Proc. World MultiConf on Systemics, Cybernetics & Info

Design and Performance Evaluation of Global History Dynamic Branch Predictors

Chih-Chieh Lee
Digital Equipment Corp.
181 Lytton St., Palo Alto, CA 94086
cclee@pa.dec.com

I-Cheng K. Chen
California Microprocessor Division
Advanced Micro Devices
One AMD Place, Sunnyvale, CA 94088
kevin.chen@amd.com

Trevor Mudge
EECS Department
University of Michigan
1301 Beal Ave., Ann Arbor, MI 48109
tnm@eecs.umich.edu

Abstract

The importance of accurate branch prediction to future processors has been widely recognized. The correct prediction of conditional branch outcomes can help avoid pipeline bubbles and attendant loss in performance. In order to achieve high prediction accuracy, numerous dynamic branch prediction schemes that exploit branch correlation have recently been proposed. Several of the best predictors are the gshare, agree, filter, skewed and bi-mode predictors.

However, despite the intensive research work to propose these new schemes, there is no direct and comprehensive performance comparison among them. Such comparison is essential to guiding the design of future microprocessors. Therefore, in this paper, we conduct extensive experiments to calibrate the performance of each prediction scheme. Furthermore, we discuss the design philosophy and underlying mechanism for these schemes, and contrast their relative advantages and disadvantages. Among the schemes examined, we find that the skewed predictor performs the best for small budgets, while the bi-mode predictor outperforms others for large budgets.

Keywords: Dynamic Branch Predictor, Gshare, Agree, Filter, Skew, Bi-mode.

1. Introduction

As the design trends of modern microprocessors move toward wider issue and deeper pipelines, accurate branch prediction becomes essential to exploring the full performance of microprocessors. A good branch prediction scheme can increase the performance of a microprocessor by eliminating the instruction fetch stalls in the pipelines. As a result, different styles of branch prediction schemes have been proposed and, among them, the global history dynamic branch predictors have been shown to achieve high accuracy [10]. To make predictions, these global history branch predictors exploit the correlation of combined outcomes from neighboring branches (which form a *global* history of branch outcomes). Nevertheless, the global history predictors are still limited by destructive interference that occurs when two branches have the same global history, but opposite biases. To reduce this destructive interference, many variations of global predictors have been designed and proposed. Several of the best predictors are the gshare, agree, filter, skewed and bi-mode predictors.

However, regardless of all the research efforts to design these new branch prediction schemes, there is no direct comparison to measure their performance. In order to assess the relative performance of each scheme, this comparison has to be made under the same set of benchmarks, input data sets, instruction sets, and hardware budgets. Such comparison is essential to guiding microprocessor designers in selecting the most appropriate branch prediction schemes. To provide this comprehensive comparison, we conduct extensive simulations to calibrate the performance of each prediction scheme. Moreover, to contrast their advantages and disadvantages, we discuss the design philosophy and underlying mechanism for these schemes.

This paper is organized into five sections. In section 2 we describe the design philosophy and underlying mechanism of various global history dynamic predictors. In section 3 we describe our simulation environments and benchmarks used. Section 4 presents simulation results and discussion. Finally, we present some concluding remarks in section 5.

2. Description of global history dynamic predictors

2.1 Gshare predictor

McFarling [5] proposed a variant of global history branch predictors, referred to as *gshare*, in which the global history is xor-ed with the low-order address bits of a branch to form an index, as shown in Figure 2. This index is then used to select a two-bit saturating up-down counter from the second-level table. Depending on the sign bit of the selected two-bit counter, the branch is either predicted as taken or not taken.

The gshare scheme attempts to reduce interference by randomizing the index to the second-level table using xor-ing. By xor-ing the global history pattern with branch addresses, the gshare scheme can produce new distinct indexing values for the counters, each associated with a static branch. This xor-ing can reduce interference between branches while retaining the advantages of using long global history to exploit branch correlation. However, this scheme offers limited benefits, because randomization can only "blindly" separate aliased branches. Consequently, this process may reduce destructive interference simply by chance.

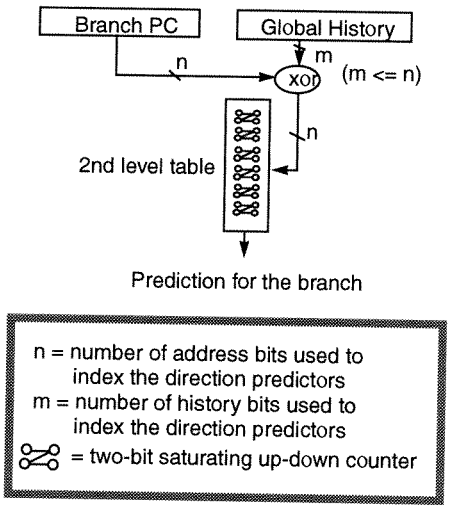


Figure 1: Diagram for the gshare predictor

2.2 Agree predictor

The agree predictor, proposed by Sprangle *et al.* [8], attempts to reduce interference for the global history predictors by changing the usage and interpretation of two-bit counters in the second-level table.

In original dynamic branch predictors, such as the *gshare* scheme, the value of a two-bit counter in the second-level table is directly used to predict whether a branch is taken or not. When the branch outcome is resolved, the counter value is updated with the true outcome. For example, the counter is increased by one if the outcome is taken and is decreased by one if it is not-taken. Then, if the counter value is greater than or equal to half of the maximum value, the branch is predicted taken; otherwise, the branch is predicted not taken.

The authors of the agree predictors realized that if two oppositely biased branches are aliased to the same counter, the counter value will bounce back and forth between two saturated values. To reduce this aliasing, they proposed the agree predictor, as shown in Figure 2. In the agree predictor, each static branch is assigned a biasing bit, which is stored in the branch target buffer (BTB) or in the cache. In addition, the meaning and usage of two-bit counters have also changed: instead of predicting the direction for a branch, the two-bit counters predict if the branch will go in the direction indicated by the biasing bit. In other words, the counter in the second-level table has now to *agree* whether or not the biasing bit should be used for predicting the outcome of the current dynamic branch instance. In effect, the counters measure the confidence on the validity of the biasing bits. If the branch outcome matches with the biasing bit, the counter is increased by one; otherwise, the counter is decreased by one. Depending on the sign bit of the counter, the branch is either predicted to go in the direction indicated by the biasing bit or in the opposite direction.

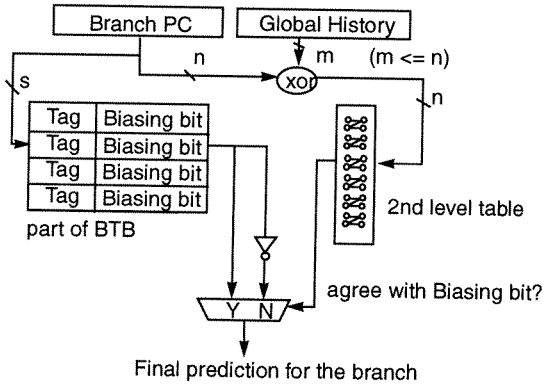


Figure 2: Diagram for the agree predictor

In the agree predictor, if two strongly but oppositely biased branches alias to the same counter, they may no longer cause destructive interference. This is because, if the biasing bits for both branches are set correctly, the aliased counter should agree that both biasing bits are correct.

However, good prediction now depends on setting the biasing bits correctly. Ideally, the biasing bit should indicate the direction that the corresponding branch takes most of time during program execution. Obtaining the optimal value for this biasing bit requires profiling the program with the same data set. Since the optimal value is difficult to obtain, Sprangle *et al.* have suggested a more general technique in which the biasing bit can be estimated by the first outcome of each static branch. In this paper, we also use the same technique for the biasing bit estimator in the agree predictor.

2.3 Filter predictor

Chang *et al.* have also proposed a scheme to reduce interference for the global scheme [1]. Their idea is based on the observation that most branches are predicted accurately by their last outcomes without the need of using a global predictor. They proposed a scheme, shown in Figure 3, which adds an n-bit counter for each static branch to record the number of times the same outcome is repeated. If a branch has 2^n repeating outcomes that are the same, it is simply predicted by its last outcome without consulting or updating the global history predictor; otherwise, it is predicted by the global history predictor. In effect, this scheme uses n-bit counters to separate and filter out easy-to-predict branches (whose last 2^n outcomes are the same) from the rest, so that interference in the global history predictor can be significantly reduced. The interference in the global history predictor is greatly reduced because a large percentage of branches are predicted using the last outcome and thus only the remaining branches are actually used to update the global history predictor. In addition, Chang *et al.* showed that, usually 3-bit or 4-bit counters can deliver good prediction accuracy.

One requirement for this design is that it needs to identify the ownership of the n-bit counters, because each counter is counting the repeating times of the same outcome for a particular branch, and thus it should not be polluted by other

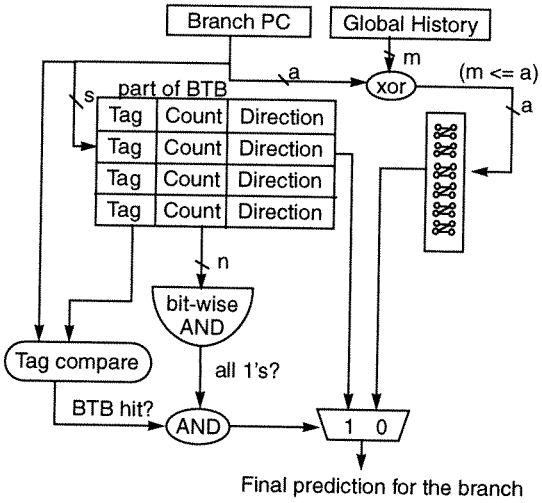


Figure 3: Diagram for the filter predictor

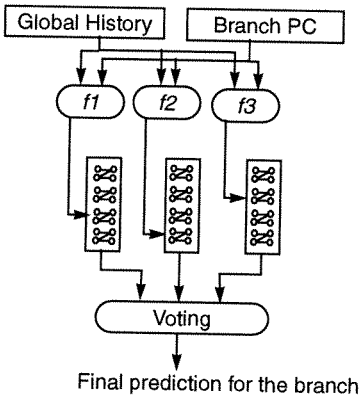


Figure 4: Diagram for the skewed predictor

branches. As a result, the scheme requires extra hardware for storing the branch address, or the scheme has to be implemented in the branch target buffer (BTB) so it can use the branch address tags for this purpose. This may limit the flexibility of the scheme.

2.4 Skewed predictor

Another scheme to reduce interference, referred to as the skewed branch predictor, has been proposed by Michaud *et al.* [6]. Michaud *et al.* found that the interference in the second-level table is similar to the conflict miss in a cache system. To reduce conflicts in the predictor, they proposed a reorganization of the second-level table. Specifically, the original second-level table is divided into three sub-tables, all three of which record counts for the histories and make prediction for every branches, as shown in Figure 4. The final prediction of the predictor is then based on the majority vote among the three sub-tables. The key ingredient in this predictor is that the three sub-tables use three different indexing functions so that the chance for two branches to collide in all three sub-tables at the same time is little. A potential drawback is that using three sub-tables to record history for each branch may reduce the effective capacity of the second-level table in a skewed predictor.

To compensate for this capacity loss, Michaud *et al.* adopted a partial update policy that will not update the incorrect sub-table when the final prediction is found to be correct. In other words, when the final prediction is known to be correct (so at least two of the three sub-tables voted correctly) and there is a sub-table which made an incorrect vote, the incorrect sub-table will not be updated with the branch outcome. Michaud *et al.* realized that when a final prediction is correct, the sub-table which made an incorrect vote may actually be recording the history for another branch. Therefore, this sub-table should remain unchanged so that it can keep the history information for other branches. They showed that this partial update policy can compensate for the capacity loss and thus improve the prediction accuracy significantly.

The skewed predictor has been shown to be an improvement over the conventional global history schemes. However, the three indexing functions need to be carefully selected to achieve the goal of reducing interference. Michaud *et al.* have found a set of indexing functions that are not costly to implement but may still lengthen the clock cycle because of the address decoder for the sub-tables. The indexing functions used for the skewed predictor examined in this paper will be the same as those proposed by Michaud, which are described next.

Supposed there is a $2n$-bit index vector, V, which is formed by concatenating the address (c bits) and history bits (r bits, $c+r = 2n$). This V is decomposed into two n-bit vectors, $V1$ and $V2$. $V1$ is the low-order half of V, while $V2$ is the upper half, i.e., both V1 and V2 are n-bit vectors and one of them can contain address and history bits if c and r are not equal to n. V is then equivalent to the concatenated vector ($V2$, $V1$). Let $(y_n, y_{n-1}, .., y_1)$ be the bit representation of an n-bit vector. A hash function, H, is defined as follows,

$$H: (y_n, y_{n-1}, .., y_1) \rightarrow ((y_n \text{ xor } y_1), y_n, y_{n-1}, .., y_3, y_2),$$

and its inverse, H^{-1}, is defined as:

$$H^{-1}: (y_n, y_{n-1}, .., y_1) \rightarrow (y_{n-1}, y_{n-2}, .., y_2, y_1, (y_n \text{ xor } y_{n-1}))$$

Three indexing functions are then defined as follows, using bit-wise xors as mappings of $2n$-bit vectors to n-bit vectors,

$$f1: (V2, V1) \rightarrow H(V1) \text{ xor } H^{-1}(V2) \text{ xor } V2$$

$$f2: (V2, V1) \rightarrow H(V1) \text{ xor } H^{-1}(V2) \text{ xor } V1$$

$$f3: (V2, V1) \rightarrow H^{-1}(V1) \text{ xor } H(V2) \text{ xor } V2$$

In these three indexing functions, some index bits require three xor operations. This may increase the access time to the second-level table.

Though three different hashing functions are important to reduce chances of interference, the partial updating of the sub-tables is key to the performance of the skewed predictor. When the branch outcome is resolved and the predictor is correct, only

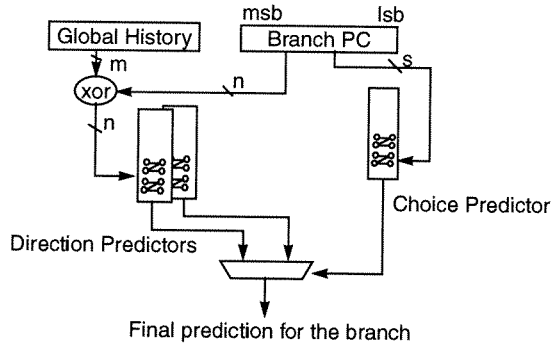


Figure 5: Diagram for the bi-mode predictor

the sub-tables that make correct predictions are updated. In contrast, when the prediction is not correct, all sub-tables are updated. This selective update policy is crucial to the prediction accuracy of the skewed predictor, but it may complicate the logic design.

Michaud *et al.* have also proposed an enhanced skewed predictor, in which the *f1* hashing function is replaced with partial branch address bits [6]. This enhanced scheme can further increase the prediction accuracy, so it will be used in the comparison throughout this paper.

2.5 Bi-mode predictor

To further reduce destructive interference in global history schemes, Lee *et al.* proposed the bi-mode branch predictor [3]. The bi-mode predictor splits the second-level table of a gshare into two halves, shown in Figure 5. Given a history pattern, two counters, one from each half, are selected. These counters are referred to as the direction predictors. Another two-bit counter table, indexed by the branch addresses only, is used to provide a final selection for these two counters. This counter table providing selection is referred to as the choice predictor. The final prediction is then made by the state of the counter selected from the direction predictors and, equally importantly, only the selected counter will be updated with the branch outcome; the status of the un-selected one, will not be altered. The choice predictor is always updated with the branch outcome.

The bi-mode scheme delivers high prediction accuracy because, although the behavior of global history patterns are still kept in the second-level table, they are dynamically classified before being stored to reduce destructive interference. The global history patterns 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. These two groups of branches correspond to the strongly-taken and strongly-not-taken cases, and each group is sent to its own half of direction predictor respectively. The effect of the choice predictor is to separate the destructive interference streams while keeping the harmless interference streams together.

	Benchmarks	Dynamic conditional branches	Static conditional branches	Static branches constituting 90% of total dynamic conditional branches
SPEC CINT95	compress	10,114,353	482	25
	xlisp	25,008,567	636	50
	perl	39,714,684	1,974	156
	vortex	27,792,020	6,599	354
	go	17,873,772	5,112	1,021
	gcc	26,520,618	16,035	3,244
IBS-Ultrix	nroff	22,574,884	5,249	228
	groff	11,901,481	6,333	459
	sdet	5,514,439	5,310	508
	mpeg_play	9,566,290	5,598	532
	video_play	5,759,231	4,606	757
	verilog	6,212,381	4,636	850
	gs	16,308,247	12,852	1,160
	real_gcc	14,309,867	17,361	3,214

Table 1: Benchmarks characteristics

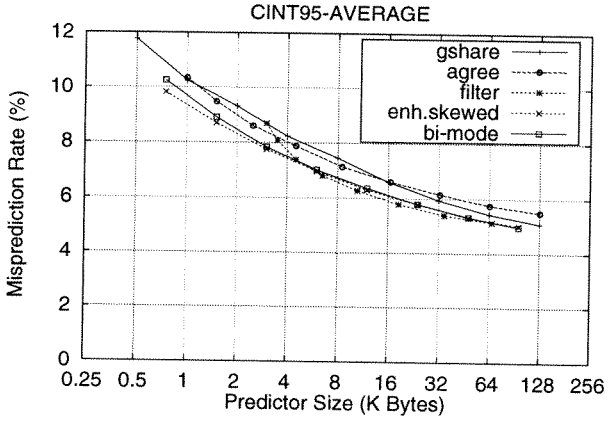
3. Simulation Environment and Benchmarks

To assess the performance for each scheme, we conduct trace-driven simulation using the SPEC CINT95 and the Ultrix version of the Instruction Benchmark Suite (IBS). The characteristics of these benchmarks are listed in Table 1.

SPEC CINT95 [7] programs were compiled on an Alpha 21064-based workstation with the OSF/1 C compiler using the -O optimization. We employed DEC's ATOM instrumentation tool [2] to capture all user-level instructions for the CINT95 traces, including shared libraries.

The IBS-Ultrix benchmarks are a set of applications running under Ultrix 3.1. The traces were collected through hardware monitoring of a MIPS R2000-based workstation by Uhlig *et al.* [9]. These traces include both instructions executed from the user applications and the operating system.

The main difference between these two set of benchmarks is their footprint sizes. By examining the number of static branches constituting 90% of dynamic conditional branches shown in Table 1, we can see that IBS-Ultrix benchmarks on average have more static branches than SPEC CINT95. This is because IBS also include branches from the operating system.



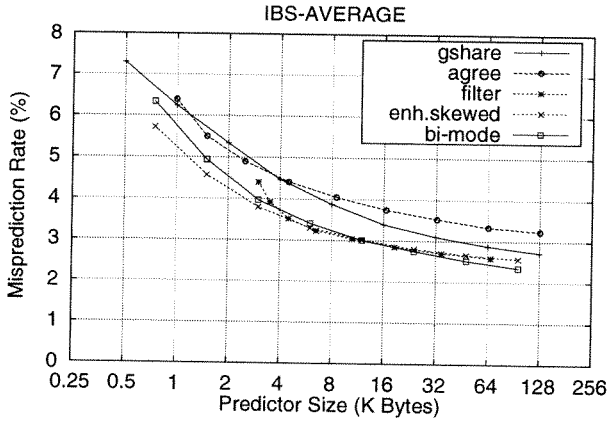


Figure 6: Comparison among gshare, agree, filter, enhanced skewed, and bi-mode schemes

2nd-level table sizes: (2^n counters)		n=11	12	13	14	15	16
# of global history bits employed	SPEC CINT95	3	6	6	9	10	13
	IBS-Ultrix	2	4	8	9	10	11

Table 2: The gshare configurations that yield the best average prediction accuracy

4. Simulation results and discussion

In this section, we provide cost-effective performance comparisons for all schemes discussed in section 3, in which cost is measured by the amount of storage in bytes that each requires. The average results are summarized in Figure 6. This figure compares the average misprediction rates for the SPEC CINT95 and IBS suites among the gshare, agree, filter, enhanced skewed, and bi-mode schemes. Figure 7 details the performance comparison for each individual benchmark. For each of the five schemes, we exhaustively examined all possible pair-wise combinations of history length and address length for each hardware budget. Then, we present the best result for each scheme. For example, given 2^{11} counters, the second-level table of the gshare scheme has 12 different configurations (of varying history and address lengths). The best configurations of the gshare scheme for the range of budget examined are listed in Table 2.

As can be seen from the figures, the filter, enhanced skewed and bi-mode schemes perform the best. Comparing these three schemes, we can see that the enhanced skewed predictor achieves the highest accuracy for small budgets, showing the advantage of employing complicated XOR index functions in small predictors. As the budget becomes larger, the bi-mode and filter schemes perform the best, and the bi-mode scheme eventually outperforms all other schemes for the largest budget. Note that the bi-mode scheme is naturally easier to implement than the filter scheme. The filter scheme proves to be an effective way to filter out the strongly-biased branches for global history predictors. However, to determine the best value of the n for the n-bit filtering counter is difficult, because different benchmarks require different values to achieve the best performance. This is not an issue with the bi-mode scheme; a choice predictor that uses two-bit counters can always deliver the best performance. Moreover, identifying the ownerships of the filtering counter is crucial to the performance. We have studied a filter scheme in which the BTB has no tags [4]. In this scheme, the filtering counter may be shared by several branches. The results showed that if there is no tag available to identify the ownership of the counters, the performance of the filter scheme can be significantly degraded. Since the filtering counters need to be associated with the BTB, the filter scheme may limit the design flexibility of high-speed microprocessors.

It is interesting to compare closely the mechanism of the bi-mode and enhanced skewed predictors. Each predictor has its own advantages and disadvantages. The enhanced skewed predictor treats all three banks equally; when the address-indexed bank does not do a good job, the predictor can still rely on the other two banks to make the final prediction. In contrast, the bi-mode scheme always relies on its address-indexed choice predictor to select a bank for the final prediction. The choice predictor may not be able to make good selections all the time, thus limiting the overall prediction accuracy. This limitation occurs when the choice predictor has relatively high interference as in the case of small budgets.

The bi-mode scheme, on the other hand, has shorter training periods for new branches, compared to the skewed predictor. In the bi-mode scheme, one bank of the second-level table is mostly biased in the not-taken direction, while the other bank is biased in the taken direction. Since most branches in realistic programs are strongly biased, the bank selected by the choice predictor has typically been trained by some other similarly biased branches; it is enough for just the choice predictor to capture the bias of a new branch for the bi-mode predictor to start making accurate predictions. In contrast, the skewed predictor requires at least two of the banks to be well trained for new branches to make good predictions.

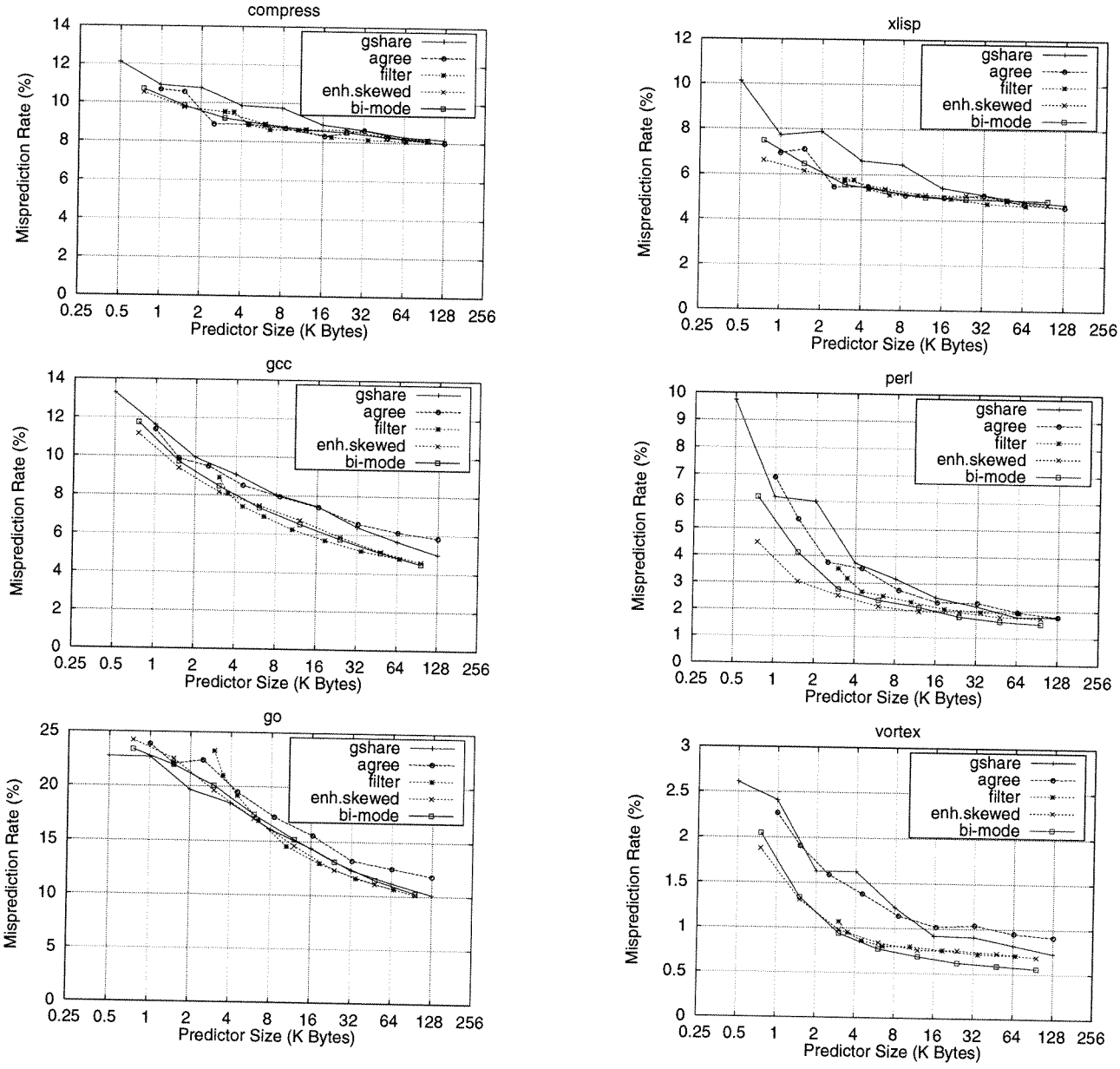


Figure 7: Comparison between gshare, agree, skewed, and bi-mode schemes — SPEC CINT95

The agree predictor does not perform as well as expected; it even performs worse than the best of gshare schemes for large budgets. This performance degradation is partly due to the capacity limitation in the buffer storing biasing bits. To explore its potential prediction accuracy, we increased the buffer size to infinite and observe that the agree predictor can deliver comparable performance to the best schemes [4]. Since an infinite size buffer is unrealistic for implementation, one possible approach is to profile branches and encode biasing bits along with the instructions during compilation. However, this compilation approach is beyond the scope of this paper and thus is not discussed.

5. Conclusion

Global history based dynamic branch predictors have been shown to achieve high prediction accuracy. Nevertheless, this type of predictors still suffer from severe interference, which restricts the accuracy. To reduce this interference and thus improve the performance, many novel variations of global predictors have been proposed, including gshare, agree, filter, skewed and bi-mode predictors. However, there is no direct comparison among these predictors.

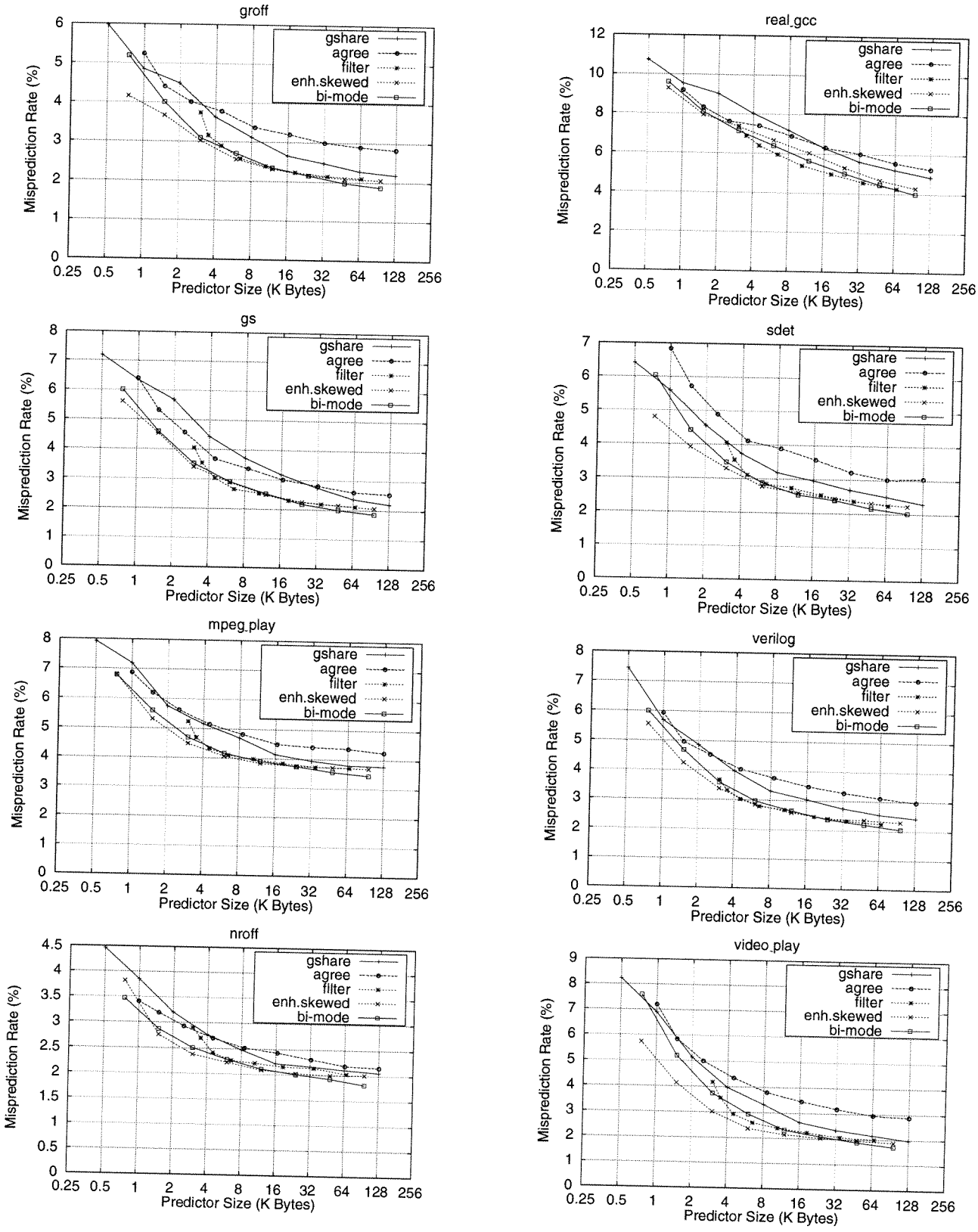


Figure 7 (continued): Comparison between gshare, agree, skewed, and bi-mode schemes — IBS-Ultrix

To provide a fair and direct comparison among these schemes, this paper examined and evaluated each scheme using the same set of benchmarks, input data set and instruction set. A cost-effective comparison is performed, in which cost is measured by the amount of storage in bytes used by each scheme.

Our simulation results show that the filter, enhanced skewed and bi-mode predictors perform the best. Comparing these three schemes, we find that the enhanced skewed predictor achieves the highest accuracy for small budgets, showing the advantage of employing complicated XOR index functions in small predictors. As the budget becomes larger, the bi-mode and filter predictors perform the best, and the bi-mode predictor eventually outperforms all other schemes for the largest budget due to the short training period it needs. Note that the bi-mode predictor is inherently easier to implement than the filter predictor, and hence the bi-mode predictor may overall be the best choice for large budgets.

6. Reference

[1] Chang, P-Y., Evers, M., and Patt, Y. Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. *Proc. of the 4th Int. Conference on Parallel Architectures and Compilation Techniques,* 1996, pp. 48-57.

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

[3] Lee, C-C., Chen, I-C., Mudge, T., The Bi-Mode Branch Predictor. *Proc. of the 30th Ann. Int. Symp. on Microarchitecture,* Dec. 1997, pp. 4-13.

[4] Lee, C-C., Optimizing High-Performance Dynamic Branch Predictors. *Ph.D Dissertation,* The University of Michigan, 1998.

[5] McFarling, S. Combining Branch Predictors. *WRL Technical Note TN-36*, June 1993.

[6] 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, pp. 292-303.

[7] SPEC CPU'95, *Technical Manual*, August 1995.

[8] 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, pp. 284-291.

[9] Uhlig, R., Nagle, D, Mudge, T., Sechrest, S., and Emer, J. Instruction Fetching: Coping with Code Bloat. *Proc. of the 22th Ann. Int. Symp. on Computer Architecture*, Italy, June 1995, pp. 345-356.

[10] Yeh, T-Y. and Patt, Y. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History. *Proc. of the 20th Ann. Int. Symp. on Computer Architecture,* May 1993, pp. 257-266.