The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference

Eric Sprangle‡† Robert S. Chappell†‡ Mitch Alsup‡ Yale N. Patt†

Advanced Computer Architecture Laboratory †
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122
{robc, patt}@eecs.umich.edu

ROSS Technology, Inc. ‡
5316 Hwy 290 West Austin, Texas 78735
{sprangle, mitch}@ross.com

Abstract

Deeply pipelined, superscalar processors require accurate branch prediction to achieve high performance. Two-level branch predictors have been shown to achieve high prediction accuracy. It has also been shown that branch interference is a major contributor to the number of branches mispredicted by two-level predictors.

This paper presents a new method to reduce the interference problem called agree prediction, which reduces the chance that two branches aliasing the same PHT entry will interfere negatively. We evaluate the performance of this scheme using full traces (both user and supervisor) of the SPECint95 benchmarks. The result is a reduction in the misprediction rate of gcc ranging from 8.62% with a 64K-entry PHT up to 33.3% with a 1K-entry PHT.

Keywords: *branch prediction, superscalar, speculative execution, two-level branch prediction.*

1 Introduction

The link between changes in branch misprediction rate and changes in performance has been well documented [1–4, 6]. Yeh and Patt have shown that a two-level branch predictor can achieve high levels of branch prediction accuracy [4]. In a two-level predictor, the first level generates an index into a Pattern History Table (PHT) using some function of the outcomes of preceding branches. The first level function was originally the value of an N-bit shift register, called the Branch History Register (BHR), that kept track of the directions of the previous N branches. The second level is an array of PHT entries (in this paper, 2-bit saturating counters) which tracks the outcomes of branches mapped to it by the first level's indexing function. Figure 1 depicts a general two-level predictor.

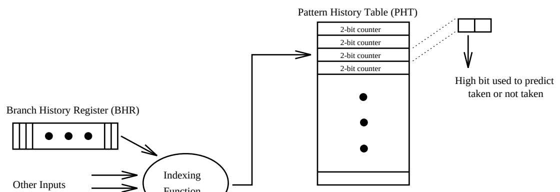
Pattern History Table (PHT)
2-bit counter
2-bit counter
2-bit counter
2-bit counter
High bit used to predict taken or not taken
Branch History Register (BHR)
Other Inputs
Indexing Function

Figure 1: A general two-level predictor.

Since the number of PHT entries is finite, it sometimes is the case that two unrelated branches in the instruction stream are mapped to the the same PHT entry by the predictor's indexing function. This phenomenon is known as PHT interference, since the outcome of one branch is interfering with the subsequent prediction of another completely unrelated branch. Figure 2 shows a case of interference in a general two-level predictor.

2 Interference in Two-Level Predictors

We define an instance of PHT interference as a branch accessing a PHT entry that was previously updated by a different branch. In most cases, this does not change the direction of the prediction. We refer to this as neutral interference, since the presence of interference had no effect on that prediction. However, it is possible for the influence of the previous branches to change the prediction of the current branch. This effect can cause a correct prediction where there would be a misprediction otherwise. This is referred to as positive interference, since the introduction of interference causes a positive effect on performance by removing a misprediction. It is also possible that interference can cause a misprediction where there would not be otherwise. We refer to this as negative interference, since it has a negative effect on performance. We will confirm that negative interference is a substantial contributor to the number of branch mispredictions [7, 8].

Given that negative interference has a substantial impact on performance, it is worthwhile to attempt to reduce it.

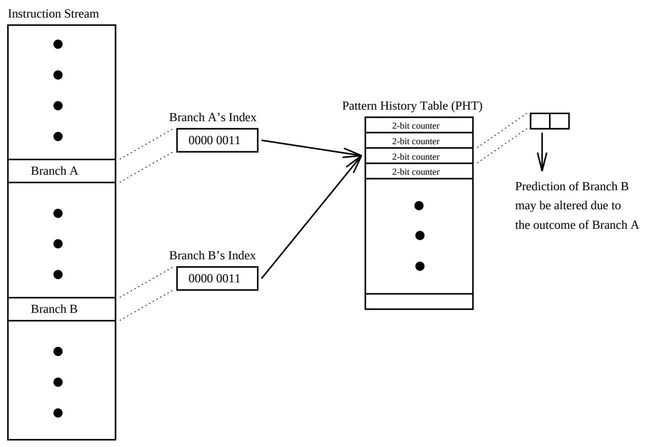

Figure 2: Interference in a two-level predictor.

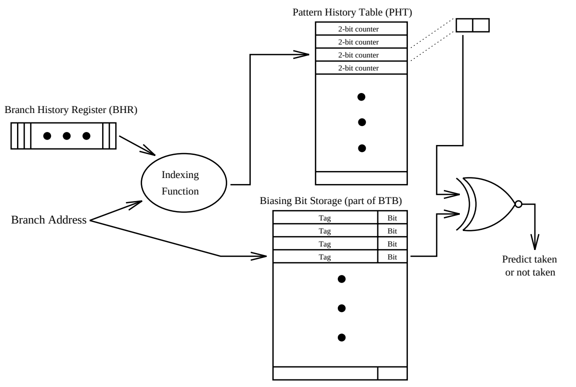

Figure 3: Agree predictor operation.

Most previous studies have examined three basic ways to counteract the interference problem:

1) Increasing predictor size, causing conflicting branches to map to different table locations.

2) Selecting a history table indexing scheme that best distributes history state among the available counters.

3) Separating different classes of branches so that they do not use the same prediction scheme, and thus cannot possibly interfere.

The first, and somewhat obvious, approach is to increase the number of entries in the PHT. This allows interfering branches to be mapped by the indexing function to different PHT entries. The fewer branches using the same PHT entry, the less interference and the less opportunity for negative interference to occur. Of course, this approach is limited by the number of transistors available to devote to the PHT.

Another means of reducing interference is to choose an indexing function that more effectively utilizes the PHT entries. For example, the gshare predictor [9] XORs the Branch History Register (BHR) with the lower bits of the branch address to generate the index into the PHT. Since BHR histories are not uniformly distributed, the introduction of address bits into the index gives a more useful distribution across all PHT entries.

Chang et al introduced *filtering*, another method of reducing PHT interference [10]. By using a simple predictor for easily predicted branches, these branches can be "filtered" out of the PHT, reducing the number of branches that are using PHT entries. This leads to a reduction in PHT interference.

3 The Agree Predictor

This paper introduces a new method of reducing negative branch interference called *agree prediction.* All of the methods discussed above focus on reducing the harmful effects of negative interference by reducing overall interference. Rather than attempting to reduce interference as a whole, the agree scheme converts instances of negative interference into either positive or neutral interference. This is done by modifying the structure and the use of the PHT.

In a traditional two-level branch predictor, each PHT entry is a 2-bit saturating counter that is used to predict whether branches using that entry are taken or not taken. The counter's state is updated using the resolved direction of the branch. If the branch was taken, the PHT entry's counter is incremented. If the branch was not taken, the counter is decremented. In this manner, future occurrences using that PHT entry are more likely to predict the correct directions.

In an agree predictor (shown in figure 3), we attach a biasing bit to each branch (for example, in the instruction cache or in a BTB) that predicts the most likely outcome of the branch. Rather than predicting the direction of a given branch, the 2-bit counters predict whether or not the branch will go in the direction indicated by the biasing bit. In effect, this scheme predicts if the branch's direction *agrees* with the biasing bit. Like a traditional predictor, the 2-bit counter is updated when the branch has been resolved. However, in the agree scheme, the counter is incremented if the branch's direction agreed with the biasing bit. The counter is decremented if the branch did not agree with the biasing bit.

This agree scheme effectively reduces negative interference. If the biasing bit is well chosen, two branches using the same PHT entry are more likely to update the counter in the same direction—towards the agree state. This fundamental change in the PHT interpretation results in negative interference being converted to positive and neutral interference, thus increasing overall prediction accuracy.

To illustrate, assume two branches have respective taken rates of 85% and 15%. In a conventional scheme, the probability that any combination of these two branches has opposite outcomes is given by:

$$(br1taken, br2nottaken) + (br1nottaken, br2taken) = \\ (85\% * 85\%) + (15\% * 15\%) = \mathbf{74.5\%}$$

In our agree predictor experiments, we set the biasing bit to the direction the branch goes the first time it is executed. This biasing bit represents the correct direction of the branch about 85% of the time, on average. Assuming that the biasing bits are optimally determined for the branches in the previous example (taken for the first branch, not taken for the second), the probability that any combination of these two branches has opposite outcomes becomes:

$$(br1agrees, br2disagrees) + (br1disagrees, br2agrees) = \\ (85\% * 15\%) + (15\% * 85\%) = \mathbf{25.5\%}$$

Therefore, the PHT entries in the agree predictor are less likely to be different for two different branches, and negative interference is decreased.

Another important side-effect of this change is the potential for increased prediction accuracy during a new branch's warm-up period. Any new branch is more likely to have it's PHT entries already in the warmed-up state—the agree state. Details of both of these effects are studied below. Finally, we should point out that the agree predictor modifies only the PHT. Any of the first level history structures can be used with this scheme. In this paper, we restrict the first level history mechanism to that of gshare.

4 Experiments

The experimental results presented in this paper were generated using SoSS, a full system simulator designed to model a Sun 4m workstation. SoSS models the CPU, MMU, memory system, disk, and other I/O devices in a Sun 4m in enough detail to allow it to boot and run SunOS 4.1.4. Users can then log into the simulated machine's console and run arbitrary user processes on it. SoSS provides mechanisms for gathering a wide variety of statistics on both the user and operating system code that are executed in a given process. These include full symbolic breakpoints, conditional counter enabling and resetting, and full disassembly of all or part of the executed code.

Without the overhead of simulating the branch predictors, the current implementation of SoSS is capable of executing at approximately 1/30th the speed of the host machine. SoSS is able to achieve this level of performance through a number of optimizations. Two of the most important sources of speed are a special fetch-decode cache (FD-cache) and host pointer entries in the simulated TLB. The FD-cache, instead of caching the instruction binary to be simulated, caches a pointer to a segment of code that simulates the instruction on the host machine. Most instructions can be simulated in less than ten host instructions. Host pointers in TLB lines allow target machine virtual addresses to be translated directly to pointers on the host machine without having to be translated to target machine physical addresses as an intermediate step.

For this paper, we ran the SPECint95 benchmarks on SoSS for 2 billion instructions, or to completion. All benchmarks were compiled to the SunOS operating system using the gcc compiler with the -O3 option. The biasing bit was set to the direction of the branch the first time it was brought into the Branch Target Buffer (BTB). Since this bit is unknown on the first prediction after a BTB fill, the agree predictor uses the sign of the branch's offset as the biasing bit. The biasing bits were recorded in a 4K–entry direct-mapped BTB.

4.1 Interference Study

As the number of PHT entries is increased, three factors affect prediction accuracy:

1) Interference decreases. This increases prediction accuracy.

2) Better branch correlation. Increasing PHT size means it is more likely for an indexing function to map only those branches with similar behavior to the same PHT entry. This also increases prediction accuracy.

3) Training time increases. Larger PHT sizes require larger indices. This typically translates to more pattern history bits being used in the address, resulting in a longer warm-up period for new branches. A longer training period reduces prediction accuracy.

These first experiments model the performance of a hypothetical interference-free predictor versus the performance of conventional and agree predictors of increasing PHT sizes. Since the interference-free predictor is not affected by the first of the above three factors, we can gain use its results to gain insight into the relative magnitudes of the other two forces. We model 1K, 2K, 4K, 8K, 16K, 32K, and 64K–PHT entry predictors.

Figures 4 to 9 show the prediction accuracy for the SPECint95 benchmarks using three versions of the gshare predictor. All three predictors use the same gshare indexing function; the lower N bits of the address are XORed with the BHR. The first version, conventional, updates the PHT entries in a conventional manner. The second version, agree, uses the agree scheme to update the PHT entries. The third version, interference-free, allocates a separate 2-bit counter for each branch/BHR index combination. All three predictors are simulated with 10-16 bit BHRs (1K to 64K–entry PHT entries, respectively).

The results show that the interference-free predictor consistently outperforms both the conventional and agree predictors, implying that negative interference dominates positive interference. In addition, the interference-free predictor gains only a slight improvement in performance as the PHT size is increased. This affirms our hypothesis that interference is the major contributor to reducing prediction accuracy.

The agree predictor clearly outperforms the conventional two-level scheme, especially in the smaller predictors and on certain benchmarks. Smaller predictors naturally have more contention for the PHT entries. The agree predictor attacks this interference and is able to increase performance. However, as the PHT size becomes very large, the agree predictor becomes less and less effective, since there is less opportunity to remove negative interference. In addition, secondary effects, such as misses in the BTB (where the biasing bits are stored) become more significant.

The agree predictor also seems to perform better on some benchmarks than on others. Benchmarks such as gcc and go have a larger branch "footprint"; they have a large number of static branches that are frequently executed. Figure 10 plots the percentage of total branches executed by each benchmark versus the number of static branches in each benchmark. We can see that some benchmarks, such as gcc and go, have much larger branch footprints than others. In general, these benchmarks are the ones that benefit the most from interference reduction. While the branch footprint is not directly representative of the amount of interference that occurs in a benchmark, it is a good first-order indicator.

Table 1 lists the improvement in prediction accuracy when moving from a 1K–entry PHT to a 64K–entry PHT, for each predictor class. All of the predictors achieve higher prediction accuracies as the number of PHT entries is increased. For the interference-free predictor, the change in performance is due solely to the combination of better correlation and greater training time. Since the plot for the interference-free predictor is monotonically increasing, we can conclude that the increase in performance due to better correlation is greater than the decrease due to increased training time.

As the predictor size is increased, both the agree and conventional predictors achieve greater improvements in accuracy than the interference-free predictor. This implies

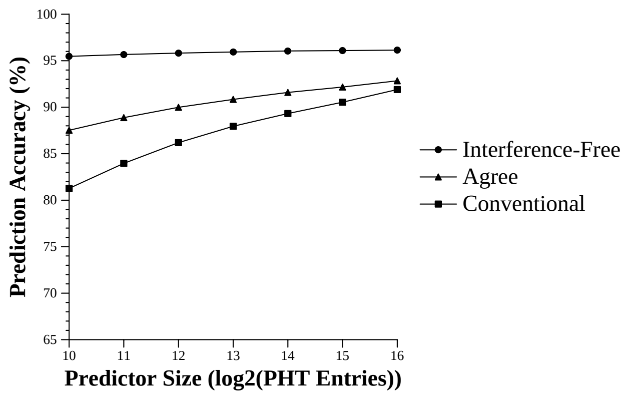

Figure 4: Prediction accuracy versus predictor size for gcc.

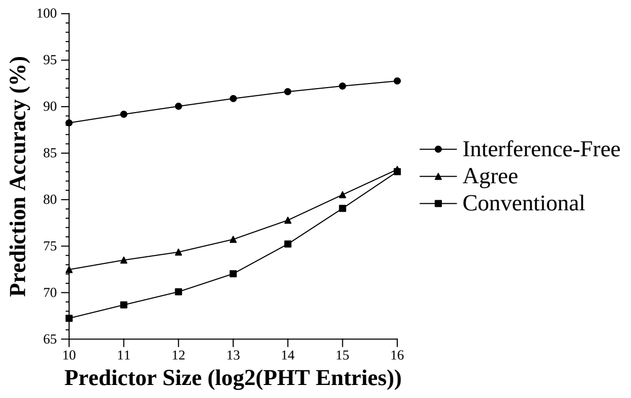

Figure 5: Prediction accuracy versus predictor size for go.

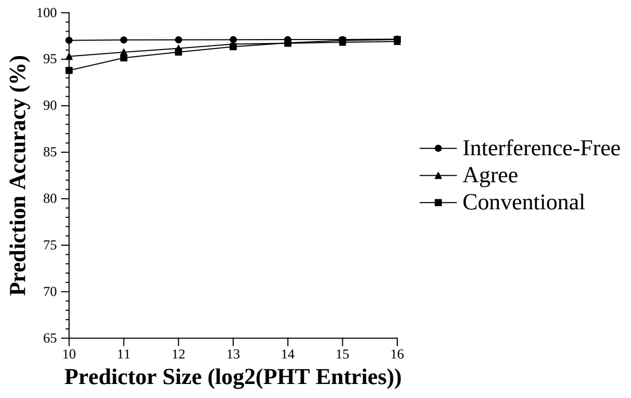

Figure 6: Prediction accuracy versus predictor size m88ksim.

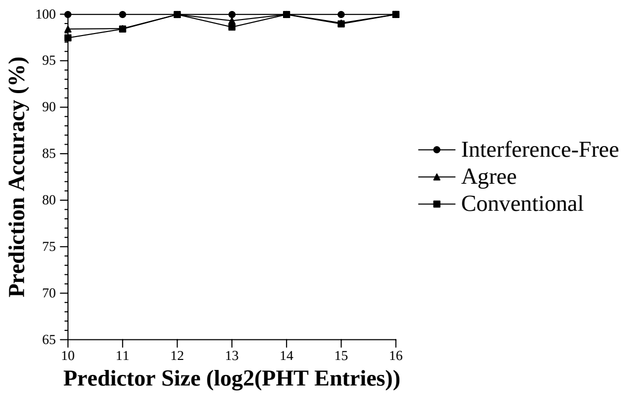

Figure 7: Prediction accuracy versus predictor size for perl.

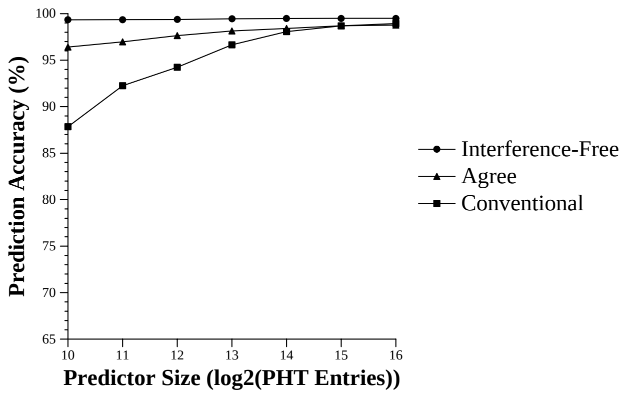

Figure 8: Prediction accuracy versus predictor size for vortex.

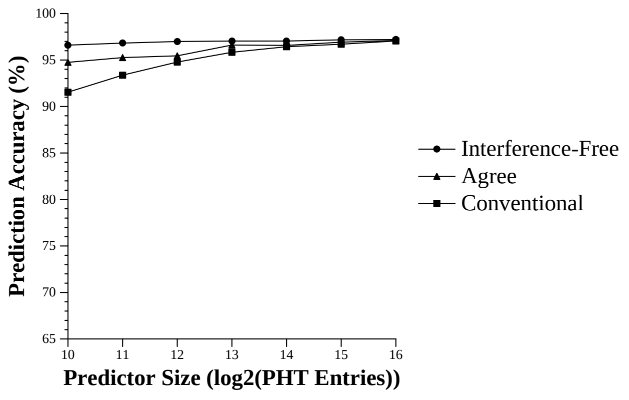

Figure 9: Prediction accuracy versus predictor size for xlisp.

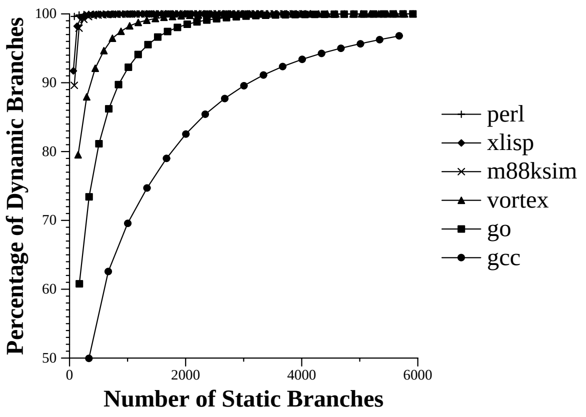

Figure 10: Percentage of static branches accounting for dynamic branches by benchmark.

Table 1: Improvement in accuracy from 1K to 64K–entry PHT.

Benchmark	Interference-Free	Agree	Conventional
gcc	.67%	5.22%	10.63%
go	4.50%	10.77%	15.77%
m88ksim	.13%	1.60%	3.32%
perl	.00%	1.57%	2.52%
vortex	.16%	2.36%	11.10%
xlisp	.60%	2.35%	5.51%

that interference is a much greater factor in predictor performance than is correlation. We can estimate the improvement in accuracy due to interference reduction (as the PHT size is increased) by examining the equation:

$$ChangeinAccuracy = InterferenceReduction + (BetterCorrelation - TrainingTimeIncrease)$$

The sum *(Better Correlation - Training Time Increase)* can be approximated by the increase in performance of interference-free predictor. By subtracting this term from the total change in PHT accuracy, we are left with the change in accuracy due to interference reduction. Table 2 lists estimates of what percentage of total improvement in accuracy is due to reduced interference as predictor size is increased from 1K to 64K–entries. These calculations imply that, as the PHT size is increased, interference reduction is responsible for most (over 71%) of the improvement in prediction accuracy.

4.2 Characterizing Positive, Neutral, Negative Interference

There are three classes of interference: positive, neutral and negative. We define an occurrence of positive interference as a branch being predicted correctly by a predictor when it would have been mispredicted by a similar predictor with no interference. Neutral interference occurs when the realistic predictor and interference-free predictor are both correct or both incorrect. Negative interference occurs if the interference-free predictor is correct, but the realistic predictor mispredicts.

Table 2: Estimated percent of improvement from reduced interference when going from a 1K– entry PHT to a 64K–entry PHT conventional scheme.

Benchmark	
gcc	93.70%
go	71.46%
m88ksim	96.08%
perl	100.0%
vortex	98.56%
xlisp	89.10%

Table 3 shows the percentage of occurrences of positive, neutral, and negative interference for conventional and agree predictors for each benchmark and predictor size.

This table confirms many of our hypotheses. In general, as the predictor size is increased, most of the interference ends up being neutral. We also see that negative interference occurs much more often than positive interference. Finally, these figures show the agree predictor effectively converts negative interference into positive and neutral interference. As with overall prediction accuracy, the reduction of negative interference lessens as the predictor size increases.

4.3 Warm-up Study

One of the problems with two-level branch prediction is the number of times a new branch instruction must execute before the predictor "warms up". This section studies the relative performance of the agree predictor versus the conventional two-level predictor for branches seen 1 to 16 times previously (we choose the first 16 branches because that is the length of the largest BHR simulated). We refer to this as the predictor's *warm-up period* for a new branch. The agree predictor achieves a higher accuracy than the conventional predictor during this period due to the increased likelihood that the PHT entry in the agree predictor will already be in a trained (agree) state. Figures 11 and 12 show the prediction accuracies of each predictor for all branches encountered exactly N times. Recall that for branches seen for the very first time (N = 1), the agree predictor is using the branch offset's sign for the biasing bit. Because of this, for some benchmarks, the conventional two-level predictor sometimes achieves higher accuracy on the first prediction for a new branch.

These results demonstrate the ability of an agree predictor to, in effect, warm up more quickly than a conventional predictor of the same history length. This is an advantage for prediction in the presence of context switches, as an agree predictor would adapt more quickly.

4.4 Choosing the Biasing Bit

In these experiments, we use the direction of the branch the first time it was executed after being placed in the BTB as the biasing bit. Once it is defined, it does not change over the course of the program, unless the branch is replaced in the BTB and later retrieved. We refer to this as the "first time" selection mechanism. It has been suggested that a better biasing bit selection mechanism would be to set the biasing bit to the direction most often taken by that branch over run of the entire benchmark. We refer to this hypothetical mechanism as "most often". As a first step toward determining an optimal biasing bit selection mechanism, we

Table 3: Percentage of total interference classified as positive, neutral, and negative.

Benchmark	Positive (conv,agree)	Neutral (conv,agree)	Negative (conv,agree)
gcc			
1K	1.15%,1.23%	83.51%,89.59%	15.35%,9.18%
2K	.99%,1.11%	86.29%,90.99%	12.71%,7.90%
4K	.91%,1.03%	88.53%,92.09%	10.56%,6.88%
8K	.83%,.98%	90.29%,92.88%	8.88%,6.14%
16K	.79%,.96%	91.60%,93.56%	7.61%,5.48%
32K	.75%,.96%	92.77%,94.03%	6.48%,5.01%
64K	.72%,.96%	93.90%,94.49%	5.37%,4.55%
go			
1K	4.15%,4.10%	70.68%,76.03%	25.17%,19.87%
2K	3.60%,3.62%	72.31%,77.09%	24.09%,19.29%
4K	3.10%,3.13%	73.85%,78.05%	23.05%,18.82%
8K	2.59%,2.65%	75.97%,79.55%	21.43%,17.79%
16K	2.11%,2.21%	79.39%,81.75%	18.50%,16.04%
32K	1.68%,1.83%	83.45%,84.61%	14.87%,13.56%
64K	1.29%,1.51%	87.63%,87.43%	11.08%,11.06%
m88ksim			
1K	.18%,.19%	96.41%,97.90%	3.40%,1.91%
2K	.10%,.08%	97.86%,98.51%	2.04%,1.40%
4K	.09%,.07%	98.51%,98.93%	1.41%,1.00%
8K	.03%,.03%	99.19%,99.47%	.78%, .50%
16K	.01%,.01%	99.60%,99.57%	.38%, .42%
32K	.02%,.02%	99.85%,99.64%	.13%, .33%
64K	.01%,.01%	99.91%,99.76%	.08%, .22%
vortex			
1K	.12%,.18%	88.25%,96.69%	11.63%,3.12%
2K	.11%,.12%	92.68%,97.38%	7.21%,2.50%
4K	.09%,.09%	94.69%,98.10%	5.22%,1.81%
8K	.07%,.08%	97.06%,98.53%	2.87%,1.39%
16K	.02%,.04%	98.54%,98.83%	1.44%,1.13%
32K	.04%,.05%	99.06%,99.06%	.91%, .88%
64K	.03%,.04%	99.27%,99.20%	.70%, .76%
xlisp			
1K	.21%,.23%	94.53%,97.70%	5.26%,2.07%
2K	.10%,.08%	96.34%,98.48%	3.55%,1.44%
4K	.13%,.15%	97.53%,98.10%	2.34%,1.75%
8K	.04%,.03%	98.67%,99.49%	1.28%, .47%
16K	.10%,.03%	99.11%,99.40%	.79%, .56%
32K	.02%,.02%	99.28%,99.61%	.70%, .37%
64K	.08%,.04%	99.54%,99.76%	.39%, .20%

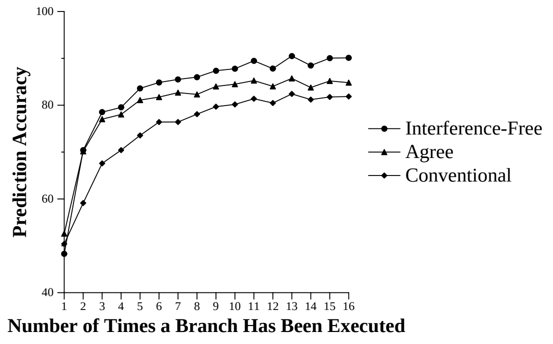

Figure 11: Prediction of branches during warm-up period with a 16-bit BHR for gcc.

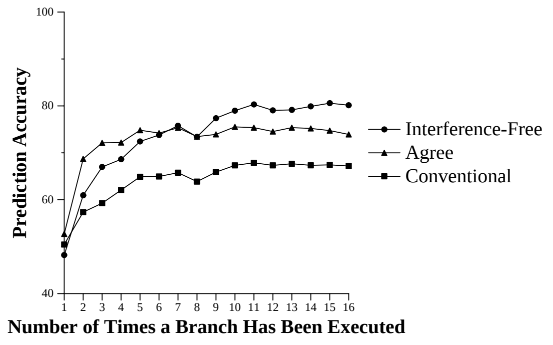

Figure 12: Prediction of branches during warm-up period with a 16-bit BHR for go.

compare the agree predictor's performance using "first time" selection to the performance using "most often" selection.

We simulate the hypothetical "most often" selection scheme by running the benchmark twice. The first run records, for each static branch in the benchmark's instruction stream, whether the branch is more often taken or not taken. This information is used to set the biasing bits for the second run. Table 4 shows the results of this experiment. Our heuristic of "first time" is comparable to the hypothetical "most often" mechanism. Future agree predictor studies will examine other dynamic schemes for selecting biasing bits.

Table 4: Comparison of prediction accuracies for "first time" versus "most often" biasing bit selection for gcc.

PHT Entries	First Time	Most Often	Difference
1K	87.52%	89.44%	1.92%
2K	88.88%	90.43%	1.55%
4K	89.99%	91.29%	1.30%
8K	90.84%	92.01%	1.17%
16K	91.59%	92.66%	1.07%
32K	92.17%	93.17%	1.00%
64K	92.84%	93.81%	.97%

5 Conclusions

This paper proposes a new method for reducing negative interference in two-level branch predictors. We have shown that an interference-free predictor performs much better than a conventional two-level predictor scheme. By redefining the operation of the PHT in a two-level predictor, we can recover some of the lost prediction accuracy due to interference. The agree scheme, while relatively simple to implement, achieves a significant improvement in branch prediction accuracy. This improvement stems from two sources. First, the agree scheme effectively converts negative PHT interference to positive and neutral interference. Second, prediction accuracy during warm-up periods is improved. The most dramatic improvements by the agree scheme over the conventional predictors can be seen in benchmarks with large branch footprints and in predictors with small PHT sizes. The agree predictor reduces misprediction rates in gcc ranging from 8.62% with a 64K–entry PHT up to 33.3% with a 1K–entry PHT.

6 Acknowledgments

This work is the result of collaborative research performed jointly by researchers at the University of Michigan and at Ross Technology and was funded in part by a gift from Ross Technology. We gratefully acknowledge their support. We also acknowledge the contributions of Sanjay Patel, Jared Stark, and Daniel Friendly, PhD students at Michigan who have been very helpful to our work.

References

[1] B. Calder and D. Grunwald, "Fast and Accurate Instruction Fetch and Branch Prediction", Proceedings of the 21st International Symposium on Computer Architecture, (April 1994), pp. 2-11.

[2] S. McFarling and J. Hennessy, "Reducing the Cost of Branches", Proceedings of the 13th International Symposium on Computer Architecture, (1986), pp. 396-403.

[3] J. A. Fisher and S. M. Freudenberfer, "Predicting Conditional Branch Directions from Previous Runs of a Program", Proceedings 5th Annual International Conference on Architectural Support for Programming Languages and Operating Systems, (October 1992).

[4] T-Y. Yeh and Y. N. Patt, "Two-Level Adaptive Training Branch Prediction", 24th ACM/IEEE International Symposium on Microarchitecture, (November 1991), pp. 51-61.

[5] T-Y. Yeh and Y. N. Patt, "A Comprehensive Instruction Fetch Mechanism for Processor Supporting Speculative Execution", 25th ACM/IEEE International Symposium on Microarchitecture, (December 1992), pp. 129-139.

[6] B. Calder, D. Grunwald, and J. Emer, "A System Level Perspective on Branch Architecture Performance," 28th ACM/IEEE International Symposium on Microarchitecture, (November 1995).

[7] A. R. Talcott, M. Nemirovsky, and R. C. Wood. “The Influence of Branch Prediction Table Interference on Branch Prediction Scheme Performance”, International Conference on Parallel Architectures and Compilation Techniques, (1995).

[8] C. Young, N. Gloy, and M. D. Smith, “A Comparative Analysis of Schemes for Correlated Branch Prediction”, Proceedings of the 22nd Annual International Symposium on Computer Architecture, (1995), pp. 276-286.

[9] S. McFarling, “Combining Branch Predictors”, Technical Report TN-36, Digital Western Research Laboratory, (June 1993).

[10] P-Y. Chang, M. Evers, and Y.N. Patt, “Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference”, International Conference on Parallel Architectures and Compilation Techniques, (October 1996).