Branch Classification: a New Mechanism for Improving Branch Predictor Performance

Po-Yung Chang Eric Hao Tse-Yu Yeh* Yale Patt

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Intel Corporation*
Santa Clara, CA 95051

Abstract

There is wide agreement that one of the most important impediments to the performance of current and future pipelined superscalar processors is the presence of conditional branches in the instruction stream. Speculative execution seems to be one solution of choice to the branch problem, but speculative work is discarded if a branch is mispredicted. Therefore, we need a very accurate branch predictor; 95% accuracy is not good enough. This paper proposes branch classification to help improve the accuracy of branch predictors. Branch classification allows an individual branch instruction to be associated with the branch predictor best suited to predict its direction. Using this approach, a hybrid branch predictor can be constructed such that each component branch predictor predicts those branches for which it is best suited. This paper suggests one classification scheme, analyzes several branch predictors, and proposes a hybrid branch predictor that achieves higher prediction accuracy than any branch predictor previously reported in the literature.

Keywords: branch classification, branch predictor, speculative execution, processor performance, superscalar.

1 Introduction

Branches can significantly reduce the performance of pipelined processors if they interrupt the steady supply of instructions to the instruction pipeline [4]. A branch predictor minimizes the number of pipeline stalls by predicting the direction of the branch and fetching the instructions from that path. Because all speculative work beyond a branch must be thrown away if that branch is mispredicted, a very accurate branch prediction algorithm is important to a high-performance microprocessor.

If we ignore stalls such as cache misses and bus conflicts, the branch penalty is defined as C * ((1-p) * r * ipc), where C denotes the number of cycles wasted due to a branch misprediction, p denotes the prediction accuracy, r denotes the ratio of the number of branches over the number of total instructions, and ipc denotes the average number of instructions that are executed per cycle. For C=5 and $r * ipc = 0.9$, a branch penalty of less than 10% requires a prediction accuracy p of greater than 97.7%.

We introduce branch classification as a technique that can improve the accuracy of branch predictors. Using one method of branch classification, we analyze several branch predictors and propose hybrid predictors that achieve higher accuracy than any branch predictors that have previously been reported.

This paper is organized as follows: Section 2 presents the concept of branch classification. Section 3 describes a branch classification model, analyzes previously proposed prediction schemes, proposes several new hybrid branch prediction schemes, and presents simulation results. Section 4 provides some concluding remarks.

2 Branch Classification

Branch classification partitions a program's branches into sets or *branch classes*. The partitioning of branches can be done statically and/or dynamically. A good classification scheme partitions branches possessing similar dynamic behavior into the same branch class; thus, once we understand the dynamic behavior of a class of branches, we can optimize for this class.

For example, the compiler can try to eliminate hard-to-predict branches or the hardware can special case the handling of these branches (e.g. execute both paths of the branch).

Branch classification can be used to maximize the prediction accuracy obtained from a given hardware budget. Prediction accuracy is increased by associating each branch class with the most suitable predictor for that class. For example, we could use a simple predictor for predictable branches and dedicate more resources to handle branches that are more difficult to predict.

3 Experiments

To collect the dynamic branch behavior, the compiler replaces the code that calculates the branch conditions with function calls. Each of these functions calls the branch predictor simulator to generate the branch prediction and to update the state of the simulated prediction hardware. Using this method, we can compare the performance of various branch predictors on a per-branch basis. The behavior of the program is not changed because these functions return the actual branch conditions; the program always executes down the correct path. This approach, however, is not accurate enough to fine-tune a real design. A more accurate approach uses instrumented executable files to measure per-branch prediction accuracy.

3.1 Benchmarks

The results presented in this paper are for the six integer programs from the SPECint92 suite: `espresso, lisp, eqntott, compress, sc, and gcc.` Table 1 shows the training and testing data sets for each of these benchmarks. Because `eqntott` and `lisp` are not provided with different data sets, they use inputs which are not from the SPECint92 suite for training.

Benchmark	Training Data	Testing Data
008.espresso	cps	bca
022.li	deriv.cl[1]	nine queens
023.eqntott	bool. eq.[2]	int_pri_3.eqn
026.compress	gcc source	in
072.sc	loada2	loada1
085.gcc	jump.i	stmt.i

Table 1: Training and Testing Data Sets of Benchmarks

[1] Common Lisp version of a symbolic derivative benchmark written by Vaughan Pratt.

[2] 27 boolean equations with 37 different variables

3.2 Branch Classification

In our experiment, branches are classified based on their dynamic taken-rate collected by profiling as shown in Table 2. Because this partitioning of branches is done statically, we will refer to these branch classes as *static classes*. In the following section, we will refer to SC1, SC2, SC5, and SC6 branches as mostly-one-direction branches and SC3 and SC4 branches as mixed-direction branches. With this classification, we determine whether branches that have similar taken rates have similar dynamic behavior and whether hybrid prediction schemes based on this classification will be able to outperform previously known predictors.

Classes	Descriptions
SC1	0% <= pr(br) <= 5%
SC2	5% < pr(br) <= 10%
SC3	10% < pr(br) <= 50%
SC4	50% < pr(br) <= 90%
SC5	90% < pr(br) <= 95%
SC6	95% < pr(br) <= 100%

Table 2: Static Classes

3.3 Previous Branch Prediction Work

Static branch prediction algorithms use information gathered before program execution, such as branch opcodes or profiles, to predict branch direction. The simplest kind of branch prediction is to predict that all conditional branches are always taken (as in Stanford MIPS-X [1]), or always not-taken (as in Motorola MC88000 [8]). Predicting all branches to be taken achieves about 65% accuracy whereas predicting not-taken achieves about 35% [3, 5, 9].

Dynamic branch prediction algorithms use hardware to record branch execution history at run-time, and predict future branch directions by studying their previous behavior. Many dynamic prediction methods have been studied [2, 7, 9, 10, 11]. One important class, branch target buffers, uses fast hardware logic to detect branches at an early stage of the instruction pipeline, and predict the branch direction and target address [9]. High branch prediction accuracies, about 85%-90%, have been reported for simple history bit and counter-based schemes [9]. By keeping more history information, an even higher level of branch prediction accuracy, about 90%-95%, can be attained [6, 10, 11, 12]. To further improve prediction accuracy, McFarling [6] proposed a new technique that combines two branch predictors. His technique uses 2-bit up-down counters to keep track of which predictor is currently more accurate for each branch; the hybrid predictor then uses the more accurate predictor for making its prediction.

In this paper, we introduce a new method for combining branch predictors. Branches are partitioned into different branch classes based on not only run-time information but also compile-time information. Our hybrid branch predictor then associates each branch class with the most suitable predictor for that class. Furthermore, our technique can combine the advantages of several branch predictors.

3.4 Simulation Results

In this section, we will first show the advantages of branch classification. We will then present the design and performance of several hybrid branch predictors.

In our experiment, three different single-scheme branch predictors are simulated: profile guided(PG), 2-bit up-down counter (2bC), and the Two-Level Branch Predictor. Three different implementations of the Two-Level Branch Predictor are studied. They are the Per-address Two-Level Branch Predictor using a set of pattern history tables (PAs), the Global Two-Level Branch Predictor using a set of pattern history tables (GAs), and a modified GAg scheme (gshare [6]) that exclusive-ORs the global history with the branch address to select the appropriate pattern history table entry.

3.4.1 Advantages of Branch Classification

In our study, static branches with similar dynamic taken-rates are grouped together. Because branches in different classes have different dynamic behavior, the optimal branch prediction scheme may be different for each of these classes. In this section, we report and analyze the performance of branch prediction schemes on each static class to show the performance benefits of branch classification.

- *Analysis of the Mostly One-direction Branches (SC1,SC2,SC5,SC6)*

  Figures 1, 2, and 3 show the average prediction accuracy on integer benchmarks using the PAs, the GAs, and the gshare schemes respectively. Each curve shows the prediction accuracy for one implementation cost. On each curve, as the branch history length decreases by one, the number of pattern history tables doubles. As shown in these figures, branch prediction schemes with short history registers are effective in predicting the mostly-not-taken branches (SC1). With the PAs scheme, a long history is required to capture the odd occurrence of the dynamic behavior of the mostly-one-direction branches; e.g. a repeating branch history pattern with a leading 1 followed by ninety-nine 0s (1% taken-rate) will require 100 branch histories in order to capture the occurrence of "1". However, since these branches are mostly-not-taken, prediction accuracy remains high even if this taken occurrence of the branch is mispredicted. The prediction accuracy of the GAs scheme also decreases with longer branch history because it takes longer to fill the pattern history. A short history results in a faster predictor warm-up time. Furthermore, for a given implementation cost, a shorter history register means more PHTs in the prediction scheme. With more PHTs, fewer branches are mapped to the same PHT; thus, the amount of interference between the pattern history of different branches is reduced. As in the GAs scheme, the prediction accuracy of the gshare scheme also decreases with longer branch history because more history patterns means more PHT entries are accessed. Thus, more PHT conflicts between branches of the mostly-one-direction branches and the mixed-direction branches occur. The performance on the SC2 branches is similar to that of the SC1 branches.

  The performance on the mostly taken branches (SC5,SC6) is similar to that mentioned above (SC1). For example, if branches are mostly taken, the branch history register will consist mostly of 1's. These branches will tend to access the same part of each PHT, possibly causing mispredictions due to different branches accessing the same PHT entry. We can reduce these conflicts by having a short branch history register and more PHTs.

- *Analysis of the Mixed-direction Branches (SC3,SC4)*

  Figures 4, 5, and 6 show the prediction accuracy of branches whose dynamic taken-rates are between 10% and 50% (SC3). Unlike the mostly-one-direction branches, the mixed-direction branches are effectively predicted by prediction schemes with long branch history registers. Because these branch have dynamic taken-rates between 10% and 50%, we will see more execution patterns due to these branch characteristics. By having a longer branch history, we can distinguish more execution states. In addition, with a longer branch history, histories of correlated branches are more likely to remain in the branch history register. Thus, the branch prediction schemes with long branch histories are effective in predicting the mixed-direction branches.

  The performance of GAs, PAs, and gshare on mostly taken branches (SC4) is similar to that mentioned above (SC3).

- *Weight of Branch Classes*

  Let us define the dynamic weight of a branch class as

$$\frac{No.\ of\ dyn.\ brns\ belonging\ to\ that\ brn\ class}{total\ number\ of\ dynamic\ branches}.$$

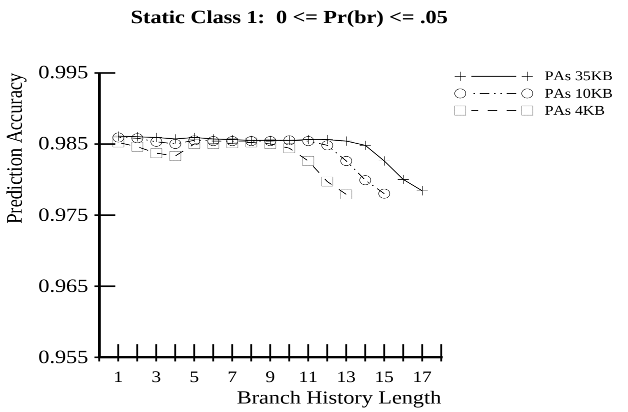

Figure 1: Prediction Accuracy of PAs on the Mostly Not-taken Branches

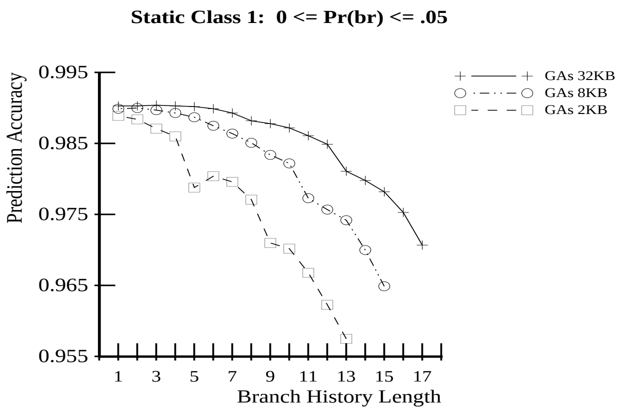

Figure 2: Prediction Accuracy of GAs on the Mostly Not-taken Branches

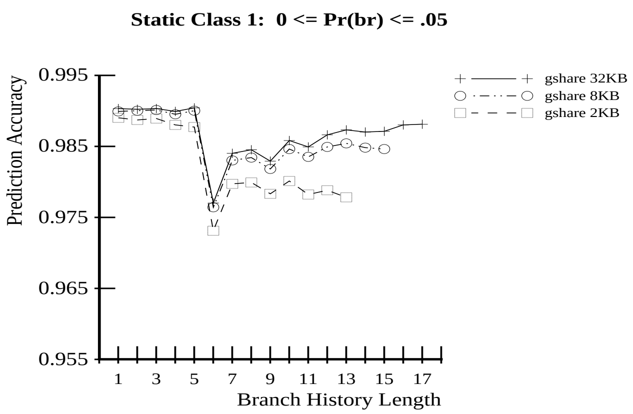

Figure 3: Prediction Accuracy of gshare on the Mostly Not-taken Branches

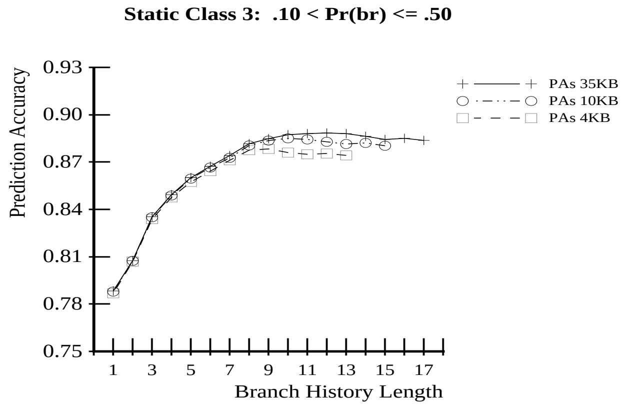

Figure 4: Prediction Accuracy of PAs on the Mixed-direction Branches

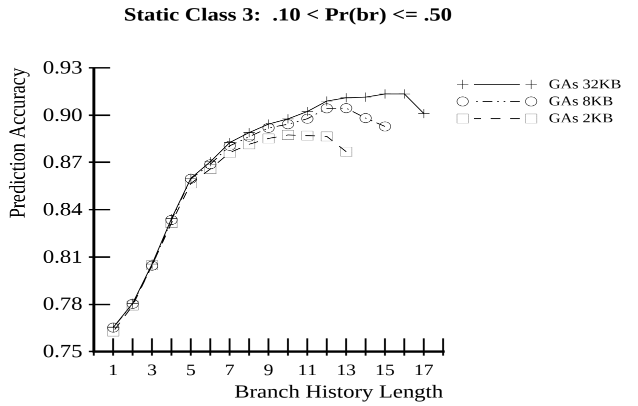

Figure 5: Prediction Accuracy of GAs on the Mixed-direction Branches

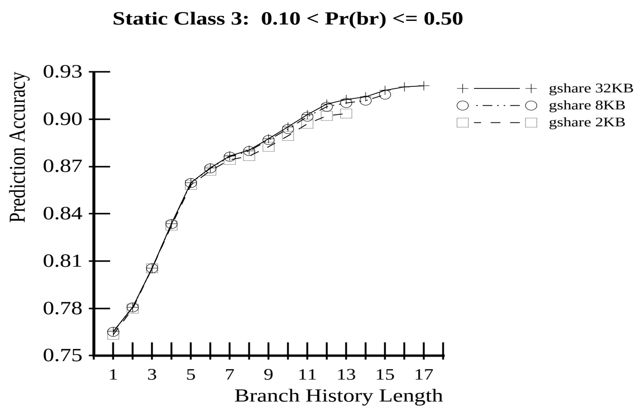

Figure 6: Prediction Accuracy of gshare on the Mixed-direction Branches

Figure 7 shows the dynamic weight of each static class. Approximately 50% of all dynamic branches are mostly-one-direction branches; the other 50% are mixed-direction branches. Thus, the performance of a predictor is dependent on its prediction accuracy on both the mostly-one-direction branches and on the mixed-direction branches.

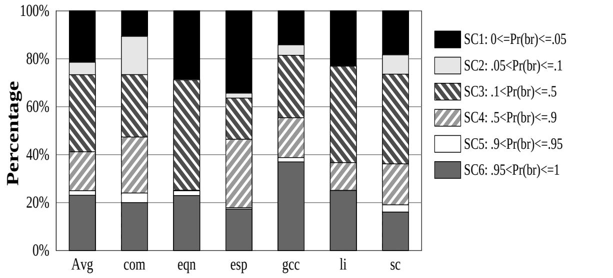

Figure 7: Percentage of dynamic branches in each static class

We have shown that the optimal predictor configuration for the mostly-one-direction branches is different from that of the mixed-direction branches. Thus, these single-scheme predictors cannot be configured optimally for both types of branches. Figures 8, 9, and 10 show the average prediction accuracy using the GAs, PAs, or gshare scheme with the branch history length ranging from 1 to 18. Each curve in the graphs indicates the performance of a branch predictor at a fixed hardware cost. The hardware cost of a predictor is estimated using the following equations [12]:

$$\begin{aligned} \text{GAs}(k,p) &= \quad k \quad + (p \times 2^k \times 2) \qquad (bits) \\ \text{PAs}(k,p) &= (b \times k) + (p \times 2^k \times 2) \qquad (bits) \\ \text{gshare}(k,t) &= \quad k \quad + \quad (2^t \times 2) \qquad (bits) \end{aligned}$$

where k is the history register length, p is the number of pattern history tables (PHTs), b is the number of entries in the branch history table, and t is the number of PHT entries in gshare. For example, the highest curve in Figure 9 shows the performance of a 32K-byte size GAs. The left-most point of this curve shows the prediction accuracy of $\text{GAs}(1, 2^{16})$. The right-most point shows the prediction accuracy of $\text{GAs}(17, 1)$. Our results match those presented in [12]. Let $\texttt{PA}(\text{x})$ denote the prediction accuracy of the branch prediction scheme x. With a fixed branch history length, the prediction accuracy increases as the number of PHTs increases, e.g. $\texttt{PA}(\text{GAs}(5, 2^8)) < \texttt{PA}(\text{GAs}(5, 2^{10})) < \texttt{PA}(\text{GAs}(5, 2^{12}))$. With a fixed number of pattern history tables, the prediction accuracy increases as the length of history register increases; e.g. $\texttt{PA}(\text{GAs}(11, 4)) < \texttt{PA}(\text{GAs}(13, 4)) < \cdots < \texttt{PA}(\text{GAs}(15, 4))$. The performance of the PAs scheme is less sensitive to the branch history length than that of the GAs scheme, e.g. $\texttt{PA}(\text{PAs}(15, 4))$ - $\texttt{PA}(\text{PAs}(13, 4)) < \texttt{PA}(\text{GAs}(15, 4))$ - $\texttt{PA}(\text{GAs}(13, 4))$.

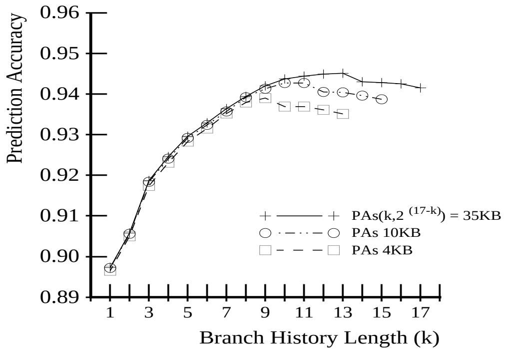

Figure 8: Per-address history schemes with different branch history length

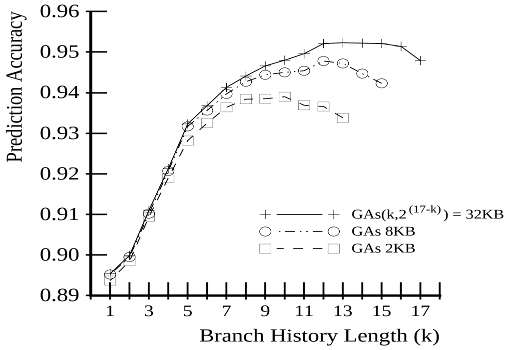

Figure 9: Global history schemes with different branch history length

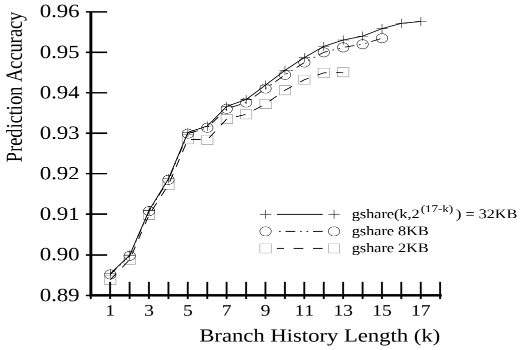

Figure 10: gshare with different branch history length

Figures 1, 2, 4, and 5 show that the optimal configuration of GAs and PAs is sub-optimal for both the mostly-one-direction branches and the mixed-direction branches. Without branch classification, the gshare configuration is also sub-optimal for the mostly-one-direction branches, as shown in figures 3 and 6.

3.4.2 Combining the Advantages of Different Predictors

In this section, we introduce hybrid branch predictors which combine the advantages of different branch predictors. We will first introduce hybrid branch predictors which statically select a branch predictor for each branch. We then show a hybrid branch predictor design which selects a branch predictor both statically and dynamically. Finally, we summarize these hybrid prediction schemes.

3.4.2.1 Static Branch Predictor Selection

- **GAs with Multiple Branch History Length** We have shown that the GAs scheme is not designed to optimize prediction accuracy on both the mixed-direction and the mostly-one-direction branches. To increase prediction accuracy on both types of branches, we propose a new branch prediction scheme, which uses multiple history length, called GAs.mhl; it uses a short global history for the mostly-one-direction branches and a long branch history for the mixed-direction branches. Figure 11 shows the structure of GAs with multiple history length. Because we are using fewer bits for the mostly-one-direction branches, there may not be enough history bits to identify each branch. To better identify each branch, the gshare scheme [6] uses both the global history and the branch address. As in the gshare scheme, GAs.mhl exclusive-ORs the global history with the branch address to select the appropriate PHT entry. This is done to hash the frequent global history patterns to different PHT entries. Because the data set used to gather profile information is different from the one used in the actual testing run, some branches may only be executed during the testing run and, thus, have no profiled information. In this case, these branches are predicted with the scheme that is using a long branch history.

  Figure 12 compares the performance of GAs.mhl with GAs and gshare. This figure shows the best prediction accuracy of these predictors at each hardware cost. For all predictor sizes, GAs.mhl outperforms both GAs and gshare. Figure 13 shows the prediction accuracy of 1K-byte GAs.mhl, GAs, and gshare on each static class. GAs.mhl and gshare outperform GAs because they more effectively utilize the PHTs. The most significant performance difference between GAs.mhl and gshare is on SC1 and SC6 branches. By using a short history to predict the mostly-one-direction branches, GAs.mhl is able to achieve prediction accuracies that are .015 and .0098 higher than those of gshare on SC1 and SC6 branches respectively. For the mixed-direction branches, the slight improvement in prediction accuracy is due to the fact that the mostly-one-direction branches are now hashed into fewer PHT entries and result in fewer conflicts between the pattern history of the mostly-one-direction branches and that of the mixed-direction branches.

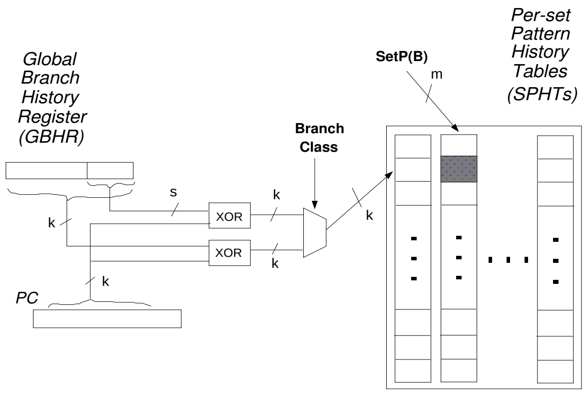

Figure 11: Structure of GAs with Multiple Branch History Length

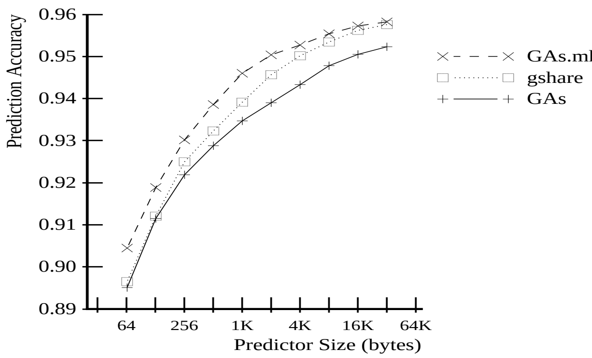

Figure 12: Performance of GAs with Multiple Branch History Length

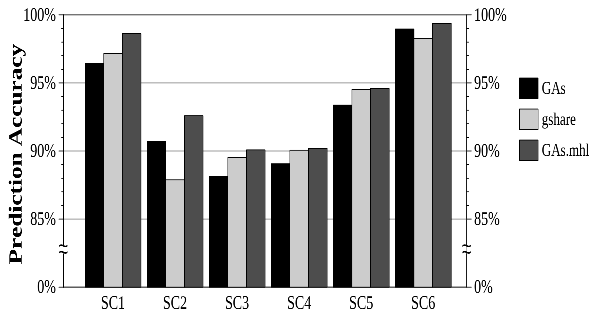

Figure 13: Performance of 1K Predictors on each static class

- **Combination of static and dynamic predictors**
The static predictors can accurately predict the mostly-one-direction branches. Using static predictors for those branches, the hardware predictor can be optimized for accurately predicting the mixed-direction branches. In our experiment, the PG+gshare scheme uses the profile guided predictor to statically predict SC1 and SC6 branches and the gshare scheme to dynamically predict the other branches. If a branch is not executed during the training run, then the dynamic predictor is designated for predicting this branch during the testing run.

  Figure 14 shows that PG+gshare outperforms GAs.mhl and single-scheme predictors. Figures 15 and 16 show the prediction accuracy of PG+gshare and GAs.mhl on SC1 and SC3 branches respectively. For SC1 and SC6 branches, the profile-guided predictor can accurately predict these branches because these branches are mostly taken or mostly not-taken. In addition, PHT conflicts between the mixed-direction branches and the mostly-one-direction branches can significantly reduce the accuracy of a gshare scheme, especially at lower implementation costs. Thus, PG+gshare outperforms GAs.mhl on the SC1 and SC6 branches.

  For the mixed-direction branches, PG+gshare achieves a slightly higher prediction accuracy than GAs.mhl. Because the gshare scheme in PG+gshare hybrid predictor only predicts the mixed-direction branches, the PHT contention between mostly-one-direction branches and mixed-direction branches no longer exists. Also, by dealing only with mixed-direction branches, histories of correlated branches are more likely to remain in the PG+gshare's branch history register.

3.4.2.2 Static plus Dynamic Predictor Selection

Up to this point, we have statically selected the optimal predictor for each branch. Another method of combining branch predictors is to select the optimal predictor dynamically [7]. In this section, we first compare the performance of both types of hybrid branch predictor. We then propose a new hybrid predictor design that uses both dynamic and static predictor selection to further improve prediction accuracy.

- **Hybrid Predictor with Dynamic Predictor Selection**
One method of dynamically selecting the optimal predictor is to use 2-bit saturating up-down counters (i.e. 2bC) to keep track of which predictor is doing better [6]. Specifically, let BD denote the actual branch direction, P1 denote predicted direction from predictor 1, and P2 denote the predicted direction from predictor 2. The counters can be incremented or decremented based on the rule shown in Table 3.

  In our study, we associate a counter with each entry in the fully associative branch address cache(BAC).

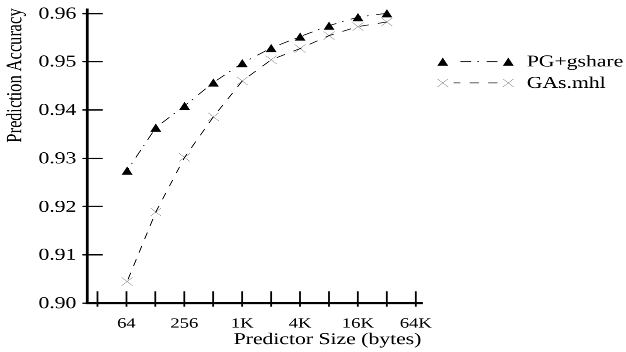

Figure 14: Performance of Hybrid Predictors with Static Branch Classification

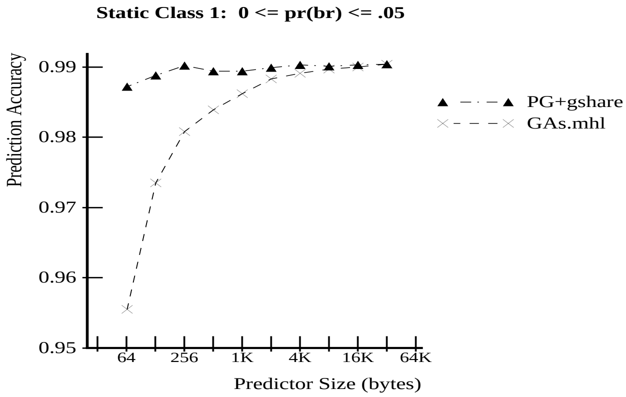

Figure 15: Performance of PG+gshare and GAs.mhl on mostly not-taken branches

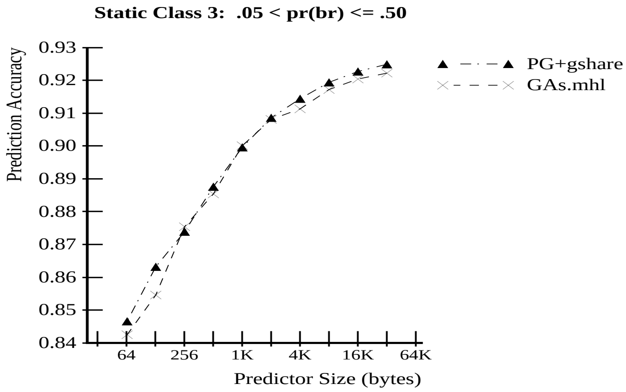

Figure 16: Performance of PG+gshare and GAs.mhl on mixed-direction branches

BD	P1	P2	
0	0	0	no change
0	0	1	decrement counter
0	1	0	increment counter
0	1	1	no change
1	0	0	no change
1	0	1	increment counter
1	1	0	decrement counter
1	1	1	no change

Table 3: Updating rule for predictor selection counters

When a branch is fetched, the corresponding counter found in the BAC is then used to determine which predictor to use, as shown in Figure 17.

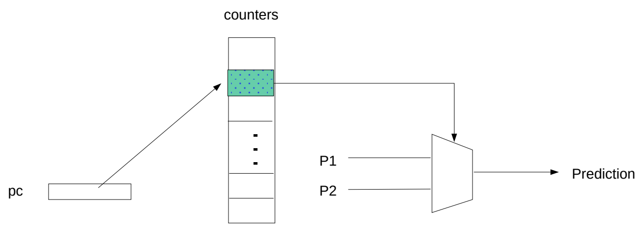

Figure 17: Structure of Hybrid Predictor with Dynamic Predictor Selection

We simulated two different combinations of predictors: the 2-bit up-down counter predictor with gshare (2bC/gshare) and PAs with gshare (PAs/gshare). Figure 18 compares the performance of these hybrid predictors with the static predictor selection scheme PG+gshare. The hardware costs for 2bC/gshare and PAs/gshare are estimated using the following equations:

$$\begin{aligned} \text{2bC/gshare}(k,p,a) &= (a \times 2) + (a \times 2) \\ &\quad + k + (p \times 2^k \times 2) \\ \text{PAs/gshare}(k,p,a) &= (a \times 2) + (b \times k) + (p \times 2^k \times 2) \\ &\quad + k + (p \times 2^k \times 2) \end{aligned}$$

where a is the number of entries in the branch address cache, k is the history register length, p is the number of PHTs, and b is the number of entries in the branch history table. That is, the cost of the hybrid predictor is determined by summing the cost of the single-scheme predictors and the counters used to select the optimal predictor. For PAs/gshare, we only considered combining PAs and gshare configurations with the same implementation cost.

For predictors smaller than 16K bytes, the PG+gshare scheme outperforms the PAs/gshare scheme. With PG+gshare and PAs/gshare of a similar size, the gshare in the PG+gshare scheme is approximately twice the size of either PAs or gshare

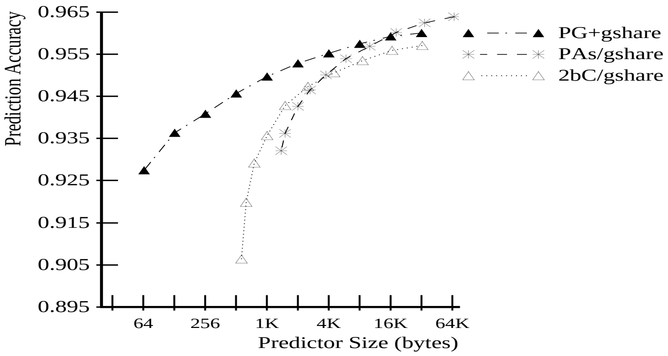

Figure 18: Prediction Accuracy of Hybrid Branch Predictors

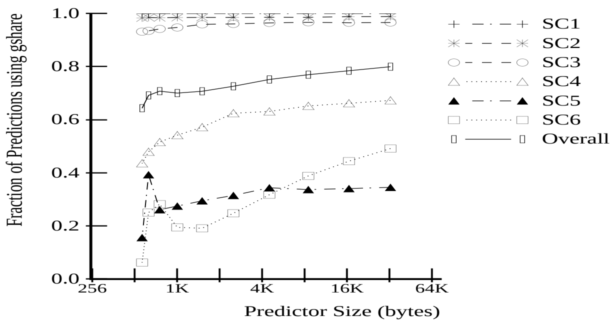

Figure 19: Fraction of gshare usage in the 2bC/gshare scheme

in the PAs/gshare scheme. With an implementation cost below 16K bytes, the larger gshare was able to outperform the combination of two smaller predictors. On the other hand, for predictors larger than 16K bytes, PAs/gshare is able to outperform PG+gshare. Because the benefits of a larger predictor diminishes as the size of the predictor increases, a combined predictor outperforms a larger single-scheme predictor.

For the SPEC integer benchmarks, PG+gshare outperforms the 2bC/gshare scheme. In this study, the 2bC/gshare predictor scheme uses a 1K-entry BAC and 1K 2-bit counters. Figure 19 shows how often the gshare was used to make predictions in the 2bC/gshare scheme. The size of 2bC/gshare is increased by only increasing the gshare portion; the 2bC portion remains fixed at 256 bytes. With increasing 2bC/gshare size, the prediction accuracy of gshare increases and, thus, more branches are predicted using the gshare scheme. Figure 19 shows that the majority of predictions on the mostly not-taken branches are made by gshare. Since gshare, at a low implementation cost, is not as accurate as the profile-guided predictor, PG+gshare outperforms 2bC/gshare on the mostly-one-direction

	mostly one-direction branches	mixed-direction branches
GAs.mhl	GAs with a short Branch History	GAs with a long Branch History
PG+gshare	PG	gshare
2bC/gshare	2bC or gshare (selected dynamically)	
PAs/gshare	PAs or gshare (selected dynamically)	
PG+PAs/gshare	PG	PAs or gshare (selected dynamically)

Table 4: Summary of Hybrid Branch Prediction Schemes

	mostly one-direction branches	mixed-direction branches
PG+PAs	PG	PAs
PG+GAs	PG	GAs
2bC/GAs	2bC or GAs (selected dynamically)	
PAs/GAs	PAs or GAs (selected dynamically)	
PG+PAs/GAs	PG	PAs or GAs (selected dynamically)

Table 5: Omitted Hybrid Branch Prediction Schemes

branches. Figure 19 also shows that the 40% of predictions on SC4 branches are made by the 2bC scheme. While 2bC can predict mostly-one-direction branches well, it is outperformed by gshare on the mixed-direction branches. Thus, PG+gshare also outperforms 2bC/gshare on the the mixed-direction branches.

- **A New Hybrid Predictor with Both Dynamic and Static Predictor Selection**

  In this section, we propose a new hybrid branch predictor that exploits the advantage of using both run-time and compile-time information to assist branch prediction. The PG+PAs/gshare scheme uses the profile-guided predictor for the mostly-one-direction branches and the PAs/gshare scheme for the mixed-direction branches. Figure 20 shows the performance of PG+PAs/gshare. For predictors smaller than 4K bytes, the PG+gshare scheme provides the optimal performance. On the other hand, for predictors larger than 4K bytes, the PG+PAs/gshare scheme outperforms all other predictors. For example, with a fixed implementation cost of 32K bytes, PG+PAs/gshare is able to achieve prediction accuracy of 96.4% on the SPEC integer benchmarks, as compared to 95.7% for gshare and 95.2% for GAs. For the SPEC92 benchmark gcc, which contains many branches, PG+PAs/gshare achieves prediction accuracy of 96.91%, as compared to 96.47% for the best previously known predictor (PAs/gshare) [7].

3.4.2.3 Summary of Hybrid Branch Predictors

We examined many different hybrid branch prediction schemes. In this report, we present the most successful ones (see Table 4). Table 5 lists several prediction schemes that were omitted.

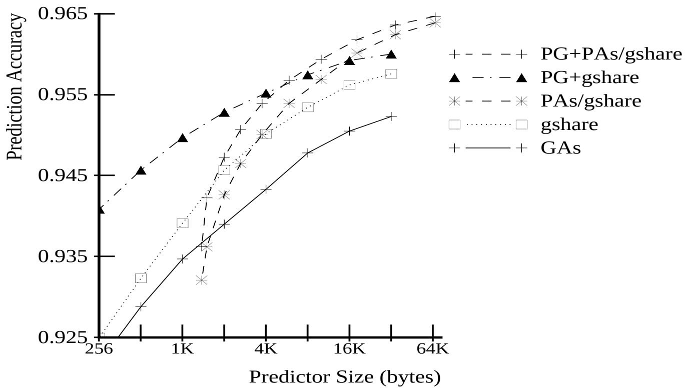

Figure 20: Performance of hybrid branch predictor with both dynamic and static selection

4 Conclusion

We have introduced branch prediction classification as a means for analyzing predictor performance as well as a means for combining the advantages of different predictors. The proposed branch classification model groups branches based on their dynamic taken rates gathered during profiling. With this branch classification model, we showed that using a short history for the mostly-one-direction branches and a long history for the mixed-direction branches improves the performance of the global history Two-Level Branch Predictors.

We then proposed a hybrid branch predictor that combines the advantages of static predictors and dynamic predictors. Using a profile-guided predictor for the mostly-one-direction branches, more hardware can be dedicated to the dynamic predictor. Furthermore, the dynamic predictor can be optimized to more accurately predict the mixed-direction branches. With an implementation cost of 32K bytes, PG+gshare achieved a 96.0% prediction accuracy, as compared to 95.2% for GAs, reducing the miss rate by 16.7%.

With branch classification, selection of the suitable predictor for each branch can be done dynamically and/or statically. Thus, information from both compile time and execution time can be used to assist the hybrid branch predictor in achieving higher accuracy. With a fixed implementation cost of 32K bytes, our proposed combination of a profile-guided predictor, PAs, and gshare achieved a prediction accuracy of 96.91% on gcc, a branch intensive benchmark, as compared to 96.47% for the best previously known predictor, reducing the miss rate by 12.5%.

In summary, we have shown that the idea of branch classification will allow construction of predictors that are more effective than currently known predictors.

5 Acknowledgments

This paper is one result of our ongoing research in high performance computer implementation at the University of Michigan. The support of our industrial partners: Intel, AT&T/GIS, Motorola, Hewlett-Packard, and Scientific and Engineering Software is greatly appreciated. In addition, we wish to gratefully acknowledge the other members of our HPS research group for the stimulating environment they provide, and in particular, Carlos Fuentes for his comments and suggestions on this work. We would also like to thank the reviewers for their helpful suggestions.

References

[1] P. Chow and M. Horowitz, "Architecture tradeoffs in the design of MIPS-X," *Proceedings of the 14th Annual International Symposium on Computer Architecture*, June 1987.

[2] J.A. DeRosa and H. M. Levy, "An Evaluation of Branch Architectures," *Proceedings of the 14th International Symposium on Computer Architecture*, May 1989.

[3] J. Emer and D. Clark, "A Characterization of Processor Performance in the VAX-11/780," *Proceedings of the 11th Annual Symposium on Computer Architecture*, June 1984.

[4] P.M. Kogge, *The Architecture of Pipelined Computers*, pp.237-243, McGraw-Hill, 1981.

[5] J.K.F. Lee and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Computer*, pp.6-22, January 1984.

[6] S. McFarling, "Combining Branch Predictors", WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.

[7] S. McFarling and J.L. Hennessey, "Reducing the cost of branches," *Proceedings of the 13th International Symposium on Computer Architecture*, pp.396-404, June 1986.

[8] C. Melear, "The design of the 88000 RISC family," *IEEE MICRO*, pp.26-38, April 1989.

[9] , J.E. Smith, "A Study of Branch Prediction Strategies," *Proceedings of the 8th International Symposium on Computer Architecture*, pp.135-148, June 1981.

[10] T.-Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-level Adaptive Branch Prediction," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp.124-135, May 1992.

[11] T.-Y. Yeh and Y.N. Patt, "Two-level Adaptive Branch Prediction," *Proceedings of the 24th ACM/IEEE International Symposium on Microarchitecture*, pp.51-61, November 1991.

[12] T.-Y. Yeh and Y.N. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History", *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp.257-266, May 1993.