them in clusters, their complexity, and therefore their latency, and power requirements decrease and are more easily managed.

Many recent clustered microarchitecture proposals assume a centralized L1 data cache and disambiguation hardware. Since the cache cannot be close to all clusters, the cache access latencies experienced by distant clusters may be very high due to wire delays. In addition, the disambiguation hardware may be complex due to its associative nature and its size. By partitioning the L1 data cache into disjoint banks and placing one bank close to each cluster, access latency can be greatly reduced.

To minimize communication latency, the cluster assignment of a memory instruction (which occurs early in the pipeline), needs to know the bank it will access, but the address is not resolved until late in the pipeline. Hence, most clustered architectures with a distributed L1 data cache include a bank (or address) predictor, in order to guide the steering decisions for memory instructions [2][8][11]. Of course, cache access latency—hence performance—depends on the accuracy of the bank predictor.

We find that some of the proposed predictors achieve better accuracies than those reported so far in the literature [2][10][11]. Although not evaluated, results are expected to be even better for FP programs due to their more regular memory access patterns.

2. Bank Predictors

Many of the techniques used for branch and value prediction can also be applied to bank prediction. There are however some differences, like the range of the values being predicted.

Below, we propose a number of bank predictors that are inspired by schemes previously proposed for branches and values.

The *last bank predictor* has a single lookup table that is indexed by the least-significant bits of the program counter and contains the identifiers of the last bank accessed. This lookup table is referred as pattern history table or PHT for short.

Global history predictors keep the identifiers of the most recently accessed banks in a shift register. When a new identifier is shifted in, the oldest one is discarded. The contents of this shift register will be referred to as *global history* because it may stem from different static instructions. The global history is used as the index into the PHT and the contents of the corresponding PHT entry are used to produce the prediction.

The *Global* predictor uses just the contents of the global history register to index the PHT. Thus, the length of the global history register depends on how many bits are needed to index all entries in the PHT.

Gshare performs a bitwise XOR between the global history register and the least-significant bits of the program counter to obtain the index into the PHT. The optimal history size for Gshare depends on the size of the PHT. We use the history size that gives the best overall result for each PHT size.

Local history predictors use a second table to store local histories. This table is indexed by the least-significant bits of the program counter. The value obtained in this way is used to index the PHT. Because bank identifiers consist of only a few bits, hashing the local history [9] does not improve the accuracy of local bank predictors, as we have experimentally confirmed. There are several possibilities to split the available transistor budget between the two predictor tables. We have found that this division has little effect on prediction accuracy.

Instead of predicting banks directly, *stride predictors* predict strides. The final prediction is obtained by adding the predicted stride to the last bank accessed. The first stride predictor we examine is based on the last bank predictor. To make the algorithm more robust against occasional mispredictions we use the two-delta method: a stride must occur at least twice in a row before it is used in future predictions. The second alternative for a stride predictor that we consider consists in predicting strides with a local predictor. [6]

Tournament predictors contain two predictors that are used in the conventional way to predict values and a third meta predictor that chooses the final result from the two other predictors. Each entry of the meta predictor contains a saturating counter that is updated if only one predictor predicts correctly. In this paper we examine a combination of a Local and a Gshare predictor.

Voting is another way to combine predictors. The most popular prediction is chosen. However, when predicting banks tie can occur. We solve this problem by giving fixed priorities to the predictors.

In this paper *Gskew* represents the family of voting predictors.

3. Evaluation

In this section we evaluate several bank predictors. We evaluate each of the above described predictors using a simplified model. In this model we ignore the delays of a realistic pipeline and assume that once a prediction is made, the correct prediction is immediately known and the predictor updated.

We assume a first level data cache divided into eight quadword-interleaved banks. The bank number is determined by the bits 3:5 of the data address. The SPECint2000 benchmarks are used to evaluate the bank predictors. The benchmarks are compiled with Compaq's C Compiler with maximum optimization level for the Alpha 21264. The programs are simulated with the *functional* simulator of the SimpleScalar 3.0 toolset [4]. We skip the initialization phase of the benchmarks and record the first 100 million of memory references.

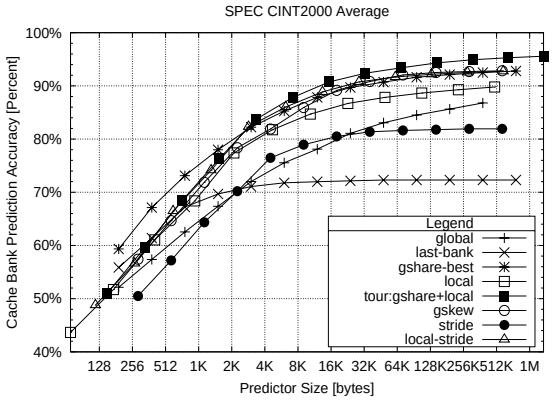

Figure 1: SPECint2000 with ideal

Figure 1 shows the prediction accuracy of the different predictors as a function of the predictor size. For predictor sizes below 2 Kbytes, Gshare achieves the best results. Above 2 Kbytes the tournament predictor obtains the highest accuracy. A tournament predictor of a reasonable size of 4 Kbytes can achieve 78% accuracy.

4. Case Study: Distributed Cache in a Clustered Microarchitecture

To demonstrate one potential use of the predictors described above, in this section we show how a bank predictor can be used to improve performance in a clustered microarchitecture with a distributed cache (see Figure 2).

For the following experiments we assumed a clustered microarchitecture with distributed register file, based on the one described by Parcerisa [7]. This architecture consists of a conventional front-end and a clustered back-end, each cluster containing its own issue queue and a set of functional units and physical registers. After renaming, each instruction is steered to the cluster where it will be executed, following a dependence based scheme.

We also distribute the first level data cache and the load store queues. Since L1 data is address-interleaved the target bank of a load is not known until late in the pipeline. A bank predictor is employed to steer load instructions to clusters. A failed prediction incurs an additional delay for obtaining the data from another cluster. Thus, a good bank predictor helps to reduce inter-cluster communication and load latency. A more detailed description of this architecture can be found in [3].

We choose the most promising bank predictor from the previous section. This predictor is a tournament predictor consisting of a Gshare with a history of one access and a

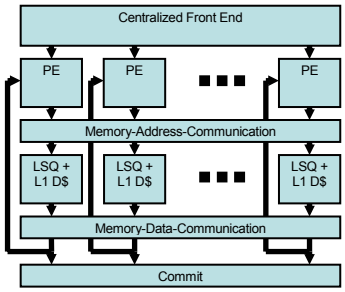

Figure 2: Pipeline for Memory Instructions

Fetch/Decode Width	16
ROB Size	256
Number of PE	8
Integer Issue Queue Size	16 per PE
Floating Point Issue Queue Size	16 per PE
Issue Width Integer	2 per PE
Issue Width Floating Point	1 per PE
Physical Registers	56 per PE
Level 1 Data Cache Size	64 Kbytes
Level 1 Data Cache Latency	3 Cycles
Intercluster Network Topology	Asynchronous Ring
Intercluster Network Latency	1 Cycle per hop

Table 1: Main architectural parameters

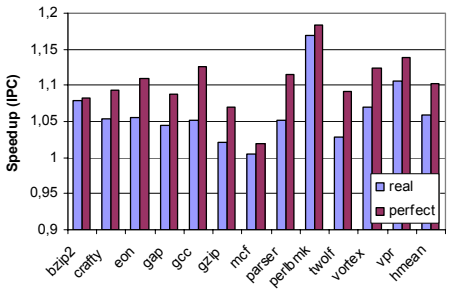

Figure 3: IPC relative to baseline

PHT with 2048 entries and a Local-Stride predictor with a history table of 2048 entries and a PHT with 256 entries. The total memory of the predictor amounts to 4 Kbytes.

In the following experiments we compare our architecture to the baseline (which has no bank predictor) and also include results for a perfect predictor to show the potential of bank prediction.

The simulations have been carried out on a modified version of sim-outorder [4] simulating 100 million instructions from each SPECint2000 benchmark, after skipping the initialization phase. The benchmarks are the same as in the previous section. A summary of the main architectural parameters of our clustered architecture is given in table 1.

As shown in Figure 3, the proposed bank predictor consistently improves performance for all the studied benchmarks, with an average 6% speedup. *Mcf* shows negligible improvements due to the high number of L2 cache misses. On the other hand, *perlbmk* experiences an impressive 18% speedup. On average, the inter-cluster communication latency of load data is reduced by 70%.

5. Conclusions

We have presented and discussed an ample number of bank predictor techniques inspired on existing branch and value predictors. They have been evaluated over a wide range of sizes, and they are compared on the basis of equal capacities. Our results have identified an effective 4Kbytes sized predictor that achieves 78% accuracy for the SPECint2000 when used to predict the accesses to an 8-bank cache in a contemporary superscalar processor. This predictor significantly outperforms previously reported bank predictors. [2][10][11]

We have also presented a case study to demonstrate the utility of bank predictors on clustered microarchitectures with a distributed L1 data cache and disambiguation hardware. We have shown that with an accurate bank predictor the access latency of loads is greatly reduced, and performance is significantly improved.

Acknowledgements

This work is supported by the Spanish Ministry of Science and Technology and FEDER funds of the EU under contracts TIN 2004-03072, and TIN 2004-07739-C02-01, and Intel Corporation.

References

[1] V. Agarwal, et. al. “Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures.” ISCA-27, 2000.

[2] R. Balasubramonian, et. al. “Microarchitectural Trade-offs in the Design of a Scalable Clustered Microprocessor,” Technical Report, University of Rochester, 2003.

[3] St. Bieschewski, el. al. “Memory Bank Predictors,” Technical Report, Universitat Politècnica de Catalunya, 2005.

[4] D. Burger, et. al. “Evaluating Future Microprocessors: The SimpleScalar Tool Set,” Tech. Report CS-TR-96-1308, University of Wisconsin-Madison, 1996.

[5] B. Goeman, et. al. “Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency,” HPCA-01, 2001.

[6] H. Neefs, et. al. “A Technique for High Bandwidth and Deterministic Low Latency Load/Store Access to Multiple Cache Banks,” HPCA-00, 2000.

[7] J. M. Parcerisa, “Design of Clustered Superscalar Microarchitectures”, Ph.D. Thesis, Universitat Politècnica de Catalunya, 2004.

[8] P. Racunas and Y. N. Patt, “Partitioned first-level cache design for clustered microarchitectures,” ICS-03, 2003.

[9] Y. Sazeides, and J. E. Smith, “Implementations of Context Based Value Predictors,” Technical Report, ECE-97-8, University of Wisconsin-Madison, 1997.

[10] A. Yoaz, et. al. “Speculation Techniques for Improving Load Related Instruction Scheduling,” ISCA-26, 1999.

[11] V. V. Zyuban, P. M. Kogge, “Inherently Lower-Power High-Performance Superscalar Architectures”, IEEE Transactions on Computers, March, 2001, Vol. 50, No. 3.