Branch Prediction Using Profile Data*

Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero

Universitat Politecnica de Catalunya
Jordi Girona 1–3, D6, 08034 Barcelona, Spain
{aramirez,larri,mateo}@ac.upc.es

Abstract. Branch prediction accuracy is a very important factor for superscalar processor performance. It is the ability to predict the outcome of a branch which allows the processor to effectively use a large instruction window, and extract a larger amount of ILP.
The first approach to branch prediction were static predictors, which always predicted the same direction for a given branch. The use of profile data and compiler transformations proved very effective at improving the accuracy of these predictors.
In this paper we propose a novel dynamic predictor organization which makes extensive use of profile data. The main advantage of our proposed predictor (the *agbias* predictor) is that it does not depend heavily on the quality of the profile data to provide high prediction accuracy.
Our results show that our *agbias* predictor reduces the branch misprediction rate by 14% on a 16KB predictor over the next best compiler-enhanced predictor.

1 Introduction

Branch prediction was first approached using static schemes, which always predict the same direction for a branch, like predicting that all branches would be taken, or that only backwards branches would be taken [23]. Semi-static branch predictors use profile feedback information and encode the most likely branch direction in the instruction opcode, obtaining much higher accuracy than the simple static predictors [9]. Profile data was also used to align branches in the code so that they follow a given static prediction heuristic [3]. Finally, the more accurate dynamic branch predictors, which store the past branch behavior in dynamic information tables, and lookup this data to predict the future branch direction [12,16,18,23,24,25].

We believe this paper contributes in proposing a dynamic prediction scheme which makes extensive use of profile data to provide higher prediction accuracy, even with lower quality or inexact profile data.

The proposed *agbias* predictor is largely based on the static-dynamic predictor combination proposed in [20], and divides branches among four sub-streams: first, among *easy* and *hard* to predict branches; second, among mostly *taken*

* This work was supported by the Ministry of Education and Science of Spain under contract TIC-0511/98, by CEPBA, and an Intel fellowship.

© Springer-Verlag Berlin Heidelberg 2001

and *not taken* branches. This division obtains a significant interference reduction, and allows a separate resource allocation, as not all sub-streams have the same needs.

Our results show that the *agbias* predictor outperforms all other examined predictors (including their compiler enhanced version), reducing branch misprediction rate by 14% on a 16KB predictor.

1.1 Simulation Setup

All the results in the paper were obtained using a simulator derived from the SimpleScalar 3.0 tool set [2]. We run most of the SPECint95 benchmarks (except go, compress, and perl) plus the PostgreSQL 6.3 database system running a subset of the TPC-D queries. All programs were compiled statically and with -O4 optimization level using Compaq's C compiler.

The training and simulation inputs were different to simulate the effect of inaccurate profile data on the prediction accuracy. All simulations were run to completion. All figures in the paper present the arithmetic average of all executed benchmarks, where all codes have the same weight.

We have compared our agbias predictor with the compiler-enhanced versions of most modern de-aliased predictors: a gshare predictor using an optimized code layout [22], agbias using a profiled bias bit [24], bimode using a profiled selector [12,10], gskew [16], and a static-dynamic hybrid using profile data [20].

All predictors simulated use global branch history. The BHR length determines the PHT size: for N bits of history, 2^N PHT entries are allocated. We have used a 4096-entry/4-way set associative BTB to store the bias bit in the agree predictor simulations.

1.2 Paper Structure

The rest of this paper is structured as follows: In Section 2 we present previous related work, and discuss its relevance to this paper. In Section 3 we propose our predictor organization, the *agbias* predictor, and analyze the reasons why it proves more accurate than others, showing that prediction table interference is not the only relevant factor. Finally, in Section 4 we present our conclusions for this work.

2 Related Work

We can classify related work into four groups: basic branch predictors, de-aliased predictors, and compiler support for branch prediction.

Basic branch prediction schemes can be broadly classified into three groups: static, semi-static and dynamic branch predictors.

Static predictors always predict the same outcome for a given branch, and can be based on very simple heuristics, which do not require encoded information for the processor [23]. They can also be based on compiler analysis and more complex heuristics [1] which are mainly used by the compiler itself to align branches following a more simple heuristic [3,8]. The compiler can also increase static prediction accuracy by using code transformations, usually implying code replication [11,17,21,26].

Semi-static predictors are based on the observation that branches tend to behave in the same way across different executions of the same code, and use profile information obtained at run-time [4,9,15]. These predictors predict that a branch will always follow its most usual direction as observed in the profile data. Thanks to the use of profile information from an adequate training input they achieve higher accuracy than simple static predictors based exclusively on static analysis and heuristics. But the wrong inputs used to obtain the profile data can lead to decreased accuracy of the predictor.

Dynamic branch predictors store information about the recent behavior of branches in a given execution of the program, and are predicted to behave in the same way as they usually did in the same past situation. These predictors differ mainly in the way they store the past behavior of the branch and the situation in which it executed, be it relative to the recent outcomes of other branches (global branch history, path correlation) or the past outcomes of the same branch (self history) [14,18,23,25].

All dynamic branch prediction schemes use finite tables to store the past behavior of branches. When two different branches store their information in the same table entry, aliasing happens. Recent dynamic prediction schemes try to organize their tables in a clever way to reduce destructive aliasing [7,12,16,24]. We refer to these predictors as de-aliased schemes.

There have been also some dynamic branch prediction studies which have involved the compiler in the prediction scheme, using profile information to replace some predictor components [10,13,20], or to provide some information that is better obtained statically [24,19].

An alternative way of reducing prediction table interference is to reduce the number of branches stored in these tables, filtering out the strongly biased branches, which would then be predicted with a single bit stored in a separate table [5,7,19] or statically using profile information [6,13,20]. We show that these schemes heavily depend on the accuracy of the profile data, and present an alternative organization which solves this problem, proving equally accurate with both self-train and cross-train tests.

In [20] the relevance of the static predictors is increased, as they are used to predict a large fraction of branches, removing them from the dynamic prediction tables, thus reducing interference, and increasing accuracy. But most of the results shown correspond to the self-trained case, where the profile data is totally accurate.

3 The Agbias Predictor

Based on the ideas exposed thus far, we propose another combination of static and dynamic branch predictors which we call *agbias*. The agbias prediction scheme is shown in Figure 1. Largely based on the prediction scheme proposed in [20], it is composed of two dynamic direction predictors, which use some de-aliasing mechanism (we have chosen the agree mechanism in this paper), and a static meta-predictor which divides branches between the strongly biased sub-stream (the *easy* branches), and the not so biased sub-stream (the *hard* branches). The *easy* branches are those which have the same outcome 95% of the times, that is, they almost never change direction. Both dynamic components share the BHR, which is only updated for the branches belonging to the *hard* sub-stream.

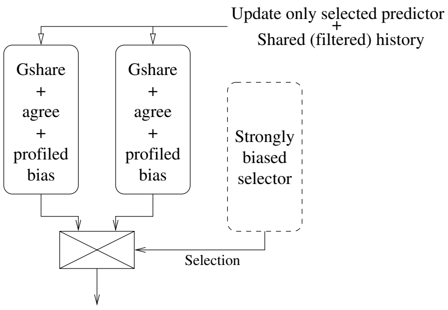

Fig. 1. The agbias predictor scheme

This way, we are dividing branches into four categories: first, using the profiled bias (strongly biased/easy *or* not strongly biased/hard branches); second, using their most likely outcome (taken *or* not taken). The first division is used to distribute branches in two separate dynamic predictors, which allows an independent resource allocation for the *easy* and *hard* sub-streams. The second division is used to minimize negative interference in the prediction tables of both dynamic components, using a de-aliased scheme (we have chosen the agree[p] scheme, hence the name *agbias*). The classification of branches among strongly biased and non-strongly biased using profile data [6] or dynamic tables [5,7,19] has been explored before, but none considered a separate dynamic component for the strongly biased stream.

The static-dynamic combination used in [20] does not allocate any dynamic resources to the *easy* sub-stream, relying entirely on the accuracy of the profile data. Our scheme avoids this too strong dependency by using a small dynamic predictor instead. Even if the profile wrongly classifies a branch as belonging to the easy sub-stream, the dynamic component will be more accurate than a pure semi-static predictor. In this paper we have used a 512 byte agree predictor with compiler enhancements. The agbias predictor used in this paper requires two bits encoded in the instruction: a branch bias bit, and a branch direction bit.

Figure 2 shows the prediction accuracy of the agbias predictor compared to other compiler-enhanced predictors. When not specified, results correspond to the cross-trained test.

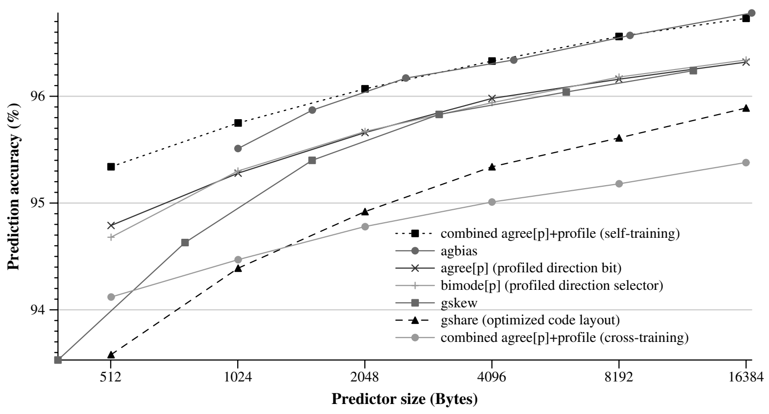

Fig. 2. Prediction accuracy of the agbias predictor compared to other compiler-enhanced predictors

Our results show that the agbias predictor is more accurate than the other predictors examined for all predictor sizes, reaching 96.8% average accuracy for the studied benchmarks, improving on the 96.3% obtained with the compiler-enhanced agree and bimode predictors.

The only comparable predictor is the static-dynamic combination when running the self-trained test, because it obtains the same prediction accuracy but does not require any hardware resources for the *easy* sub-stream prediction. However, note that the accuracy of that static-dynamic combination drops to the level of a gshare predictor with the cross-trained test.

The advantage of the agbias predictor is that it obtains the same performance in the self and cross-trained tests, being less dependent on the profile data accuracy.

3.1 BHR Filtering

Clearly, one major advantage of the agbias predictor is that the *hard* sub-stream predictor has much less interference than other predictors, because it only has to worry about 30% of all branches.

But there is a second difference between the agbias predictor and the other de-aliased schemes: updating the BHR only for branches in the *hard* sub-stream. This selective BHR update achieves an important result: it is increasing the amount of *useful* history information kept in the BHR.

The outcome of the easy branches does not provide the predictor with any extra information, because we *already know* the outcome of the branch. The outcome of the next branches does not depend on it, because it never changes. Not shifting these outcomes into the BHR prevents other -variable- bits from leaving the BHR. This increases the usefulness of the information stored in the first level table, making it easier for the predictor to guess the correct branch outcome.

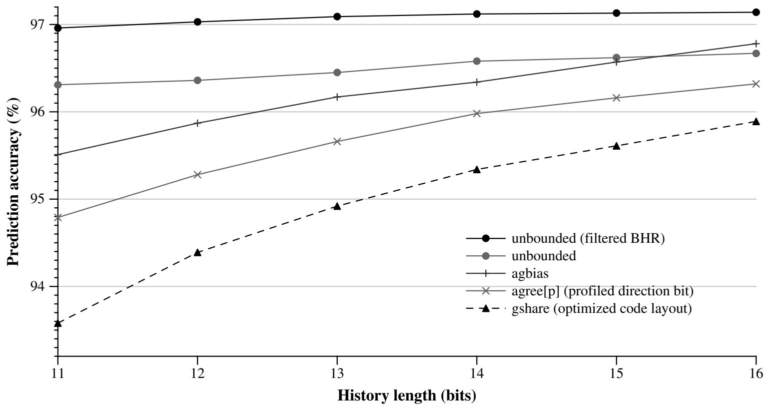

Fig. 3. Effect of BHR filtering on prediction accuracy

Figure 3 shows the prediction accuracy of the agree[p] and the agbias predictors compared to an unbounded predictor using the same history length, and an unbounded predictor with a filtered BHR. The agree and agbias predictors use the history length to determine the PHT size (a 14-bit agree predictor has 2^{14} PHT entries, requiring 4KB of storage, the same 14-bit agbias requires 512 extra bytes for the *easy* sub-stream component). An unbounded predictor has a separate PHT entry for each branch and each possible BHR value, so it is free of interference.

The unbounded predictor increases performance from 96.3% to 96.7% as history length increases, showing that a longer history register represents an advantage to this predictor. The agree and agbias predictors experience larger improvements with increasing history lengths because a longer history also implies a larger PHT, and less interference.

The most remarkable result is that the agbias predictor obtains equivalent performance to an unbounded predictor for 15 bits of history (96.6%), and it actually obtains higher accuracy with 16 bits (96.8% vs 96.7%). But the comparison is not fair, because the agbias predictor also benefits from the BHR filtering: over 70% of all branches do not update the BHR, causing the history length of the agbias predictor to behave like it were 70% larger.

As expected, the unbounded predictor also benefits from a filtered history length, increasing accuracy of a 16-bit history predictor from 96.7% to 97.14%, the same performance as a non-filtered predictor of 22 history bits (not shown). This shows that PHT interference is not the only relevant factor to two-level prediction accuracy: the amount of *useful* information stored in the Level 1 tables is also important.

We have shown how the agbias obtains higher accuracy than any other examined predictor (including their compiler enhanced versions): first, it obtains an important interference reduction in the PHT; second, it increases the usefulness of the BHR information, increasing the potential performance of its two-level adaptive predictor components.

4 Conclusions

In this paper we have shown how we can increase branch prediction accuracy by combining software and hardware techniques, providing yet another example of how the combination of software and hardware techniques can lead to higher performance at a lower implementation cost.

Based on previously proposed predictors and taking full advantage of the compiler, we have presented the *agbias* branch prediction scheme, a static-dynamic hybrid predictor based on the division of the branch stream in four sub streams. A first division among strongly biased (*easy* branches) and not strongly biased branches (*hard* branches), and a second division among mostly *taken* and *not taken* branches.

The agbias predictor uses the *agree* predictions scheme to separate the taken and the not taken sub-streams, and uses two separate dynamic components to separate the easy and the hard sub-streams. Branches are classified using profile information, encoding the class in the instruction opcode.

We outperform all other examined branch prediction schemes for all predictor sizes, obtaining a 96.8% prediction accuracy with a 16KB predictor versus the 96.3% obtained with a compiler enhanced version of agree or bimode, reducing misprediction rate by 14%.

References

1. Thomas Ball and James R. Larus. Branch prediction for free. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 300–313, June 1993. 388
2. D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the simplescalar tool set. Technical Report TR-1308, University of Winsconsin, July 1996. 387
3. Brad Calder and Dirk Grunwald. Reducing branch costs via branch alignment. *Proceedings of the 6th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 242–251, October 1994. 386, 388
4. Brad Calder, Dirk Grunwald, and Donald Lindsay. Corpus-based static branch prediction. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 79–92, 1995. 388
5. Po-Yung Chang, Marius Evers, and Yale N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. *Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques*, October 1996. 388, 389
6. Po-Yung Chang, Eric Hao, Tse-Yu Yeh, and Yale N. Patt. Branch classification: a new mechanism for improving branch predictor performance. *Proceedings of the 27th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 22–31, 1994. 388, 389
7. A. N. Eden and Trevor N. Mudge. The yags branch prediction scheme. *Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques*, pages 69–77, 1998. 388, 389
8. Joseph A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, 30(7):478–490, July 1981. 388

9. Joseph A. Fisher and Stefan M. Freudenberger. Predicting conditional branch directions from previous runs of a program. *Proceedings of the 5th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, 1992. 386, 388
10. Dirk Grunwald, Donald Lindsay, and Benjamin Zorn. Static methods in hybrid branch prediction. *Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques*, pages 222–229, 1998. 387, 388
11. Andreas Krall. Improving semi-static branch prediction by code replication. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 97–106, 1994. 388
12. Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The bi-mode branch predictor. *Proceedings of the 30th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 4–13, December 1997. 386, 387, 388
13. Dondald Lindsay. Static methods in branch prediction. *Ph.D. thesis, Department of Computer Science, University of Colorado*, 1998. 388
14. Scott McFarling. Combining branch predictors. Technical Report TN-36, Compaq Western Research Lab., June 1993. 388
15. Scott McFarling and John Hennessy. Reducing the cost of branches. *Proceedings of the 13th Annual Intl. Symposium on Computer Architecture*, pages 396–403, 1986. 388
16. Pierre Michaud, Andre Seznec, and Richard Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. *Proceedings of the 24th Annual Intl. Symposium on Computer Architecture*, pages 292–303, 1997. 386, 387, 388
17. Frank Mueller and David A. Whalley. Avoiding conditional branches by code replication. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 56–66, 1995. 388
18. Ravi Nair. Dynamic path-based branch correlation. *Proceedings of the 28th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 15–23, November 1995. 386, 388
19. Sanjay Jeram Patel, Marius Evers, and Yale N. Patt. Improving trace cache effectiveness with branch promotion and trace packing. *Proceedings of the 25th Annual Intl. Symposium on Computer Architecture*, pages 262–271, June 1998. 388, 389
20. Harish Patil and Joel Emer. Combining static and dynamic branch prediction to reduce destructive aliasing. *Proceedings of the 6th Intl. Conference on High Performance Computer Architecture*, pages 251–262, January 2000. 386, 387, 388, 389
21. Jason R. C. Patterson. Accurate static branch prediction by value range propagation. *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 67–78, 1995. 388
22. Alex Ramirez, Josep L. Larriba-Pey, and Mateo Valero. The effect of code reordering on branch prediction. *Proceedings of the Intl. Conference on Parallel Architectures and Compilation Techniques*, pages 189–198, October 2000. 387
23. James E. Smith. A study of branch prediction strategies. *Proceedings of the 8th Annual Intl. Symposium on Computer Architecture*, pages 135–148, 1981. 386, 387, 388
24. Eric Sprangle, Robert S. Chappell, Mitch Alsup, and Yale N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. *Proceedings of the 24th Annual Intl. Symposium on Computer Architecture*, pages 284–291, 1997. 386, 387, 388

25. T. Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. *Proceedings of the 24th Annual ACM/IEEE Intl. Symposium on Microarchitecture*, pages 51–61, 1991. 386, 388
26. Cliff Young and Michael D.Smith. Improving the accuracy of static branch prediction using branch correlation. *Proceedings of the 6th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 232–241, October 1994. 388