To appear in MICRO-30, 1997

The Bi-Mode Branch Predictor

Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge
EECS Department, University of Michigan
1301 Beal Ave., Ann Arbor, Michigan 48109-2122
{leecc, icheng, tnm}@eecs.umich.edu

Abstract

Dynamic branch predictors are popular because they can deliver accurate branch prediction without changes to the instruction set architecture or pre-existing binaries. However, to achieve the desired prediction accuracy, existing dynamic branch predictors require considerable amounts of hardware to minimize the interference effects due to aliasing in the prediction tables. We propose a new dynamic predictor, the bi-mode predictor, which divides the prediction tables into two halves and, by dynamically determining the current "mode" of the program, selects the appropriate half of the table for prediction. This approach is shown to preserve the merits of global history based prediction while reducing destructive aliasing and, as a result, improving prediction accuracy. Moreover, it is simple enough that it does not impact a processor's cycle time. We conclude by conducting a comprehensive study into the mechanism underlying two-level dynamic predictors and investigate the criteria for their optimal designs. The analysis presented provides a general framework for studying branch predictors.

1. Introduction

The ability to minimize stalls or pipeline bubbles that may result from branches is becoming increasingly critical as microprocessor designs implement greater degrees of instruction level parallelism. There are several techniques for reducing branch penalties including guarded execution, basic block enlargement, and static and dynamic branch prediction [PnevmatikatosSohi94, Hwu93, Smith81, FisherFreudenberger92, YehPatt91, PanSoRahmeh92]. Among these, dynamic branch prediction is perhaps the most popular, because it yields good results and can be implemented without changes to the instruction set architecture or pre-existing binaries.

The strength of dynamic branch prediction is that it can track branch behavior closely at run-time, providing a degree of adaptivity that other approaches are lacking. This adaptivity is especially critical when behavior of branches can be affected by the input data of different program runs. With the introduction of two-level schemes [YehPatt91], the prediction accuracy of dynamic branch predictors has been pushed above 90%. As a result, two-level dynamic branch predictors have been incorporated in several recent high-performance microprocessors. Perhaps the best known examples, at the time of writing, are the Pentium Pro [Gwennap95] and Alpha 21264 [Gwennap96].

Among two-level predictors, those using global history schemes have been shown to yield the best performance for integer benchmarks [YehPatt93]. However, to achieve high levels of accuracy, current dynamic branch predictors require considerable amounts of hardware because their most significant weakness, the destructive aliasing problem, is most easily solved by increasing the size of the predictors [SechrestLeeMudge96]. This paper proposes a new technique, the bi-mode branch predictor, that is economical and simple enough to avoid critical timing paths. Furthermore, we demonstrate that on the IBS and SPEC CINT95 benchmarks the bi-mode predictor performs on average better than gshare, one of the best global history based predictors, for the same cost. Finally, we conduct a comprehensive study into the mechanism underlying two-level dynamic predictors and investigate the criteria for their optimal designs. The study explains why our proposed scheme performs well and provides a general framework for studying branch predictors.

The report is organized into five sections. In section 2, we summarize the aliasing problem, and then introduce our solution for de-aliasing. Section 3 describes our simulation methodology and presents the simulation results. In section 4 we present an analysis of aliasing in dynamic branch predictors that explains the source of the improved performance for the bi-mode predictor. Finally, in the conclusion we propose future directions for this work.

2. Aliasing and De-aliasing

2.1 The aliasing problem

Branch outcomes are not usually the result of random activities; most of the time they are correlated with past behavior and the behavior of neighboring branches. By keeping track of the history of branch outcomes, it is possible to anticipate with a high degree of certainty which direction future branches will take.

However, current dynamic branch predictors still exhibit performance limits. These are due in part to the restricted availability of information upon which to base predictions, but more importantly due to shortcomings of design, especially the way that branch outcome history is exploited. In current designs, dynamic predictors spend large amounts of hardware to memorize this branch outcome history. Each static (per-address) branch often has a biased behavior so that it is either usually taken or usually not-taken. This can be exploited by the conventional two-bit counter scheme to predict future outcomes of a particular static branch. However, two-bit counter schemes are limited because branches may behave differently from their biases under some special conditions. These conditions are not difficult to recognize, but recognition requires memory space. Therefore, to achieve very high prediction accuracy, both the per-address bias and the special conditions need to be identified and memorized by dynamic predictors.

Global history—the outcomes of neighboring branches—is a common way to identify special branch conditions. Previous studies have shown that the global history indexed schemes achieve good performance by storing the outcomes of global history patterns in two-bit counters, e.g., the GAg and GAs schemes [PanSoRahmeh92, YehPatt92]. Another way to identify special branch conditions is to use per-address history—the past outcomes of a branch itself, such as PAg and PAs schemes [YehPatt91]. The per-address history scheme is also shown to be effective, especially for loop-intensive floating-point programs. However, as we noted earlier, [YehPatt93] shows that, for integer programs, global history schemes tend to perform better than per-address history schemes because global schemes can make better predictions for if-then-else branches due to their ability to track correlation with neighboring branches.

Nevertheless, the global history scheme is still limited by destructive aliasing that occurs when two branches have the same global history pattern, but opposite biases [TalcottNemirovskyWood95, YoungGloySmith95]. This is not due to the limited availability of information, but to the indexing method which does not discriminate between branches with the same global history patterns.

One proposal to overcome the destructive aliasing, gshare, randomizes the index by xor-ing the global history with the branch address [McFarling93]. It provides only limited improvement [SechrestLeeMudge96]. Recently, there have been several new proposals to reduce aliasing problems [ChangEversPatt96, Sprangle97, [MichaudSeznecUhlig97]. The best of these [MichaudSeznecUhlig97] employs a hardware hashing scheme. A comparative study of these and the bi-mode scheme can be found in [Lee97]. The study shows that hardware hashing is useful for small low cost systems. For large systems the bi-mode scheme is the best cost-effective scheme to date.

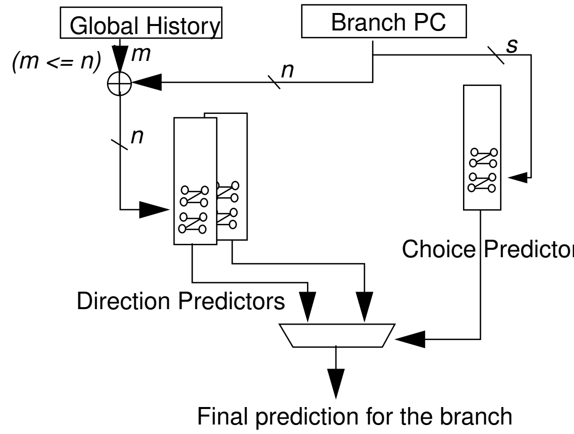

Figure 1: Proposed branch prediction scheme diagram

2.2 Proposed branch prediction scheme

The bi-mode branch predictor is aimed at the elimination of destructive aliasing in global history indexed schemes. This scheme, shown in Figure 1, splits the second-level two-bit counter table into two halves. Given a history pattern, two counters, one from each half, are selected. We refer to these as the direction predictors. Meanwhile, another two-bit counter table, indexed by the branch addresses only, is used to provide a final selection for these two counters. The counter table providing selection will be referred to as the choice predictor. The final prediction is then made by the state of the counter selected from the direction predictors and, importantly, only the selected counter will be updated with the branch outcome; the status of the unselected one, will not be altered. The choice predictor is always updated with the branch outcome, except that when the choice is opposite to the branch outcome but the selected counter of the direction predictors makes a correct final prediction. This partial update policy is particularly effective when the total hardware budget is small.

Our proposed scheme can improve global history indexed schemes because although global history patterns are still kept in the second level table, they are dynamically classified before being stored. They are classified by a preliminary prediction from the choice predictor which is simply a conventional two-bit counter scheme, and, as such, typically can provide 80% or better prediction accuracy with relatively modest cost. Thus, the bi-mode scheme divides branches into two groups according to the per-address bias of the choice predictor, and then uses the global history patterns to identify the special conditions for each of two groups separately. The effect of the choice predictor is to separate the destructive aliases while keeping the harmless aliases together.

3. Experimental Results

In this section, we demonstrate that our proposed bi-mode branch predictor is more accurate and cost-effective than one of the best two-level branch predictors, gshare. To evaluate the improvement, we have conducted trace-driven simulations.

3.1 Description of gshare scheme

In gshare, the global history is xor-ed together with the low-order address bits of a branch to form an index. This index is then used to select a 2-bit saturating up-down counter from a pattern history table (PHT)[1]. Depending on the sign bit of the selected 2-bit counter, the branch is either predicted as taken or not taken.

To make a fair comparison with the gshare predictor, the best configuration of gshare must be determined and used. This point is often overlooked and the single-PHT gshare configuration is used for comparisons. However, this single-PHT gshare configuration is not the optimal configuration as was shown in [SechrestLeeMudge96]. To find the best configuration, we exhaustively simulated all pair-wise combinations of history length and address length. In general, the best combination has multiple PHTs. Since the best configuration is different for each benchmark, we present results using the configuration that yields the best accuracy for the average of all the benchmarks studied.

3.2 Description of input trace

To assess the performance of the bi-mode branch predictor, we conducted a trace-driven simulation using the Ultrix version of the Instruction Benchmark Suite (IBS-Ultrix) benchmarks [Uhlig95] and the SPEC CINT95 benchmarks [SPEC95].

The IBS-Ultrix benchmarks are a set of applications designed to reflect realistic workloads. The traces of these benchmarks were generated through hardware monitoring of a MIPS R2000-based workstation. These traces were collected under Ultrix 3.1, and include both kernel and user activities.

For the SPEC CINT95 benchmark, we use ATOM [EustaceSrivastava95], a code instrumentation tool from Digital Equipment Corporation, to generate and capture address traces. The benchmarks were first instrumented with ATOM, then executed on a DEC 21064 workstation running OSF/1 3.0 to generate traces. These traces contained only user-level instructions. The input to the SPEC95 benchmarks was a reduced input data set and is described in Table 1. The branch statistics of traces from the IBS and the SPEC CINT95 are summarized in Table 2.

	Benchmarks	Input data file
SPEC CINT95	compress	bigtest.in, reduced
	gcc	jump.i
	go	2stone9.in, train data, reduced
	xlisp	train.lsp
	perl	scrabbl.in, reduced
	vortex	train data, reduced

Table 1: Description of the input data files used in the SPEC CINT95 programs

	Benchmarks	static conditional branches	dynamic conditional branches
SPEC CINT95	compress	482	10,114,353
	gcc	16,035	26,520,618
	go	5,112	17,873,772
	xlisp	636	25,008,567
	perl	1,974	39,714,684
	vortex	6,599	27,792,020
IBS-Ultrix	groff	6,333	11,901,481
	gs	12,852	16,307,247
	mpeg_play	5,598	9,566,290
	nroff	5,249	22,574,884
	real_gcc	17,361	14,309,867
	sdet	5,310	5,514,439
	verilog	4,636	6,212,381
	video_play	4,606	5,759,231

Table 2: Static and dynamic branch counts in the IBS and SPEC CINT95 programs

3.3 Simulation results

Figure 2 shows the misprediction rates for the best gshare and bi-mode predictors. In our simulation the best configurations of gshare, which are labeled *gshare.best*, always have multiple PHTs in the second-level table. Note that *gshare.best* is the best for the averaged results, not necessary the best for individual benchmarks. For easy comparison with other published results, we also include the misprediction rates for the single-PHT gshare configuration, which is labeled *gshare.1PHT*. In Figure 2, the vertical axis represents the branch misprediction rate, and the horizontal axis for the size of predictors. A lower curve indicates that the scheme has better performance for the same cost. Cost is measured by counting the number of bytes used in the 2-bit counters. Note that the bi-mode predictors naturally have a cost that is 1.5 times that of the next smaller gshare scheme[2]. This reflects the cost of the choice predictors.

Figure 2 shows the bi-mode predictors outperforms gshare predictors for all sizes of predictors measured. This

1. The pattern history tables are the tables constituting the second-level table of the two-level predictors, as defined in [YehPatt92]. In the two-level predictor model, the number of PHTs is determined by the branch address bits directly used as the index.

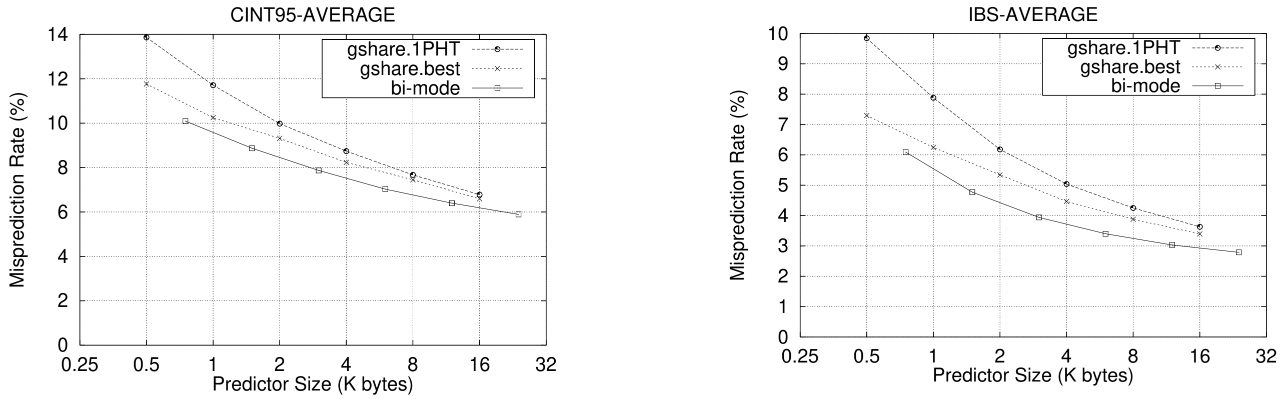

Figure 2: Averaged misprediction rates for SPEC CINT95 and IBS-Ultrix

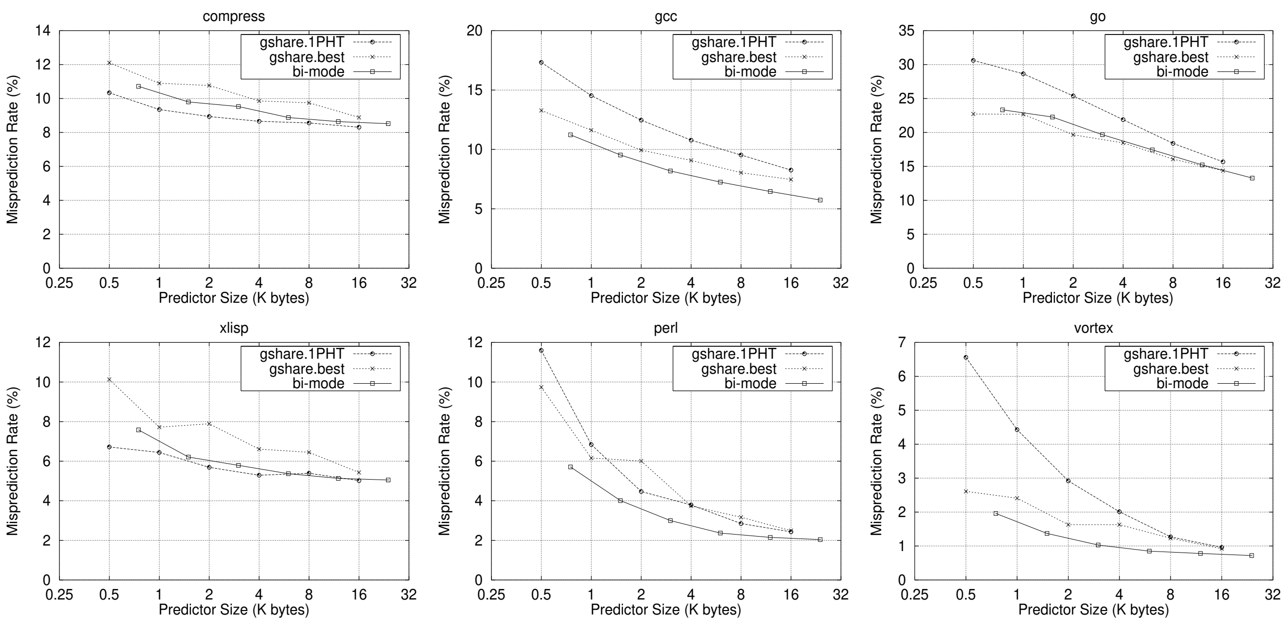

Figure 3: Misprediction rates for SPEC CINT95

is indicated by lower curves. In addition, the bi-mode predictors are more cost effective, because, for predictors larger than 4K bytes, they need less than half the size of gshare predictors to achieve the same misprediction rate.

Bi-mode predictors also outperform gshare on most of the individual benchmark examined, see Figure 3 and Figure 4. Moreover, the single-PHT gshare scheme is worse than the multiple-PHTs gshare scheme for all benchmarks except the *compress* and *xlisp*, where it outperforms even the bi-mode scheme. These two benchmarks, with the fewest static branches, have no aliasing problems and thus can enjoy the benefit from correlation in branch histories. The results of these two small benchmarks correspond to the findings reported by Sechrest *et al*. [SechrestLeeMudge96]. The case of the *go* benchmark, where the bi-mode method is beaten by the multiple-PHTs, will be discussed in more detail in the next section.

4. Analysis

Many branches have a tendency to be either taken or not-taken most of time. Common examples are branches for error checking and looping. These kinds of branches are usually described as being strongly biased in one direction. As might be expected, strongly biased branches are much easi-

2. In our experiments, all two-bit counters in gshare schemes are initialized to weakly-taken for each benchmark run. For the bi-mode scheme, the choice predictor is reset to weakly-taken, and one bank of the direction predictor is reset to weakly-not-taken and the other bank is weakly-taken.

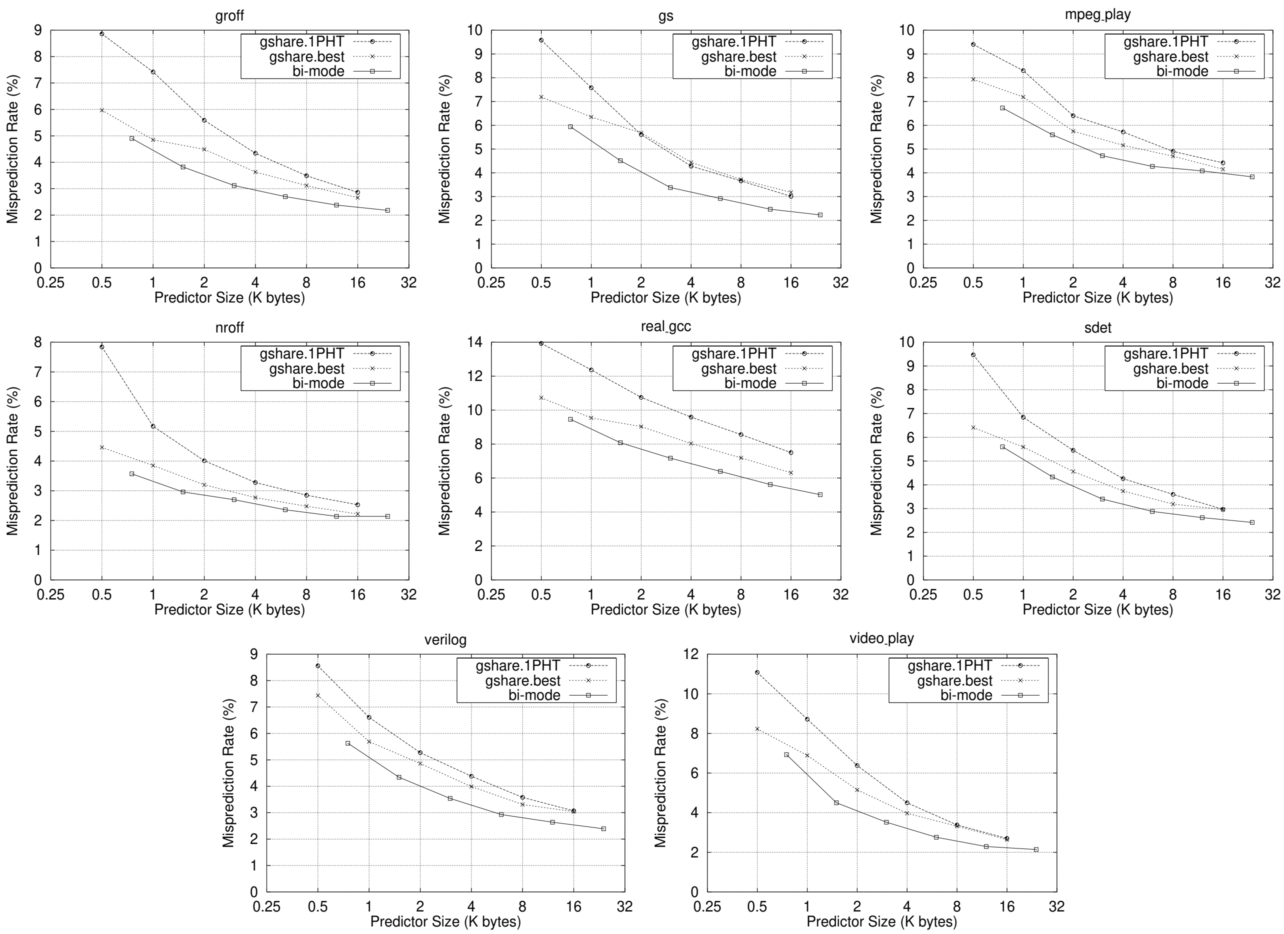

Figure 4: Misprediction rates for IBS-Ultrix

er to predict than weakly biased branches in dynamic branch predictors, and this was confirmed by Chang *et al.* [Chang94]. In the same study, they also measured the distribution of branch biases for SPEC CINT92. Their measurement showed that on average about 50% of total dynamic branches are attributed to the static branches that are biased in either the taken or not-taken direction for more than 90% of the time.

In this section, our analysis extends the idea of bias to the dynamic branch substreams that arrive at each two-bit counter in the second-level table. Using this concept, we will demonstrate the advantages and drawbacks of two kinds of information used in the two-level scheme, specifically, the branch address and global history. The analysis allows us explain why the bi-mode scheme can improve on current dynamic branch predictors.

4.1 Bias measurement for global-history based schemes

As we have noted before, the reason that two-level dynamic branch predictors can achieve higher prediction accuracy than the traditional two-bit counter scheme proposed by Smith [Smith81] is because, in addition to the branch address, they incorporate the branch history information to form the index for the second-level two-bit counter table. The index for the second-level table divides the dynamic branch stream into substreams that are directed to a saturating two-bit counter. Ideally, the index should generate highly biased substreams so that the value of the saturating counter selected by the index can stay at one of the saturated values most of time. Global history, compared to the branch address, can divide a dynamic branch stream into more highly biased substreams, as we will show later. However, if the indexing method mixes oppositely biased substreams together, then destructive aliasing can arise and the associated counter will perform badly as a predictor, because it will oscillate between the two saturated values. Our study will compare using branch addresses with global history to separate out oppositely biased substreams, and how aliasing can degrade the performance of two-level schemes using global history.

To contrast the benefits of address and global history

branch address, i	dynamic count when using counter c, $\|s_{ic}\|$	count of taken outcomes when using counter c	bias class	normalized count from i=b to c, N_{bc}
0x 001	12	11	ST	12/50 = 24%
0x 005	20	1	SNT	20/50 = 40%
0x 100	8	3	WB	8/50 =16%
0x 150	10	1	SNT	10/50 = 20%

Table 3: An example of calculating the normalized count for a counter *c*

bits, we consider two alternative two-level gshare style predictors. Both have the same size second-level tables, 256 counters, but differ in that one employs more history bits, representing history-indexed schemes, while the other represents address-index schemes. The first scheme xors 8 bits of branch address with 8 bits of global history to form the index into the second-level table ("history-indexed"). The second scheme xors 8 bits of branch address with only 2 bits of global history as the index ("address-indexed").

We define three bias classes on a stream of branch outcomes: 1) strongly taken (ST) if the outcomes are taken 90% of the time or more; 2) strongly not taken (SNT) if the outcomes are not taken 90% of the time or more; and 3) weakly biased (WB) if the neither of the above apply.

We are interested in the stream of branch outcomes, s_{ij}, from a particular static branch, i, to a particular prediction counter, j. This stream belongs to one of the three bias classes, i.e., exactly one of the following is true: $s_{ij} \in$ ST, $s_{ij} \in$ SNT, or $s_{ij} \in$ WB. A good indexing method will create these streams so that the following two conditions hold:

1. The number of streams that are in the WB class is kept small.

2. Most of the streams incident on a particular prediction counter, $j = c$, belong to only the ST class, or alternatively, only the SNT class, i.e., $s_{ic} \in$ ST for most i, or $s_{ic} \in$ SNT for most i. A counter should <u>not</u> see an even mix of streams from both classes or its prediction ability will be reduced.

Condition 2 actually states that one of the two strongly biased class should dominate the other strongly biased class at a counter. When this domination occurs, the counter will be biased at one saturated value with little destructive interference. We will refer to the more frequent strongly-biased class at a counter as the *dominant* class, and the other less frequent strongly-biased class as the *non-dominant* class.

To be more precise, we should consider streams weighted by their lengths. If $|s_{ij}|$ is the number of outcomes in the stream s_{ij}, we define the normalized count that a branch, $i = b$, contributes to a particular prediction counter, $j = c$, to be:

$$N_{bc} = \frac{|s_{bc}|}{\sum\limits_{\text{over all static branches } i} |s_{ic}|}$$

Thus the two conditions become:

1. $(\Sigma_i(N_{ic})$ | for those i such that $s_{ic} \in$ WB) << $(\Sigma_i(N_{ic})$ | for those i such that $s_{ic} \notin$ WB)

2. $(\Sigma_i(N_{ic})$ | for those i such that $s_{ic} \in$ ST) should differ greatly from $(\Sigma_i(N_{ic})$ | for those i such that $s_{ic} \in$ SNT). In an ideal situation, one of the sums should be 0.

Table 3 illustrates the normalized count resulting from three streams incident on the same counter c. In this example, there is a total of four static branches (i = 1,..,4) whose addresses are 0x001, 0x005, 0x100 and 0x150, respectively, that used the two-bit counter c for prediction during the program execution (they may also use other counters too). These four streams fall into different bias classes with respect to c. The normalized count of ST class at the counter c is 24%, the SNT class is 60% (40%+20%), and the WB class is 16%. Because the SNT class is more frequent than the ST class, the SNT is the dominant class in the counter c, and the ST is the non-dominant class. In fact, Table 3 shows an undesirable situation because the indexing method has done a poor job of separating the bias classes and the SNT class is not overwhelmingly dominant.

Figure 5 illustrates the bias classes for all of the prediction counters for the *gcc* benchmark. We have performed the same experiments for other SPEC benchmarks, and we select *gcc* because it is representative of the results from the other benchmarks, see [SechrestLeeMudge96]. The X axis lists all the counters in the second-level table, and the Y axis represents the normalized counts of the three bias classes in each counter. The counters listed in the X axis are sorted according to the normalized dynamic frequency of WB class. It can be seen that the area size of WB region of the history-indexed scheme is smaller than that of the address-indexed one. This suggests that the scheme employing more branch history can generate more highly biased substreams for predictors. If there is no harmful aliasing problem in the history-index scheme, i.e., each counter only needs to deal with substreams of one bias class, the prediction accuracy will be very high [TalcottNemirovskyWood95, YoungGloySmith95].

However, in the usual situation where harmful aliasing does exist, the performance of the history based scheme degrades. As shown in the same figure (Figure 5), the non-

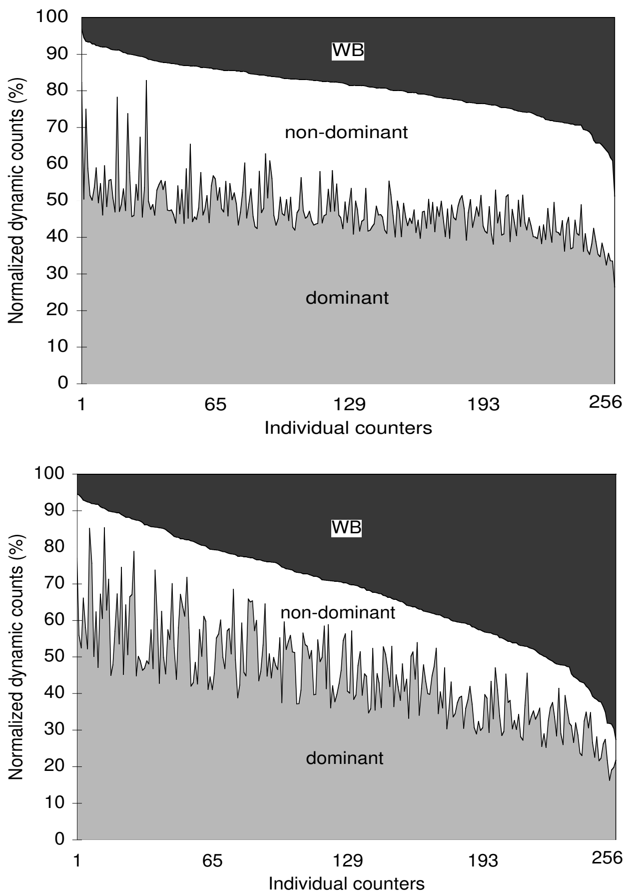

Figure 5: Bias breakdown for the gshare scheme in the SPEC CINT95 *gcc* benchmark. History-indexed on the top, address-indexed on the bottom

This figure shows the bias of branch outcome substreams arriving at each of 256 counters in a second-level table. The top graph is for the history-index scheme (8 bits of branch address xor-ed with 8 bits of global history); the bottom graph is for the address-indexed scheme (8 bits of branch address xor-ed with 2 bits of global history). These two graphs illustrate the difference between the two indexing methods. The address-indexed scheme suffers from a larger number of weakly biased (WB) branch substreams, while the history-indexed scheme suffers from more non-dominant substreams, implying a high degree of destructive interference between strongly but oppositely biased streams (between the SNT and ST classes).

dominant class in the history-indexed scheme is larger than the one in the address-indexed scheme. In other words, although the history-indexed selects the greater number of highly biased substreams, it does not separate the taken and not-taken ones as well as address-indexed scheme.

To summarize the analysis above, an ideal dynamic branch predictor should generate as few weakly biased substreams as possible; in other words, the area of the weakly biased region should be as small as possible. At the same

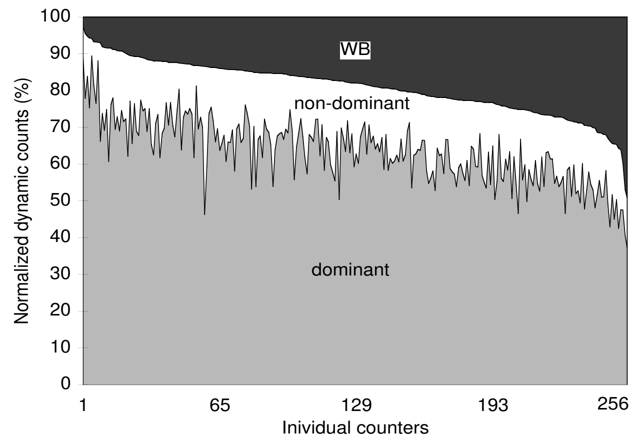

Figure 6: Bias breakdown for the bi-mode scheme

This figure shows the bias of branch substreams of each counter for the bi-mode scheme. This bi-mode scheme has a 128-counter choice predictor and two 128-counter direction predictors. As shown in the figure, the dominant substreams dominate most of the counters of the second-level table, implying that interference is reduced significantly.

time, the resulting substreams merged at each counter should be as unidirectional as possible; in other words, the dominant area in Figure 5 should be large. Unfortunately, neither the address-indexed scheme nor the history-indexed scheme can achieve both of these two design goals simultaneously.

4.2 Bias measurement for the bi-mode scheme

In this subsection, we repeat the analysis above for the bi-mode prediction scheme. The configuration under examination has a 128-counter choice predictor indexed by the branch address and two banks of 128 counters in the second-level table, each of which is indexed by 7 bits of branch address xor-ed with 7 bits of global history. This system has about 50% more bytes than the predictors in the previous subsection, so the following analysis should be viewed qualitatively.

Figure 6 presents the measurement results. It can be seen that the weakly biased class in the bi-mode scheme is kept as small as the one in the history-indexed scheme, indicating that the advantage of employing history information is preserved. On the other hand, Figure 6 also shows the bi-mode scheme yields a much larger area for the dominant class than the history-indexed scheme, implying that destructive aliasing has been reduced.

The counting arguments that we employ to classify the ST, SNT, and WB classes are open to the criticism that they do not capture the order in which the ST and SNT runs appear. For example, it is undesirable for them to be intermixed so that the stream changes between the two classes. As a final experiment, we have counted the numbers of

	Dominant	Non-dominant	WB
history-indexed	3,826,578	3,589,689	2,252,874
bi-mode	3,685,544	2,717,563	2,226,353

Table 4: Numbers of changes between different bias classes for the history-indexed and bi-mode schemes

This table shows numbers of changes between branch outcome streams of different bias classes in the history-indexed and bi-mode schemes. We first count changes for each bias class in a counter, and then accumulate the counts of all counters for a scheme. For example, the count for the dominant class of the history-indexed scheme is the total number of changes of the dominant class due to interference by the other two classes in the scheme.

changes between bias classes due to interference. Table 4 shows the results for the history-indexed and bi-mode schemes. The bi-mode scheme has fewer changes, implying that its ST and SNT classes are less intermingled. This means less interference, and further illustrates why our proposed prediction scheme perform better than conventional two-level schemes.

4.3 Breakdown of misprediction for the gshare and bi-mode schemes

We have also measured the misprediction contributed by three biased classes for the gshare and bi-mode schemes. Again, for the gshare scheme, the configurations using fewer global history bits and more global history bits are both included for comparison.

Figure 7 presents the measurement results for the *gcc* benchmark. Three different sizes are studied for the branch predictors: 256, 1024, and 32,768 counters in the second level table. For each configuration, the misprediction is broken down to three categories according to the bias classes. In other words, the sum of mispredictions from three classes is the misprediction rate for the corresponding scheme. For the gshare predictors of the same size, the one using fewer global history bits always has the least error from the strongly-biased classes, but it suffers from poor prediction for the weakly-biased substream. The bi-mode scheme keeps a reduced error for the weakly biased class, while successfully reducing the error from strongly-biased classes.

4.4 *go* benchmark

In Section 3, we noted that the bi-mode scheme was not the best for the *go* benchmark. In this section, we provide further analysis.

The *go* benchmark is intrinsically hard to predict because about half of its dynamic branches are in the WB class. Figure 8 shows the misprediction contributed by the three bias classes for the *go* benchmark. It is clear that for all the

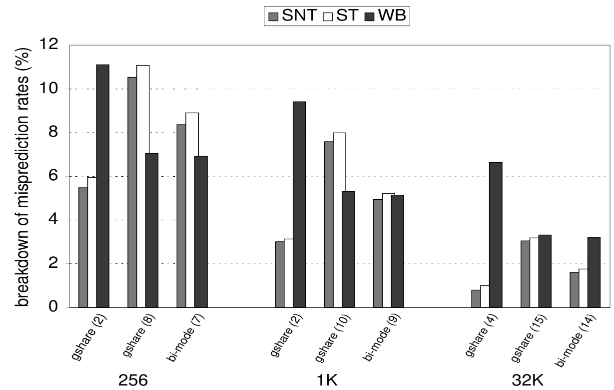

Figure 7: Misprediction contributed by three bias classes in *gcc*

In this figure, three schemes are compared, a gshare using fewer history bits (representing the address-indexed scheme), a gshare using more history bits (history-indexed) and the bi-mode scheme. For each scheme, three different sizes of second-level tables are examined: 256, 1K and 32K counters. gshare (*m*) represents a gshare scheme that uses *m*-bit global history, while bi-mode (*m*) represents a bi-mode scheme that uses *m*-bit global history for its direction predictors. The choice predictor of the bi-mode scheme is half the size of its second-level table. As shown in the figure, the address-indexed scheme always has larger misprediction for the WB class. The history-indexed scheme has less misprediction for the WB class, but has more for the SNT and ST classes due to interference. The bi-mode scheme reduces error from the WB and reduces, in most cases, the misprediction for the SNT and ST by removing interference.

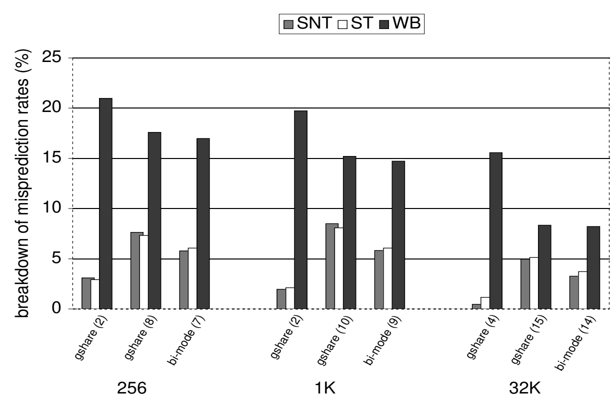

Figure 8: Misprediction contributed by three bias classes in *go*

This figure shows misprediction due to three bias classes for the *go* benchmark. As the same in Figure 7, three schemes with three different sizes of second-level tables are compared. As shown in the figure, the misprediction due to the WB class dominates in *go* for three schemes, and thus the interference between SNT and ST classes is not the major concern. To improve prediction accuracy for *go*, more history bits should be used because it is an effective way to remove the WB class. Note that as more history bits are used, the relative misprediction rates due to the WB class becomes smaller.

schemes and configurations the misprediction for the WB class dominates—destructive aliasing is not the major concern. There is not much room for the bi-mode scheme to improve because it is targeted at eliminating harmful aliasing rather than improving prediction for the weakly biased substreams. As observed in the previous subsection, the dynamic frequency of the weakly biased class is mainly determined by the number of global history bits used. From Figure 8, we see that the error of the WB class is reduced as more global history bits are applied. The prediction accuracy for programs like the *go* benchmark will only improve if more global history information is employed so that more strongly biased substreams can be generated.

5. Concluding Remarks

In this paper, a new global-history based branch prediction scheme, the bi-mode predictor, is proposed. It is designed to improve predictions by eliminating the aliasing in dynamic branch predictors. Its success relies on dynamically determining the taken or not-taken direction with an accurate but simple choice predictor. This classification can help removing much of the destructive aliasing while keeping the harmless aliasing together for the two-bit counter tables.

A detailed analysis on the mechanism of two-level scheme's index system was also presented in the paper. From the analysis we found that by using more global history bits in an index can move more branches from the weakly biased group to the strongly biased group, but these indices suffer from destructive aliasing. Using more branch address bits reduces the destructive aliasing but increases the weakly biased group. The benefits of using branch addresses and global history cannot be preserved in current two-level schemes simultaneously, but they can in the bi-mode scheme.

The bi-mode scheme can outperform other dynamic predictors, yet there is still room for improvement. One potential shortcoming for the bi-mode scheme is that, though it can distinguish strongly taken and strongly not-taken substreams as shown in Figure 6, it can still suffer from interference between the weakly biased and strongly biased substreams. Therefore, there are at least two directions for the future work: one is to find a cost-effective way to reduce the weakly biased substreams, and the other is to further separate the weakly-biased substreams from the strongly-biased substreams for the counters. We are currently investigating these issues.

Acknowledgment

This work was supported by DARPA contract DAA H04-94-G-0327.

References

[Chang94] Chang, P., Hao, E., Yeh, T., and Patt, Y., "Branch Classification: a New Mechanism for Improving Branch Predictor Performance," *IEEE Micro-27*, Nov. 1994.

[ChangEversPatt96] Chang, P., Evers, M., and Patt, Y., "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference," *International Conference on Parallel Architecture and Compilation Techniques*, Oct. 1995.

[EustaceSrivastava95] Eustace, A. and Srivastava, A., "ATOM: A flexible interface for building high performance program analysis tools," *Proceedings of the Winter 1995 USENIX Technical Conference on UNIX and Advanced Computing Systems*, 303-314, Jan. 1995.

[FisherFreudenberger92] Fisher, J.A., and Freudenberger, S.M. "Predicting Conditional Branch Directions From Previous Runs of a Program, Proc." *5th Annual Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1992.

[Gwennap95] Gwennap, L. "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report,* Vol. 9, No.2, Feb. 16, 1995.

[Gwennap96] Gwennap, L. "Digital 21264 Sets New Standard," *Microprocessor Report,* Vol. 10, No. 14, Oct. 28, 1996.

[Hwu93] Hwu, W-W., Mahlke, S.A., Chen, W-Y., Chang, P-P., Warter, N.j., Bringmann, R.A., Ouellette, R.G., Hank, R.E., Kiyohara, T., Haab, G.E., Holm, J.G., and Lavery, D.M., "The superblock: An effective technique for VLIW and superscalar compilation," *Journal of Supercomputing,* 7(9-50), 1993.

[Lee97] Lee, C-C., "Optimizing High Performance Dynamic Branch Predictors," *Ph.D Dissertation,* Univ. of Michigan, Ann Arbor, Nov. 1997.

[McFarling93] McFarling, S. "Combining Branch Predictors," *WRL Technical Note TN-36*, Jun. 1993.

[MichaudSeznecUhlig97] Michaud, P., Seznec, A., and Uhlig, R., "Trading Conflict and Capacity Aliasing in Conditional Branch Predictors," *Proc. of the 24th Ann. Int. Symp. on Computer Architecture,* May 1997.

[PnevmatikatosSohi94] Pnevmatikatos, D.N. and Sohi, G.S., "Guarded Execution and Branch Prediction in Dynamic ILP Processors," *Proc. of the 21st Ann. Int. Symp. on Computer Architecture,* Apr. 1994.

[PanSoRahmeh92] Pan, S.T., So, K., and Rahmeh, J.T., "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proceedings of the 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 1992.

[SechrestLeeMudge96] Sechrest, S, Lee, C-C, and Mudge, T, "Correlation and Aliasing in Dynamic Branch Predictors," *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[Smith81] Smith, J.E. "A Study of Branch Prediction Strategies," *Proceedings of the 8th International Symposium on Computer Architecture,* 135-148, May 1981.

[SPEC95] SPEC CPU'95, Technical Manual, Aug. 1995.

[Sprangle97] Sprangle, E., Chappell R., Alsup, M., and Patt, Y., "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference," *Proc. of the 24th Ann. Int. Symp. on Computer Architecture,* May 1997.

[TalcottNemirovskyWood95] Talcott, A.R., Nemirovsky, M., and Wood, R.C., "The Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance," *Proceedings of the 3rd International Conference on Parallel Architectures and Compilation Techniques,* Jun. 1995.

[Uhlig95] Uhlig, R., Nagle, D, Mudge, T., Sechrest, S., and Emer, J. "Instruction Fetching: Coping with Code Bloat," *Proceedings of the 22th International Symposium on Computer Architecture*, Italy, Jun. 1995.

[YehPatt91]Yeh, T-Y. and Patt, Y. "Two-Level Adaptive Training Branch Prediction," *Proceedings of the 24th International Symposium on Microarchitecture,* 51-61, Nov. 1991.

[YehPatt92]Yeh, T-Y. and Patt, Y. "Alternative Implementations of two-level adaptive branch predictions," *Proceedings of the 19th International Symposium on Computer Architecture,* 124-134, May 1992.

[YehPatt93]Yeh, T-Y. and Patt, Y. "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *Proceedings of the 20th International Symposium on Computer Architecture,* May 1993.

[YoungGloySmith95] Young, C., Gloy, N., and Smith, M. "A Comparative Analysis of Schemes for Correlated Branch Prediction," *Proceedings of the 22th International Symposium on Computer Architecture,* Italy, Jun. 1995.