The Role of Adaptivity in Two-Level Adaptive Branch Prediction

Stuart Sechrest, Chih-Chieh Lee, and Trevor Mudge
EECS Department, University of Michigan
1301 Beal Ave., Ann Arbor, Michigan 48109-2122
{sechrest, leecc, tnm}@eecs.umich.edu

Abstract

Seeking high branch–prediction accuracy, architects are making use of the extended history of individual branches. One approach is to divide the branch prediction task into two levels: the first records the results of previous branches; the second makes predictions based upon previous instances in which particular patterns arose. PAg predictors use simple state machines in the second level to provide adaptive predictions.

We show that this adaptive level benefits from a high level of hysteresis. We further show that, if the predictions for this second level are fixed rather than adaptive, i.e. a PSg organization, the performance can be superior to that of PAg predictors for short branch histories and close to PAg performance for longer predictors. The patterns of errors among these schemes provide insight into the workings of a wide variety of two-level schemes.

Key words: *dynamic branch prediction, two-level branch prediction, PAg, PSg.*

1 Introduction

Yeh and Patt [YehPatt91, YehPatt93] demonstrated that a two-level branch predictor, organized as shown in Figure 1, can achieve high accuracy. The predictor stores the outcomes of each branch for the most recent n executions in a Branch History Table (BHT). The recent history of a branch therefore serves to place the succeeding instance of the branch in one of 2^n classes. The first level, therefore, serves to divide the branch instances in the program execution stream into 2^n substreams, mixing together instances of separate static instructions, on the assumption that branches having similar histories will have similar behaviors. At the second level, a set of Moore machines, each associated with a single substream, are used to generate predictions for the substreams produced by the first level. These states of these machines, and hence their predictions, are determined by the results of previous branch instances in the associated stream. Yeh [Yeh93] investigated several alternative predictors requiring one or two bits of storage for this second-level. He found that a two-bit saturating counter to be the best predictor for a substream.

While most work on two-level branch prediction has focused on adaptive schemes, such as the scheme just described, early work by Lee and Smith [Lee84] showed that a scheme using fixed prediction tables based upon training runs or the program under consideration could achieve high prediction accuracy as well. Yeh and Patt [YehPatt93] refer to these adaptive and fixed schemes as PAg and PSg, respectively. As Figure 2 shows, a PSg predictor is essentially a PAg predictor from which the ability to adapt has been removed. Both schemes use an n-bit branch history table to dynamically divide the program execution stream into 2^n substreams. However, while a PAg scheme uses a Moore machine to adapt its prediction, a PSg scheme commits itself to a single fixed prediction for each substream.

In this study we examine the role of adaptation in two-level branch prediction using per-address branch history tables. We show that adding hysteresis to the second level adaptive predictors improves performance; a PSg scheme with no ability to adapt can sometimes perform better than a PAg scheme using two-bit saturating counters; while the ability to adapt can be beneficial, the magnitude of this benefit can be overstated by the commonly used SPECint92 benchmarks.

In Section 2 of this paper we discuss the benchmark programs used in this study. Section 3 presents results of simulations of several PAg and PSg schemes. Section 4 analyzes the interaction between benchmarks and prediction schemes. Section 5 contains our conclusions.

2 The Benchmarks

We conducted this study using the six integer programs from the SPEC92 benchmark suite [SPEC92] and eight IBS-Ultrix benchmarks [Uhlig95]. The SPECint92 programs were compiled for a MIPS R2000-based workstation and traced while executing their largest inputs. The resulting traces include only code executed at the user-level. The IBS-Ultrix benchmarks are a set of traces of applications running under Ultrix 3.1, collected through the hardware monitoring of a MIPS R2000-based workstation. These traces include not only the user-level instructions of the application, but kernel-level instructions and instructions executed by auxiliary processes such as the X server.

Table 1 shows some important statistics regarding the benchmark traces. The number of conditional branches contained within the traces varies from nearly five million to more than three hundred million, and constitutes anywhere from a tenth to a quarter of the instructions executed within a given trace. The benchmarks vary widely in the number of distinct branch instructions that they exercise. While all of the IBS-Ultrix benchmarks contain thousands of distinct conditional branches, some of the SPECint92 benchmarks contain only hundreds. Even more striking is the fact that while the executions of the IBS-Ultrix benchmarks and of the SPECint92 `gcc` benchmark tend to exercise a large number of branches, the other SPECint92 benchmarks are dominated by a small population of branches. In `compress`, `eqntott` and `xlisp`, in particular, the vast majority of dynamic branches are attributable to only a handful of instructions.

This work was supported by Advanced Research Projects Agency under ARPA/ARO Contract Number DAAH 04-94-G-0327.

1072-4451/95 $4.00 © 1995 IEEE
Proceedings of MICRO-28

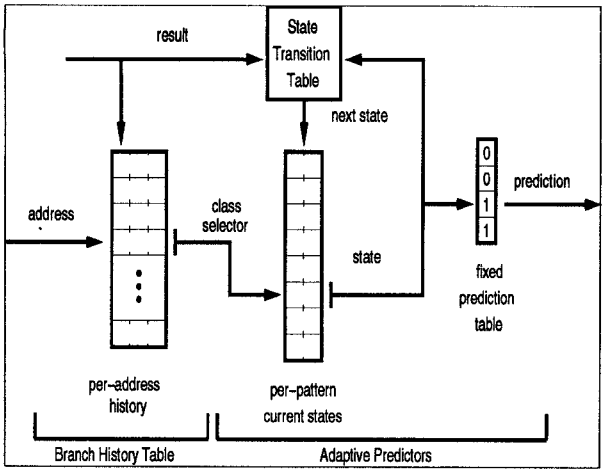


Figure 1: PAg organization

A typical PAg dynamic branch predictor organization. The front part is a Branch History Table (BHT), which stores the results of previous executions of a branch. In this paper we assume that the size of the BHT is unbounded. The back part is a group of two-bit Moore machines, each associated with a particular history pattern. The current branch's history selects a Moore machine, whose state, through a fixed mapping, determines the prediction. The actual branch result is used to update both the BHT entry and the state of the selected Moore machine.

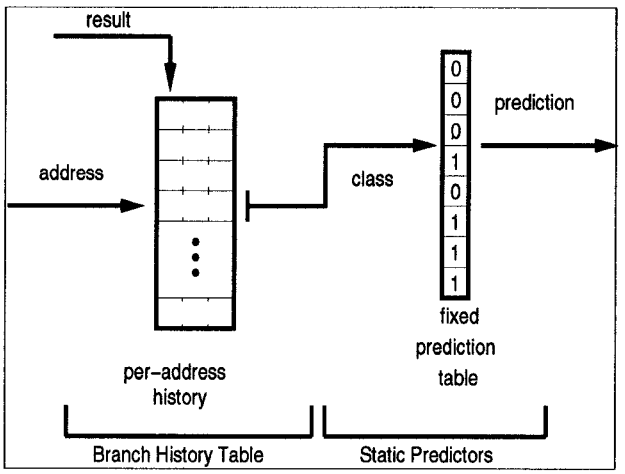


Figure 2: PSg organization

A PSg predictor is a dynamic predictor. Like a PAg predictor, a PSg predictor records the branch history in a BHT and makes predictions on the basis of the current branch's recent history. Rather than using the actual branch results to alter the prediction for each pattern, a PSg predictor relies on a fixed table.

Benchmarks	Dynamic Instructions	Dynamic Conditional Branches (Percent of Total Instructions)	Static Conditional Branches	# of Static Branches Constituting 90% of Total Dynamic Conditional Branches
compress	83947354	11739532 (14.0%)	236	13
eqntott	1395165044	342595193 (24.6%)	494	5
espresso	521130798	76466489 (14.7%)	1784	110
gcc	142359130	21579307 (15.2%)	9531	2020
xlisp	1307000716	147425333 (11.3%)	489	48
sc	889057008	150381340 (16.9%)	1269	157
groff	104943750	11874183 (11.3%)	6325	461
gs	118090975	16275133 (13.8%)	12768	1142
mpeg_play	99430055	9549954 (9.6%)	5592	532
nroff	130249374	22542119 (17.3%)	5243	229
real_gcc	107374368	14281721 (13.3%)	17354	3196
sdet	42051812	4965098 (11.8%)	5309	532
verilog	47055243	6195248 (13.2%)	4631	845
video_play	52508059	5579331 (10.6%)	4603	724

Table 1: Characterization of the SPECint92 and IBS-Ultrix benchmarks

All workloads were compiled with the Ultrix MIPS C compiler version 2.1, using the -O2 optimization flag. The IBS-Ultrix traces were gathered through hardware monitoring of applications running on a MIPS R2000-based workstation running Ultrix 3.1. The traces include both user- and kernel-mode references. The SPECint92 traces were gathered using pixie. They contain only user-mode references. The largest input files supplied were used.

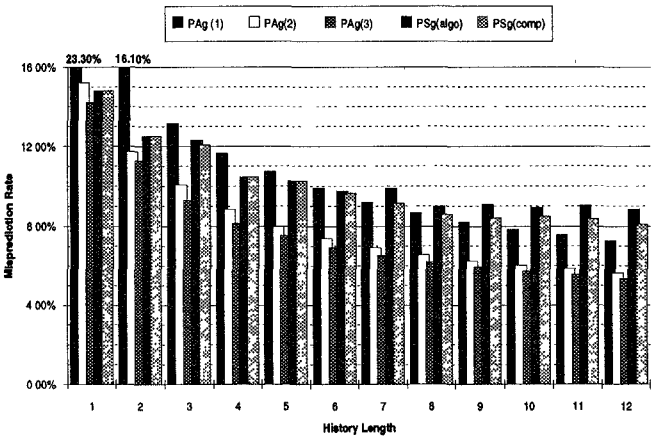


Figure 3: Misprediction rates for the SPECint92 benchmarks (Left)

This figure displays the misprediction rates for the SPECint92 benchmarks as a function of the BHT's history length. All bars represent the average of the misprediction rates for the individual programs. We show results for PAg schemes with one-, two-, and three-bit saturating counters and for PSg schemes with two fixed tables, one derived through the simple algorithm presented, the other derived from measurements of a number of actual programs. The results show the benefits of added hysteresis for PAg schemes. The PSg schemes performed worse than the PAg schemes for all of the benchmarks, but particularly badly for `eqntott`.

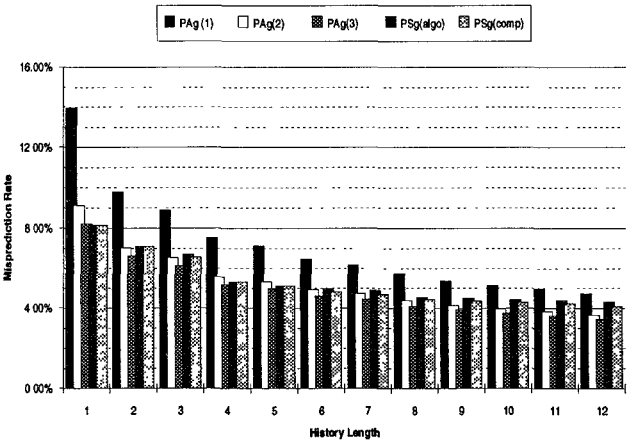


Figure 4: Misprediction rates for the IBS-Ultrix benchmarks (Right)

This figure shows the misprediction rates for the IBS-Ultrix benchmarks. Each bar represents the average of the misprediction rates for the eight individual programs. The outcomes of approximately 1% of the branches could not be determined due to interrupts that occurred between the execution of the branch and the execution of the branch target or the subsequent instruction. To bracket the range of possible misprediction rates, simulations were performed with both the assumption that all of these branches were taken and that all were not taken. The effect on the magnitude of the misprediction rate estimated is pronounced for very short histories, but negligible for longer histories. Since the interrupts are randomly distributed with respect to the branches, they do not affect the relative merit of the algorithms. The figures here are constructed with the assumption that the branches are taken.

As with the SPECint92 benchmarks, the PAg schemes improve with added hysteresis. For these programs the PSg schemes perform slightly worse than PAg schemes for longer histories. For shorter histories, the PSg schemes actually perform better than a PAg scheme with the usual two-bit counter.

3 Experiments

We simulated two groups of predictors for branch histories varying from 1 to 12 bits in length. For this study we assume an unbounded BHT. The first group comprised three PAg predictors, using one-, two- and three-bit saturating counters at the second level to maintain the state of a substream. We refer to these, respectively, as the PAg(1), PAg(2) and PAg(3) schemes. The second group comprised two PSg predictors. For each predictor, we determined a table of 2^n predictions for use with an n-bit BHT. For one predictor we used a simple algorithm to derive the prediction tables. We refer to this as the PSg(algo) scheme. For the other predictor we used the results of runs of a large number of programs to produce a compromise prediction table. We refer to this as the PSg(comp) scheme. Neither scheme uses tables specifically designed for a single program; for both schemes, the same set of tables was used for all benchmarks.

The central hypothesis used in constructing a table for the PSg(algo) scheme is that program behavior is dominated by repeated sequences of actions, causing the behavior of individual branches to display short repeated patterns. We construct a table for n-bit history patterns by first identifying all of the patterns that can be produced using cycles of at most n/2 bits. For each of these patterns we predict that the next branch will continue the pattern. For example, the ten-bit pattern 1010101010[1] is interpreted as deriving from the two-bit pattern 10, while 1101101101 is interpreted as deriving from the pattern 101. In both cases, 0 is predicted for the next instance.

Repeating patterns such as these, however, account for only 52 of the 1024 patterns possible with a 10-bit history. What ought one do for the remaining patterns? One possibility is that these patterns represent a transition between two modes of cyclic behavior. Accordingly, we ignore the two oldest bits and make predictions on the basis of repeated patterns in the remaining bits. Thus one would interpret the pattern 1010101011 as four cycles of a two-bit pattern, preceded by data that can be ignored. Consequently the same prediction is made for all four of the cases covered by the pattern 10101010xx, where x represents a "don't care" bit (provided that no prediction has been made using more bits). This process continues, substituting two additional don't care bits with each iteration.

A number of patterns will remain after this process, all of which differ in their two most significant bits. The pattern 1000000000 is an example. For these patterns we choose to regard each history bit as a separate sample and select the prediction 0 if zeroes predominate in the pattern and 1 if ones predominate (ignoring the oldest bit if the number of bits is even).

1. The most significant bit, the left-most bit, is the most recent branch result

The compromise tables were derived by measuring the outcomes of branches with particular histories in programs drawn from the IBS-Mach benchmark suite. These programs consist of the same benchmark applications as IBS-Ultrix, but with the Mach 3.0 operating system providing system services. For each program, the most-likely outcome was noted for each history pattern. The outcome that had proved most likely in the largest number of programs was chosen for the compromise table. Although the user-level code for the IBS-Mach programs matches that for the IBS-Ultrix programs, neither the behavior of any single program, nor the behavior of the shared Ultrix code could determine the content of the compromise tables. Thus, the compromise tables cannot be regarded as tuned to the IBS-Ultrix benchmarks.

Figures 3 and 4 show the results of the simulations of the PAg and PSg schemes for, respectively, the SPECint92 and the IBS-Ultrix benchmarks.

For all of the benchmarks and for all history lengths, the pattern among the three PAg schemes was the same. In every case, the change from a single bit to a two-bit saturating counter provided a decrease in the misprediction rate in excess of one percent. In every case, the change from a two-bit to a three-bit counter provided at least a small additional improvement. Further simulations of longer counters have shown that additional bits do not provide any appreciable improvement over three bits, but neither do they hurt performance.

For single-level branch prediction schemes that associate a Moore machine with a particular static branch, rather than with a particular pattern of branch behavior, it is important to strike a balance between adaptivity and hysteresis. Generally, two-bit saturating counters have appeared to perform better in this role than have three-bit counters [Smith 81]. For two-level predictors, however, the division of the execution streams by branch history results in substreams with more consistent behavior than division by branch address. Consequently, the need for adaptivity is reduced, and greater hysteresis rewarded.

For the SPECint92 benchmarks, the two PSg schemes fare poorly. Across the range of history lengths, the fixed prediction tables make numerous errors. The `eqntott` program proves to be particularly troublesome. For 4-bit histories, for example, the algorithmic table and the compromise table, which are identical, prove to make the wrong choice for seven of the sixteen patterns. For the `gcc` benchmark, on the other hand, the algorithmic and compromise tables make the right choice for all sixteen patterns and makes fewer errors than does PAg(2) (though more than does PAg(3)). For longer histories, the PSg schemes do slightly worse for `gcc` than do either PAg(2) or PAg(3).

For the IBS-Ultrix benchmarks, the performance of the PSg schemes proves to be quite similar to the behavior just described for `gcc`. For shorter histories, the performance of the two PSg schemes tends to lie between that for PAg(2) and PAg(3), with PSg(comp) providing a slight improvement over PSg(algo). The PSg schemes, however, are unable to take advantage of longer histories, while the adaptive schemes are, with the result that the PAg(2) and PAg(3) schemes show better performance for history lengths of nine bits or more.

4 Analysis

The difference between the results for the SPECint92 benchmarks and the IBS-Ultrix benchmarks illustrates that the choice of benchmark programs can strongly influence conclusions at which one arrives. The two benchmarks suites differ markedly in two respects. First, as Table 1 showed, aside from `gcc`, the SPECint92 benchmarks are dominated by a much smaller number of branches than are the IBS-Ultrix benchmarks. Second, as Figure 5 shows,

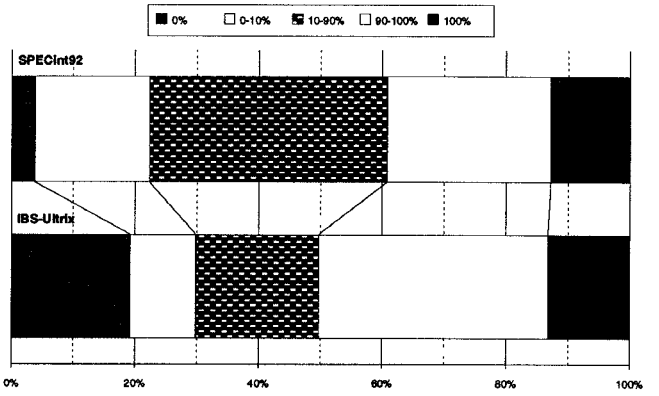


Figure 5: Classification of dynamic branches by probability taken

For each benchmark we first calculated the proportion of the time that each static branch was taken. We then determined the proportion of the dynamic branches that were to static branches that fell into each of five ranges. This figure shows the average of the proportions for the programs within each of the two benchmark suites. We see that for the IBS-Ultrix benchmarks, a much higher proportion of the dynamic branches fall into the more easily predicted categories. For example, never-taken branches, such as error checks, account for nearly 20% of the branches executed by the IBS-Ultrix programs. Of the SPECint92 benchmarks, only `sc` and `gcc` had any significant number of branches of this type.

the IBS-Ultrix benchmarks contain a higher proportion of relatively easy to predict branches.

The difference in the number of heavily exercised branches is significant because the first-level of both the PAg and the PSg schemes divides the execution streams into substreams based upon the recent history of the individual branches. Instances of separate branches that happen to have the same history will be combined in a substream. This mixing of branches in a substream will not occur if only a few branches are active. A single branch may have a strong characteristic behavior; for example a branch might be taken every third execution. For this particular branch, the four-bit history 0010 indicates that the branch is about to be taken. For most branches, however, this pattern is more likely to indicate that the branch will not be taken. If all the instances in the 0010 substream arise from the branch with the cycle-of-three behavior, an adaptive second level will quickly begin predicting 1. However, in a mixed stream in which instances of the cycle-of-three branch do not predominate, the second level will continue to predict 0, resulting in misprediction rates similar to the those for a PSg scheme. Moreover, if a burst of instances from this branch did cause a change in prediction, the subsequent instances for other branches are likely to be mispredicted. As the length of the branch histories used to divide the execution stream grows, however, the likelihood of that instances of branches with dissimilar behaviors are mixed in a substream diminishes. This accounts for the continued improvement in the PAg schemes' prediction accuracy for longer histories.

The difference between the benchmark suites in the proportion of branches that are either very frequently or very infrequently taken tends to make the SPECint92 benchmarks harder to predict. Indeed, for all of the schemes, the misprediction rates for the SPECint92 benchmarks are higher. The difference in proportions arise from the nature of the benchmarks. The IBS-Ultrix traces represent complete runs of significant programs. The code

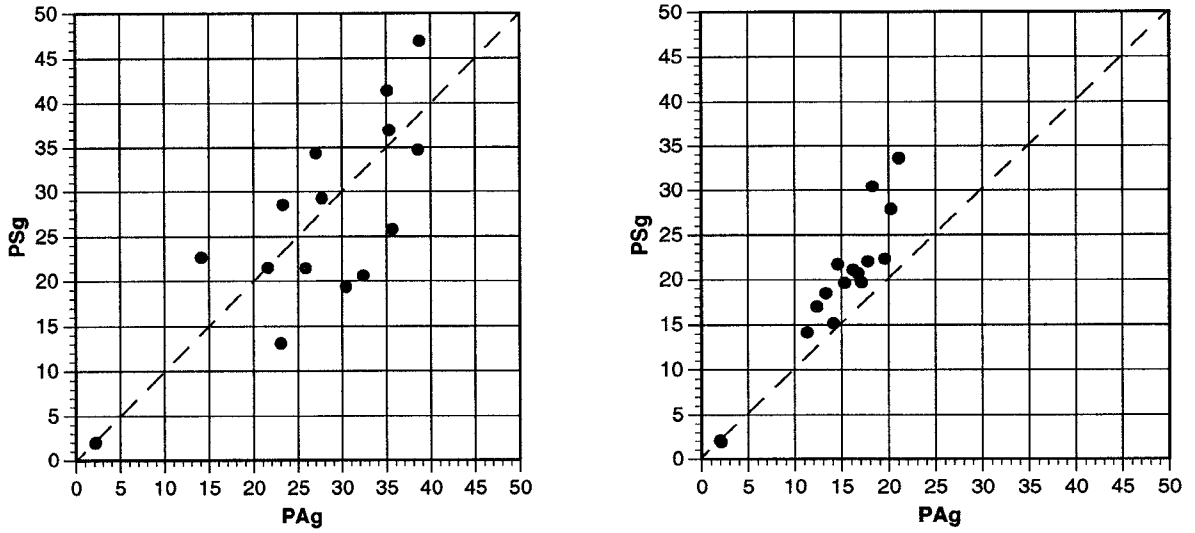


Figure 6: Misprediction rates for IBS-Ultrix using 4- and 10-bit histories

The scatter plot on the left shows the misprediction rates for each of the sixteen possible patterns for a four-bit history. The x and y coordinates of a point are determined by averaging, respectively, a pattern's PAg(2) and PSg(algo) misprediction rates across all of the IBS-Ultrix benchmarks. Thus, points lying close to the diagonal line represent patterns for which the predictive powers of the two schemes are nearly equivalent. Points lying above this line represent patterns for which the PAg scheme was superior, and points below the line patterns for which the PSg scheme was superior. The points representing the patterns 0000 and 1111 more or less coincide near (2,2), so only 15 points are visible.

The scatter plot on the right shows the misprediction rates using a ten-bit history. We coalesce the 10-bit patterns sharing the same four most recent bits into single points. As before, the points representing the patterns 0000xxxxxx and 1111xxxxxx more or less coincide near (2,2). For the other prefixes, the additional six bits of history increase prediction accuracy over the accuracy available using only four bits, with the PAg scheme showing greater improvement than the PSg scheme.

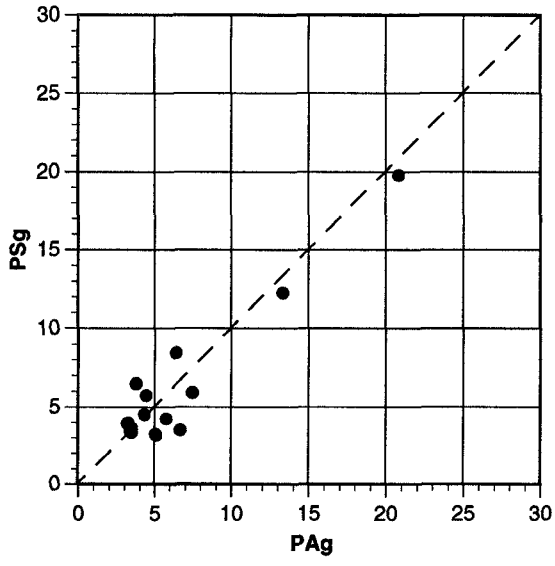
includes numerous error checks that can result in branches, both forwards and backwards, that are rarely, or even never, taken. The SPECint92 benchmarks, on the other hand, perform limited tasks large numbers of times. It has been shown elsewhere that the SPECint92 benchmarks can be misleading for the design of instruction caches[Gee93, Uhlig95]. The degree to which the misprediction rates for SPECint92 benchmarks are dependent upon a given scheme's success in handling branches with varied histories may be similarly misleading, since this dependence may exaggerate the differences between schemes.

While Figure 4 shows the overall difference in misprediction rates between the PAg and PSg schemes, Figure 6 shows a more detailed breakdown of this rate for the PAg(2) and the PSg(algo) schemes. We see that both schemes are highly accurate in their predictions for branch instances that follow a sequence of four consecutive identical decisions. The misprediction rates, however, are not nearly so low for more varied patterns. Adding additional history bits improves the misprediction rates for all of the varied patterns. For reasons we have argued above, however, the adaptive schemes are able to take somewhat better advantage of additional bits.

Figure 7 shows that the contributions of the different patterns to the overall misprediction rate are highly unequal. Although the misprediction rates for instances following several consecutive 0 or 1 decisions are low, the incidence of these patterns is so high that they prove to be the largest contributors to the total errors. Moreover, attempting to adapt the prediction for the patterns 0000 and 1111 is actually harmful. Although the effect on the misprediction rate for the pattern is slight, the high incidence of these patterns and the high degree of mixing of the substream mean that any adjustment to the prediction is likely to be counterproductive. This difficulty accounts for the superiority of the PSg schemes over PAg(2) for short histories. The added hysteresis of a three-bit counter counteracts this difficulty, resulting in better performance for the PAg(3) scheme. The other patterns are far less frequent, but their high misprediction rates cause them to contribute significant numbers of errors. The greater improvements in accuracy for these patterns accounts for the superiority of the adaptive scheme for longer histories.

5 Conclusions

The role of adaptivity at the second level of two-level branch prediction schemes is more limited than has been thought. We have shown that prediction for substreams based upon branch history benefit from greater hysteresis than does prediction for substreams based solely upon branch address. Consequently, three-bit counters prove to be superior to two-bit counters at this level. Further, we have shown that the predictions for this level may even be made by a fixed table, constructed following the observed behavior of branches in a range of programs, but constructed without reference to the particular program to be run. If the program branches follow "usual" behavior, or if many branches are exercised frequently, the prediction rates using this fixed table will be close to those of a conventional PAg(2) scheme and may actually be superior for shorter histories. PSg schemes can suffer poor performance in programs in which a small number of branches dominate the execution, but a reliance on the SPECint92 benchmarks tends to overstate the dangers of this happening.



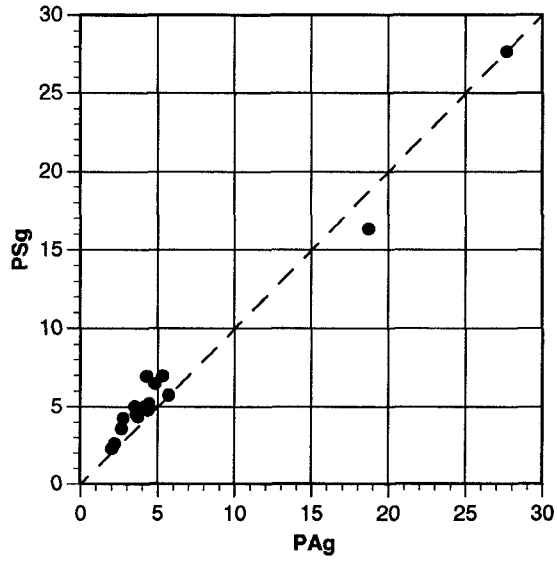


Figure 7: Distribution of errors for IBS-Ultrix using 4- and 10-bit histories

The scatter plot on the left shows the relative number of errors due to each of the 16 possible four –bit patterns. The number of errors is normalized for each benchmark, with the total number of errors made by the PAg(2) scheme on a given benchmark set to 100. The normalized numbers were then averaged across the benchmarks. We see that two patterns account for more than 30% of the errors for both the PAg and PSg schemes. These are the patterns 0000 and 1111. Despite the low misprediction rate for these patterns, their high frequency make them the most significant contributors to the total errors. We see that the PSg scheme makes somewhat fewer errors for these two patterns than does the PAg scheme.

The scatter plot on the right shows the relative number of errors using 10-bit histories. As before, we coalesce patterns with the same four most recent bits. With 10-bit histories, errors due to the patterns 0000xxxxxx and 1111xxxxxx constitute an even higher proportion of the total errors, since additional bits improves the accuracy of patterns with other prefixes, while leaving the accuracy for these patterns essentially unchanged. Again the PSg scheme does as well or better for these patterns. For the patterns with other prefixes, the edge in prediction accuracy for the PAg scheme results in a smaller number of errors.

The design of a branch prediction architecture for a processor requires attention to a great variety of factors involving tradeoffs between performance and implementation cost. We do not make the claim that any of the mechanisms discussed belongs in an actual design. The comparison and analysis, however, does provide insight into the functioning of a variety of two-level mechanisms, and points to ways in which these mechanisms may be improved, and to their ultimate limitations.

References

[Gee93]Gee, J., Hill, M., Pnevmatikatos, D. and Smith, A.J. *Cache Performance of the SPEC92 Benchmark Suite*, IEEE Micro (August): 17-27, 1993.

[Lee84]Lee, J.K.F. and Smith, A.J. *Branch Prediction Strategies and Branch Target Buffer Design*, IEEE Computer, 21(7):6-22, Jan. 1984.

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

[SPEC92]SPEC CINT92, Release V1.1, Dec. 1992.

[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.

[Yeh93]Yeh, T-Y. *Two-Level Adaptive Branch Prediction and Instruction Fetch Mechanisms for High Performance Superscalar Processors*, Ph.D Thesis, The University of Michigan, 1993.

[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.

[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, 257-266, May 1993.