# Correlation and Aliasing in Dynamic Branch Predictors

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

*Previous branch prediction studies have relied primarily upon the SPECint89 and SPECint92 benchmarks for evaluation. Most of these benchmarks exercise a very small amount of code. As a consequence, the resources required by these schemes for accurate predictions of larger programs have not been clear. Moreover, many of these studies have simulated a very limited number of configurations. Here we report on simulations of a variety of branch prediction schemes using a set of relatively large benchmark programs that we believe to be more representative of likely system workloads. We have examined the sensitivity of these prediction schemes to variation in workload, in resources, and in design and configuration. We show that for predictors with small available resources, aliasing between distinct branches can have the dominant influence on prediction accuracy. For global history based schemes, such as GAs and gshare, aliasing in the predictor table can eliminate any advantage gained through inter branch correlation. For self-history based prediction scheme, such as PAs, it is aliasing in the buffer recording branch history, rather than the predictor table, that poses problems. Past studies have sometimes confused these effects and allocated resources incorrectly.*

## 1 Introduction

The importance of accurate branch prediction to future processors has been widely noted. The correct prediction of conditional branch outcomes allows pipeline bubbles and the attendant losses in performance to be avoided. On deeply pipelined processors and processors seeking to obtain considerable instruction-level parallelism the effect on performance can be substantial. In the early 1980's it was shown that for many programs, a set of two-bit saturating counters, indexed by branch address, could give good predictions [Smith81, Lee84]. In recent years, as the number of transistors available for implementing more elaborate predictors has increased, interest in the design of branch prediction hardware has intensified. Many of the recent designs have sought to exploit information from multiple distinct branches in the prediction of each new branch instance. These have been termed *correlating branch predictors* or *two-level branch predictors*.

While a burst of activity has brought forth many new ideas and designs, much of the work proposing and evaluating new branch predictor designs has explored only a small number of sizes and configurations. As a consequence, it is difficult to assess the significance of many of the performance differences reported. Moreover, the set of benchmark programs typically used, the SPECint92 benchmark suite, is weighted toward very small programs. The program size can, in fact, have a strong influence on the performance of many two-level schemes, but this influence is rarely considered directly.

As branch predictors based upon these ideas find their way into products [MicroReport95a, MicroReport95b], it is important that architects understand the likely behavior of their designs for real applications. In this study we survey a variety of branch prediction schemes, showing their sensitivity to variations in resources allocated, design, and workload. Together, these results clarify the relationship between several existing schemes, presenting a "big picture" view of the design space for two-level branch predictors. This big picture helps to put into context some of the results that have been reported in recent studies that have included only a small number of design variations.

We have found that for programs executing large numbers of branches, the number of branches exercised has greater influence on overall prediction rates than the semantics of the program or of individual branches. This is not the case with the SPECint92 benchmarks, in which a small number of branches are exercised an enormous number of times. In these programs, the handling of a small set of cases can result in tremendous changes in prediction rates. For large programs, performance is dependent primarily upon handling the most frequent cases well. A consequence is that branch predictor designs that begin by reasoning from individual program constructs may handle a particular case better, but may still make no appreciable difference to the overall prediction rate. Designs emerging from measurements of large programs are likely to allocate resources more effectively and to improve overall system performance to a greater degree.

## 2 Benchmarks

We conducted this study through simulation driven by traces gathered for the six SPECint92 benchmarks [SPEC92] and the eight IBS-Ultrix benchmarks[Uhlig95] (see Table 1). The SPECint92 programs were compiled for a MIPS R2000-based workstation with the Ultrix MIPS C compiler version 2.1, using the -O2 optimization flag and were 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 hardware monitoring of a MIPS R2000-based workstation. These traces include both instructions executed at the user level and at the kernel level, as well as instructions executed by auxiliary processes such as the X server.

Considered statically, the SPECint92 benchmarks seem like a reasonable set of programs with which to study branch prediction. All contain a fairly large number of conditional branch instructions, with three containing over a thousand. When the dynamic frequency of these branches is considered, however, potential problems with the benchmarks appear. In five out of the six bench-

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

ISCA '96 5/96 PA, USA
© 1996 ACM 0-89791-786-3/96/0005...$3.50

| Benchmarks | Dynamic Instructions | Dynamic Conditional Branches (Percent of Total Instructions) | Static Conditional Branches | # of Static Branches Constituting 90% of Total Dynamic Conditional Branches |
|---|---|---|---|---|
| compress | 83,947,354 | 11,739,532 (14.0%) | 236 | 13 |
| eqntott | 1,395,165,044 | 342,595,193 (24.6%) | 494 | 5 |
| espresso | 521,130,798 | 76,466,489 (14.7%) | 1784 | 110 |
| gcc | 142,359,130 | 21,579,307 (15.2%) | 9531 | 2020 |
| xlisp | 1,307,000,716 | 147,425,333 (11.3%) | 489 | 48 |
| sc | 889,057,008 | 150,381,340 (16.9%) | 1269 | 157 |
| groff | 104,943,750 | 11,901,481 (11.3%) | 6333 | 459 |
| gs | 118,090,975 | 16,308,247 (13.8%) | 12852 | 1160 |
| mpeg_play | 99,430,055 | 9,566,290 (9.6%) | 5598 | 532 |
| nroff | 130,249,374 | 22,574,884 (17.3%) | 5249 | 228 |
| real_gcc | 107,374,368 | 14,309,867 (13.3%) | 17361 | 3214 |
| sdet | 42,051,812 | 5,514,439 (13.1%) | 5310 | 508 |
| verilog | 47,055,243 | 6,212,381 (13.2%) | 4636 | 850 |
| video_play | 52,508,059 | 5,759,231 (11.0%) | 4606 | 757 |

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

| Benchmarks | Number of static conditional branches constituting the given portion of the dynamic instances | | | |
|---|---|---|---|---|
| | first 50% | next 40% | next 9% | remaining 1% |
| espresso | 12 (1.0%) | 93 (5.2%) | 298 (16.7%) | 1,376 (77.1%) |
| mpeg_play | 64 (1.1%) | 468 (8.4%) | 1,372 (24.5%) | 3,694 (65.8%) |
| real_gcc | 327 (1.9%) | 2,877 (16.6%) | 8,398 (48.4%) | 5,749 (33.1%) |

**Table 2: Branch execution frequency for three benchmarks**

marks, a small number of distinct branches contribute the overwhelming majority of the branch instances. Only gcc exercises a substantial number of branches. The small number of branches is related to the small instruction cache footprints for these programs that, previous studies have shown, render them unsuitable for evaluating the instruction caches for today's processors [Gee93, Uhlig95].

The IBS-Ultrix benchmarks all exercise substantially more branches than the five small-footprint SPECint92 benchmarks. The benchmarks include a run of the GNU C compiler, real_gcc, which, though run on different inputs than the SPECint92 gcc benchmark, is quite comparable to the latter. The remaining programs are somewhat smaller than either version of gcc. It is important to note that the inclusion of operating system references does not strongly affect the predictability of branches in these programs, aside from the consequences of trying to predict a greater number of branches. The operating system code branch behavior falls within the range covered by the IBS application programs.

A higher number of branches does not necessarily lead directly to greater difficulties for predictors. A large proportion of the branches, both for the applications and for the operating system, are very highly biased; they are either almost always or almost never taken. These branches include loops, error and bounds checking, and other routine conditionals. Other studies have noted the frequency of highly biased branches in the SPECint92 benchmarks and similar programs and remarked upon their importance in achieving high accuracy [FisherFreudenberger92, ChangHaoYehPatt94, YoungGloySmith5]. SPECint92's gcc and the IBS-ultrix benchmarks execute, proportionally, even more instances of these highly biased branches. A possible consequence of this is that the small-footprint SPECint92 programs may benefit more from prediction mechanisms designed for less biased branches than do larger programs with branches that are more numerous, but also more easily predicted.

In this study we use the misprediction rate for conditional branches as the figure of merit. Changes in misprediction rate do not translate directly into changes in performance. The performance penalty associated with branches will depend, among other factors, upon the density of branches within code, the instruction-level parallelism available and exploited, the depth of pipelines, and the availability or lack of availability of the branch target instruction. A number of studies have made careful assessment of the link between changes in the misprediction rate and changes in performance[McFarlingHennessy86, FisherFreudenberger92, YehPatt92b, CalderGrunwald94, CalderGrunwaldEmer95]. We restrict ourselves in this study to an assessment of the systematic variation in the misprediction rate.

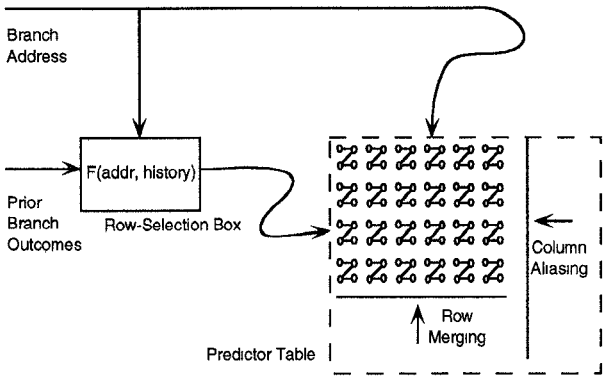



**Figure 1: The general model of two-level dynamic branch prediction**

Due to space limitations, we focus here on results for three benchmarks: SPECint92's espresso and IBS-Ultrix's mpeg_play and real_gcc. Full results for all of the benchmarks are available as a technical report [SechrestLeeMudge96]. The benchmarks on which we focus represent a range of program sizes. Espresso is a Boolean logic minimizer used in the generation of PLAs, the mpeg_play trace records the software decompression and display of a short video-clip, and real_gcc records a run of the GNU C compiler. In all of these benchmarks, half of the conditional branch instructions executed arise from less than two percent of the distinct conditional branch instructions, and in all cases a large number of branches are executed quite infrequently (see Table 2). The IBS-Ultrix benchmarks, however, are larger and spread the remaining half of the conditional branch instructions executed among a larger portion of the distinct conditional branch instructions.

## 3 Two-Level Predictors

Figure 1 shows a general model of a two-level predictor. The second level of such a predictor is generally a table of state machines, such as two-bit saturating counters. To make a prediction, one state machine is selected by row and column; the state of this machine determines the prediction, while the actual branch outcome determines the selected machine's next state. The column is selected using the branch address, while the row is selected by additional hardware. This hardware constitutes the first level of the predictor. This row-selection box generally chooses a row as a function of the branch address being predicted and the outcome of previous branches. It may keep separate history buffers for some number of branch addresses or it may use a single global history buffer.

Absent the limitations of area and access time, we might build a two-level predictor with a very large table of counters as predictors. Each distinct conditional branch address would be given a separate column in the table, with the rows representing conditions under which that branch's outcome is likely to vary. For example, separate state machines might be kept for the various combinations of outcomes of past executions of the branch, or the outcomes of correlated branches. It would be the responsibility of the row-selection box to record this history and from it to determine the appropriate row for each branch instance. The cost of the row-selection box could vary considerably, depending upon the selection scheme that we chose to implement.

Real predictors must, of course, make due with more limited resources. It is therefore necessary to determine a cost-effective combination of row-selection strategy and predictor table organization. Many possibilities have been suggested. Yeh and Patt [YehPatt92a] introduced a taxonomy for such two-level schemes, coding them with three letters. In the context of the model shown in Figure 1, the first letter states whether the row selection is based upon history kept globally (G), kept for a set of addresses (S), or kept for individual addresses (P). The second letter indicates the predictor table contains either an adaptive state machine, such as a two-bit saturating counter (A), or a fixed prediction (S). The third letter indicates whether the table has a single column used for all addresses (g), a set of columns indexed by bits extracted from the address (s), or a separate column for every address (p).

To reduce the cost of a predictor, it is generally necessary to make do with a smaller table than might give ideal results. An ideal table would be limited only by the necessity that the benefit of adding additional rows or columns must exceed the cost of training the additional state machines. Benefit derives from distinguishing between two cases that are likely to have different outcomes. With two-bit counters, the training cost of an individual counter is quite small, but if the row indexing scheme requires that the number of rows be doubled for every additional case considered, the limit for cost-effective subcasing will be reached fairly quickly. In a real table, where there is a real time or space penalty for every additional case considered, the cost-effective limit is reached even quicker. In this case, rows that are distinguishable and might serve to improve prediction accuracy must be merged. Similarly, while an ideal table would maintain a separate column for each branch whose executions reached some threshold, a real table must make do with a smaller number in which multiple branches will end up mapped to the same column. This column aliasing can be harmless if the behavior of the branches for the different subcases represented by rows is the same. If columns representing branches with dissimilar behaviors are combined, the cost will depend upon the extent of the dissimilarity and the dynamic frequency of the subcases for which they differ.

For a fixed number of state machines there are a variety of possible predictor table configurations. One extreme is to merge together all the subcases for a given branch and simply use the branch address to select a state machine from a single row. We will call this an address-indexed predictor. At the other extreme, one can reduce the table to single column. The selection of a predictor then depends upon the behavior of the row-selection box. One strategy is to use the outcome of the last n branches executed as an index into a column of $2^n$ state machines. Yeh and Patt encode this as a GAg scheme. An alternative is to keep a separate n-bit branch outcome register for every branch encountered, and to use the appropriate register to index into the column of state machines. Yeh and Patt encode this as PAg.

Figure 2 shows the misprediction rates for the SPECint92 and IBS-Ultrix benchmarks using address-indexed prediction tables of various sizes. The rate is plotted as a function of the number of address bits used to index the counters, with the gray or white tier for address length n represents the misprediction rate with $2^n$ two-bit counters in the row. The figure shows the accuracy using rows ranging from 16 counters in the rear to 32,768 counters in front. Because of the small number of branches exercised by the five small SPECint92 programs, all but the smallest of tables assign a separate column to each frequently executed branch, and no additional improvement can be found by increasing the table size. For gcc and the IBS-Ultrix benchmarks, on the other hand, some degree of aliasing occurs even for the largest tables. Aliasing conflicts between branches occur when consecutive branch instances accessing a particular counter arise from distinct branches. These conflicts correspond to the conflicts in a direct mapped cache. For mpeg_play, 6.24% of the accesses in a 1024-entry address-indexed table conflict. For a 8,192-entry table, the aliasing rate is

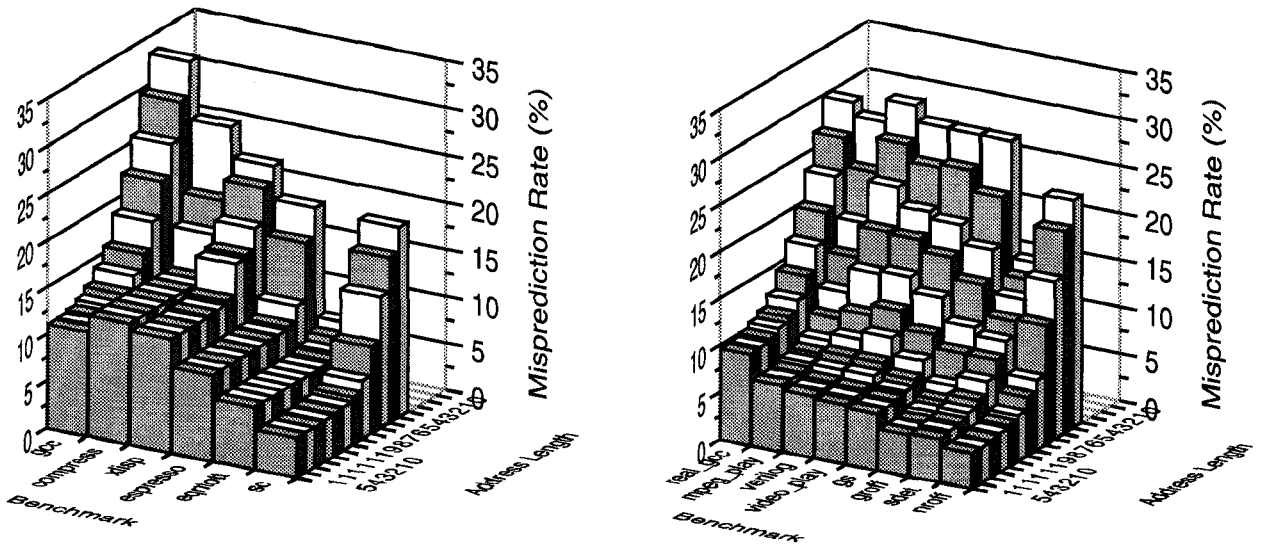



**Figure 2: Misprediction rates using a row of two-bit counters: address-indexed predictors**

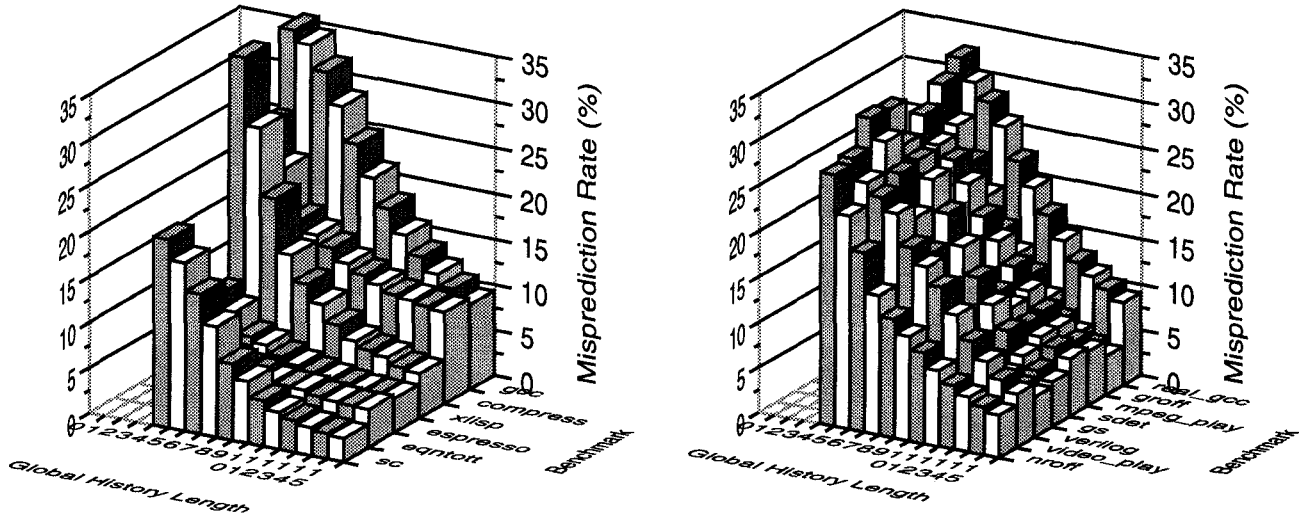



**Figure 3: Misprediction rates using a column of two-bit counters: GAg**

0.80%. For real_gcc the corresponding rates are 8.40% and 1.59%. For small tables, the IBS-Ultrix benchmark sdet bears some resemblance to the small SPECint92 programs, in that only 8 distinct branches account for 50% of its dynamic instances. However, since the remaining 50% of the dynamic instance are spread over a large number of branches, it does not saturate even large tables.

Figure 3 shows the misprediction rates for the benchmarks using a GAg scheme. The misprediction rates for a column of 16 two-bit counters is shown by the rear tier and the misprediction rates for 32,768 counters by the front tier. Using the global history to select from a column of counters proves to be an effective branch prediction strategy. As McFarling points out [McFarling92], many global history patterns occur only in concert with specific branches. Moreover, several patterns may be associated with each branch, providing the potential for discovering correlation between prior branches and the branch under consideration. For a sufficiently long column, all of these subcases can be given separate counters. For shorter columns, these patterns will be merged with one another, yielding a shorter pattern associated with two or more branches, or a history too short to include the branch most highly correlated with the outcome of the branch under consideration. With fewer branches, the small SPECint92 benchmarks suffer less aliasing under GAg than do the larger programs and, thus, have better accuracy for shorter histories. It should be noted that the aliasing for GAg is not always harmful. Approximately a fifth of the aliasing for the larger benchmarks was for the pattern with all recorded branches taken. This corresponds to repeated execution of a tight loop. The behavior of all such loops is identical, so all occurrences of the all-ones pattern, without regard to the branch under consideration, could, without harm, be aliased to a single counter.

# 4 Prediction Schemes Using Global History

The address-indexed and GAg schemes are only the extreme points of a range of predictor table configurations. GAs schemes generalize the GAg scheme by allowing the address to be used to select one of a set column, while using the global history to select the row. One can arrange $2^n$ state machines in n+1 configurations of $2^c$ columns and $2^r$ rows, where c+r = n. At the extremes, the configuration with $2^n$ rows is identical to the GAg scheme just described, while the configuration with $2^n$ columns is identical to the address-indexed schemes.

We can view the performance of the various GAs schemes as forming a surface interpolating between the performance curves for the address-indexed and the GAg schemes given in Figures 2 and 3. Surfaces for the benchmarks espresso, mpeg_play, and real-gcc are shown in Figure 4. Each gray or white tier represents a constant number of two-bit counters, ranging from 16 for the rear-most to 32,768 for the frontmost. Each tier ranges from the address-index configuration on the left to the GAg configuration on the right. Within each tier, we have blackened the top and sides of the bar representing the configuration with the best performance.

The distinction between the shape of the GAs performance surface for espresso and those for mpeg_play and real_gcc is striking. Furthermore, the shapes for all of the IBS-ultrix benchmarks are variants of this shape, while the small SPECint92 benchmarks are similar to that for espresso. The primary determinant of the shapes of these surfaces is the number of distinct branches exercised and the aliasing conflicts that result. This can be seen from Figure 5. The blackened bars represent the same best-in-tier configurations shown in Figure 4. The number of branches in espresso is so small that for moderate size tables it is possible to devote several counters to each branch. Under these circumstances, very little aliasing occurs provided even a few address bits are used. Aliasing will occur, even in large tables, if one combines global history with only a bit or two of address. This aliasing, however, is mostly the harmless aliasing that occurs for all-ones patterns in loops. For mpeg_play and real_gcc, in contrast, a good deal of aliasing occurs even in moderate size tables. Doubling the number of rows in the table, at the expense of halving the number of columns, will increase aliasing, since the global history is less useful at distinguishing between branches than are the branch addresses themselves. Since a high proportion of the branches are strongly biased, the penalty for any additional aliasing is so severe that it far outweighs any possible benefit from using history bits to assign instances to subcases within a column. Thus, the best performance available for small to moderately-sized tables comes from the simple address-indexed scheme. For larger tables, however, aliasing will be low as long as sufficient address bits are used. For these tables, dividing a column into additional rows can pay off up to a point.

The tilt towards small programs of the SPECint92 benchmark suite, and of the SPECint89 benchmarks before it, have skewed the understanding of the performance of global history schemes in two ways. First, the relatively low bias of the active branches for the small programs, particularly for eqntott and compress, tends to overstate the benefits of associating multiple state machines with individual branches. Second, the harmful effects of aliasing between branches are demonstrated by only a single program, gcc. The effect of second-level table size and configuration on prediction accuracy has been investigated using SPEC benchmarks by Pan, So and Rahmeh [PanSoRahmeh92], in their original work on correlating predictors, by Yeh and Patt [YehPatt91, YehPatt93], and, more recently, by Talcott, Nemirovsky and Wood[TalcottNemirovskyWood95]. The results presented by Pan

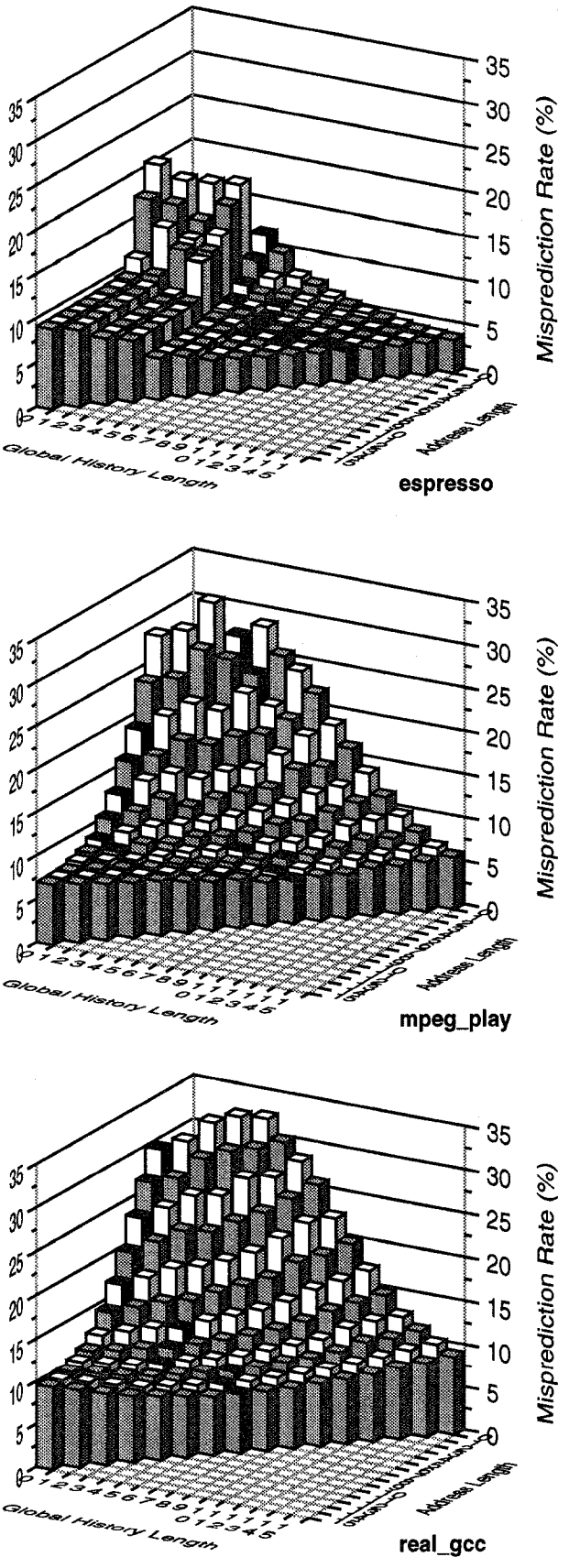



**Figure 4: Misprediction rates for GAs schemes**

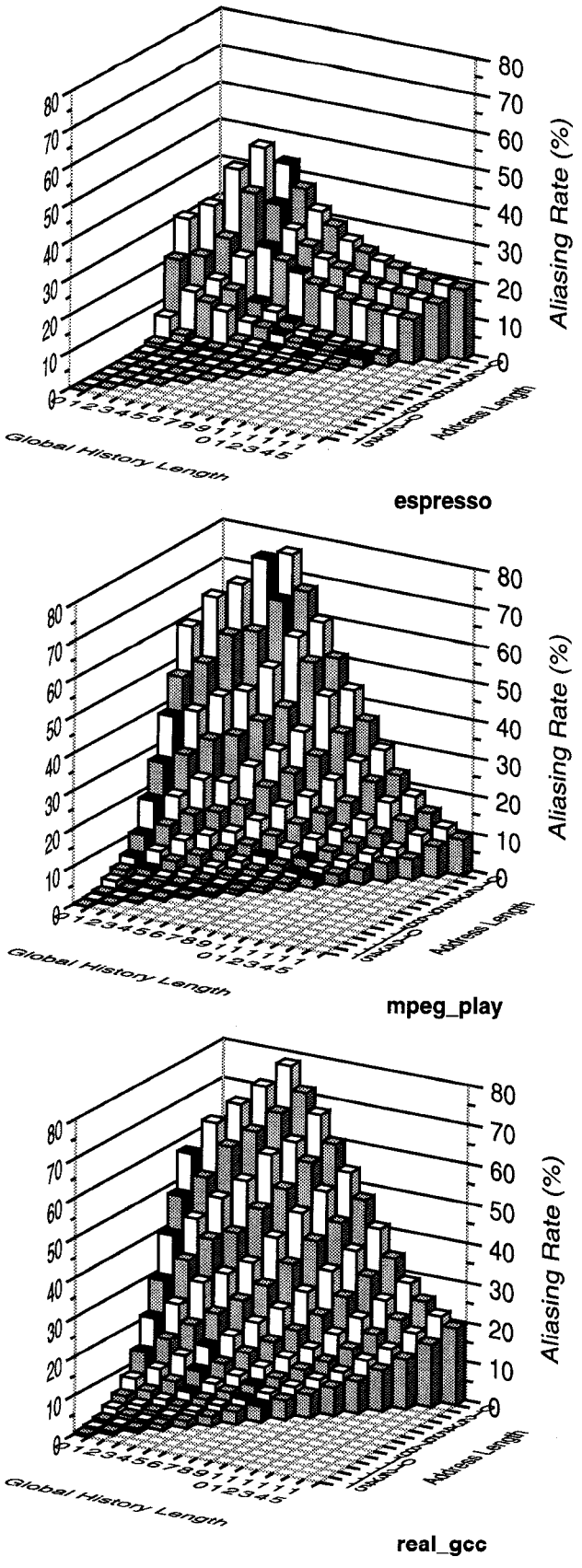



**Figure 5: Aliasing rates for GAs schemes**

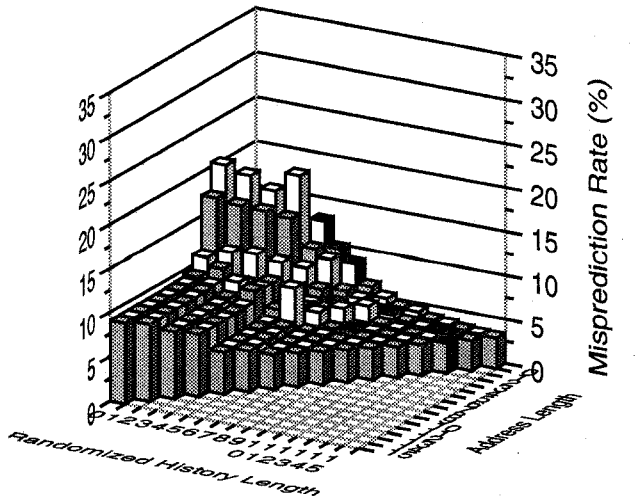
*et al.* for a number of configurations predicting li, and by Yeh and Patt [YehPatt93], for an average of four of the benchmarks, are quite similar to those shown in Figure 4 for espresso. Talcott *et al.* present results for a mixed of integer and floating point SPEC benchmarks. They report the results for each benchmark for the eleven possible configurations of a table of 1024 state machines, but, in a sense, the gcc results are outvoted by the remaining benchmarks six to one. Yeh [Yeh93], in simulations of two specific table configurations, and Talcott *et al.*, in a study of eleven configurations, considered the effects of aliasing directly. While, these studies, as well as others, note that the prediction accuracy for gcc is limited by aliasing in the predictor table, the inference that gcc's behavior is likely to be representative of a wide range of applications is not drawn.

McFarling[McFarling92] proposed a variant to the GAs scheme just described, called gshare, in which the global history is exclusively ORed with bits from the branch addresses. His reasoning was that he could produce a row selector that would combine the information of the global history and address bits. He reasoned that in sufficiently large tables this would reduce aliasing between the global history patterns while retaining the advantages of using long global history to discover branch correlation. Since global history patterns are only weakly identified with particular branches, a short pattern might not be associated exclusively with one of two branches aliased to a particular column. By XORing the pattern with each of the addresses, one can produce two new patterns, each associated with a distinct branch. As in GAs, additional address bits can be used to select one of several columns. Hence, for a fixed table size there is a range of gshare configurations. McFarling compared the best performance for a given size predictor table of any GAs configuration with the best performance for any gshare configuration and found a slight advantage for the gshare schemes on the SPECint92 benchmarks. We should note in passing that many subsequent studies of gshare have been limited to configurations with a single column.

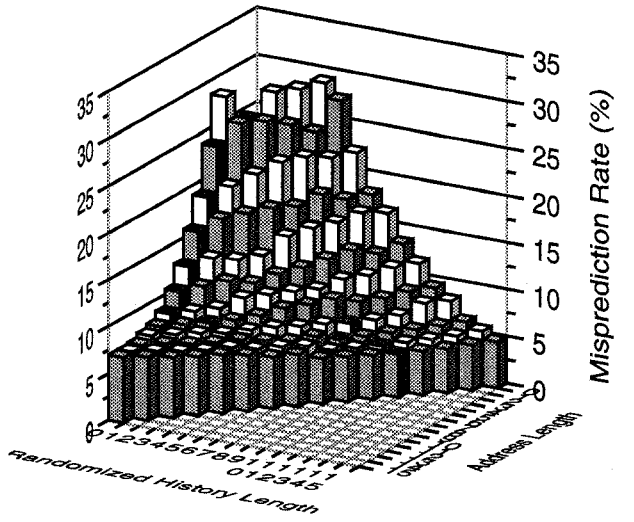
Figure 6 shows the results of simulations of gshare for the three example benchmarks. We see that the results are almost identical to those for GAs. Note that the leftmost configurations within each tier are for address-indexed prediction and are thus exactly the same as the leftmost configurations in Figure 4. As before, the blackened bars represent the best accuracy within a constant-size tier. Note that for small benchmarks, such as espresso, single-column configurations, which in many studies are the only gshare configurations evaluated, perform well. For large benchmarks, however, such configurations are suboptimal.

Figure 7 shows the difference for the mpeg_play benchmark between the GAs and gshare schemes for identically configured predictor tables. Positive numbers indicate superior prediction by gshare. We see that the differences are quite small. The areas of superior performance are clustered on the right side of the graph, where the table has more rows than columns. It is here that aliasing is highest, since global history patterns are less effective at distinguishing branches than are address bits. However, as we have pointed out, not all of this aliasing is destructive, so gshare achieves some of its reduction in aliasing by eliminating harmless aliasing. These configurations are, in fact, suboptimal for both GAs and gshare, so improving their performance is not meaningful. In the area towards the center of the graph, where GAs schemes achieve their best performance, GAs and gshare differ little in performance, although gshare does slightly expand this number of these configurations.

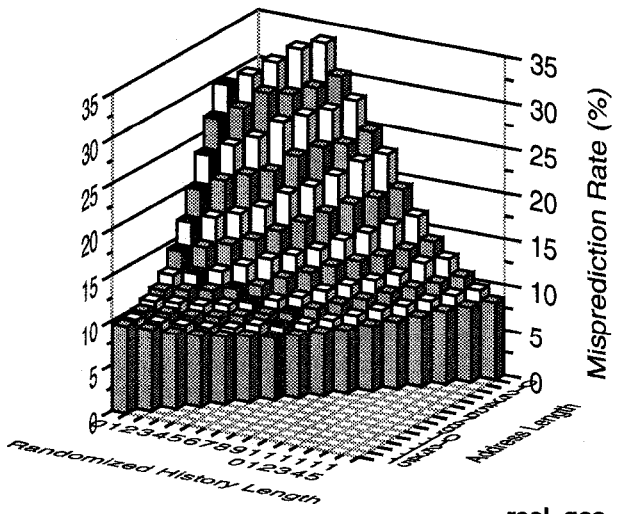
Another variant of GAs has been proposed by Nair[Nair95]. Rather than selecting a row by maintaining a history of branch outcomes, Nair proposes that these outcomes be encoded by storing a small number of bits from the addresses of branch targets. Nair reasons that this could reduce aliasing between history pat-




espresso




mpeg_play




real_gcc

**Figure 6: Misprediction rates for gshare schemes**

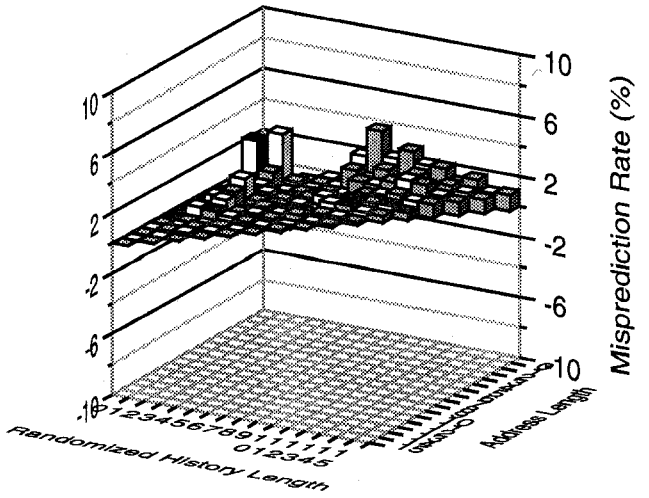



**Figure 7: Differences in misprediction rates between gshare and GAs for mpeg_play**

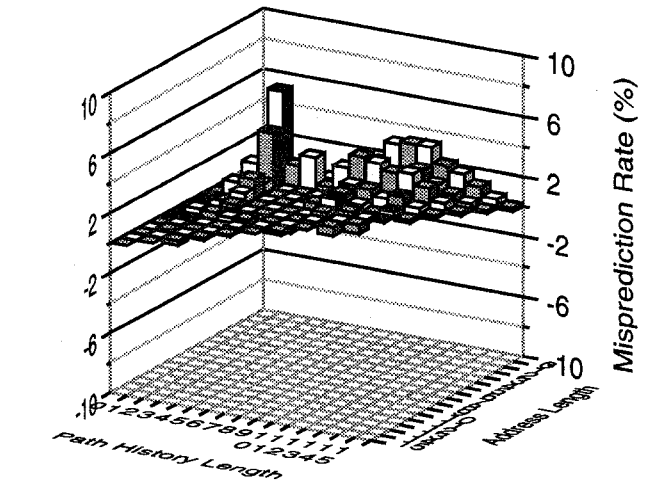



**Figure 8: Differences in misprediction rates between path and GAs for mpeg_play**

terns, allowing separate predictors to be used for distinct paths to a particular branch. Note that row merger can cause both the failure to distinguish between patterns associated with different branches and failure to distinguish between patterns associated with two behaviors of the same branch. McFarling attacked the first, while Nair attacks the second. Nair reports the results of simulations of a single configuration of his scheme (a $2^6$ x $2^4$ table) and a single configuration of GAs (a $2^7$ x $2^3$ table). He found little difference between the two configurations simulated for the SPECint92 benchmarks and an additional program. Nair did report that for gcc and his additional program, both simulated configurations were inferior to an address-indexed table of equivalent size, while for the small-footprint SPECint92 programs both schemes were superior. Note that the GAs configuration studied is one for which GAs suffers from aliasing for programs with large numbers of branches, hurting its performance relative to simple address indexing.

IBS-Ultrix benchmarks. Figure 8 shows the difference between the GAS and Nair's path scheme for the mpeg_play benchmark. As in Figure 7, positive numbers indicate superior prediction by the path based prediction scheme. The results of simulations of Nair's path encoding show that, as with gshare, the path scheme reduces aliasing for configurations with very few columns. Again, these are not the configurations for which GAs performs the best. For configurations with equal rows and columns or with more rows than columns, Nair's path encoding generally does slightly worse than GAs. Nair points out the weakness of his scheme that is the likely culprit. By using more than one bit to encode the outcome of a single event, he has limited the number of events that can be encoded in a given set of bits. Nair suggests that with alternative path codings using hashing, his scheme is likely to consistently outperform the GAs scheme, but simulations by Young, Gloy and Smith [YoungGloySmith95] suggest otherwise. Young *et al.* demonstrate the destructive effects of aliasing for GAs and gshare schemes for a single configuration and suggest that the way to better dynamic prediction schemes is to try to exploit the same pattern and address information as gshare, while avoiding destructive aliasing.

# 5 Prediction Schemes Using Per-Address History

GAs schemes are primarily effective when the predictor table is large enough to allocate several counters for every active branch. This may appear as a large number of columns selected by address or as a large number of rows selected by global patterns associated with particular branches. Aliasing is limited in the first case by ensuring that the number of columns is sufficient to handle all of the active branches and in the second by making only sparse use of rows.

Predictor tables could be made smaller without sacrificing accuracy if the additional aliasing that resulted was relatively harmless. This would be the case if the rows or columns to be combined were close on an entry-by-entry basis. The meaning and meaningfulness of any particular global history pattern varies greatly from branch to branch, since it reflects the outcome of a variety of branches, often including previous instances of the branch now to be predicted. Thus, the columns of the predictor table for global history schemes tend to be quite distinct. Self history patterns, on the other hand, often indicate a pattern of behavior whose meaning is independent of the particular branch. The appropriate predictions for the most frequently occurring patterns are strongly correlated across branches [SechrestLeeMudge95].

PAs schemes can take advantage of this strong correlation. In these schemes the row-selection box maintains history buffers for individual branches. Relying on per-address information makes the most frequently selected rows close to uniform across columns. The predominance of these patterns is so strong that provided that enough self-history bits are used to distinguish among the patterns, very little is lost by collapsing all columns into a single row. Figure 9 shows the misprediction rates for various PAs schemes, with the assumption that accurate history information is available for each branch. Note that for any given table size, the configuration with a single column is optimal or close to optimal. These surfaces are relatively flat, however, so little is lost by substituting address bits for history bits, so long as at least a few history bits are retained. Furthermore, increasing the size of this table adds little to the predictor's accuracy. For mpeg_play, the difference between the misprediction rate using 16 counters and the rate using 1024 is only 1.9%, and between 1024 counters and

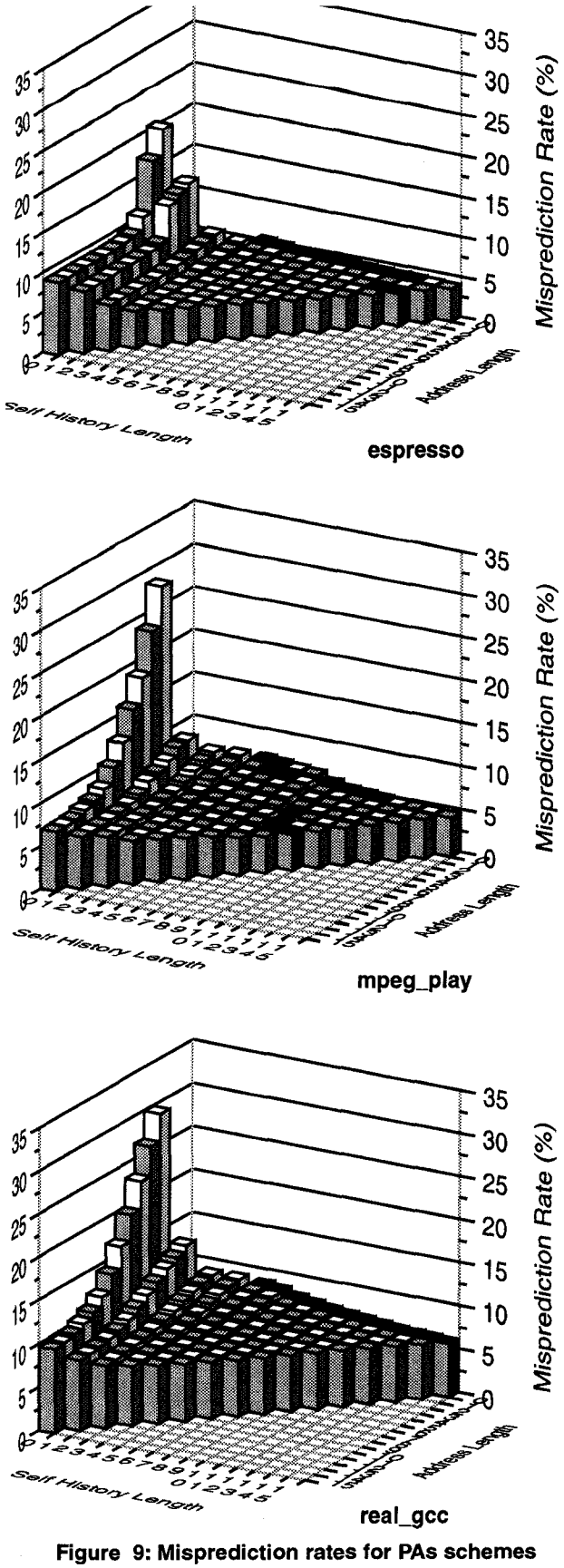



**Figure 9: Misprediction rates for PAs schemes with perfect histories**

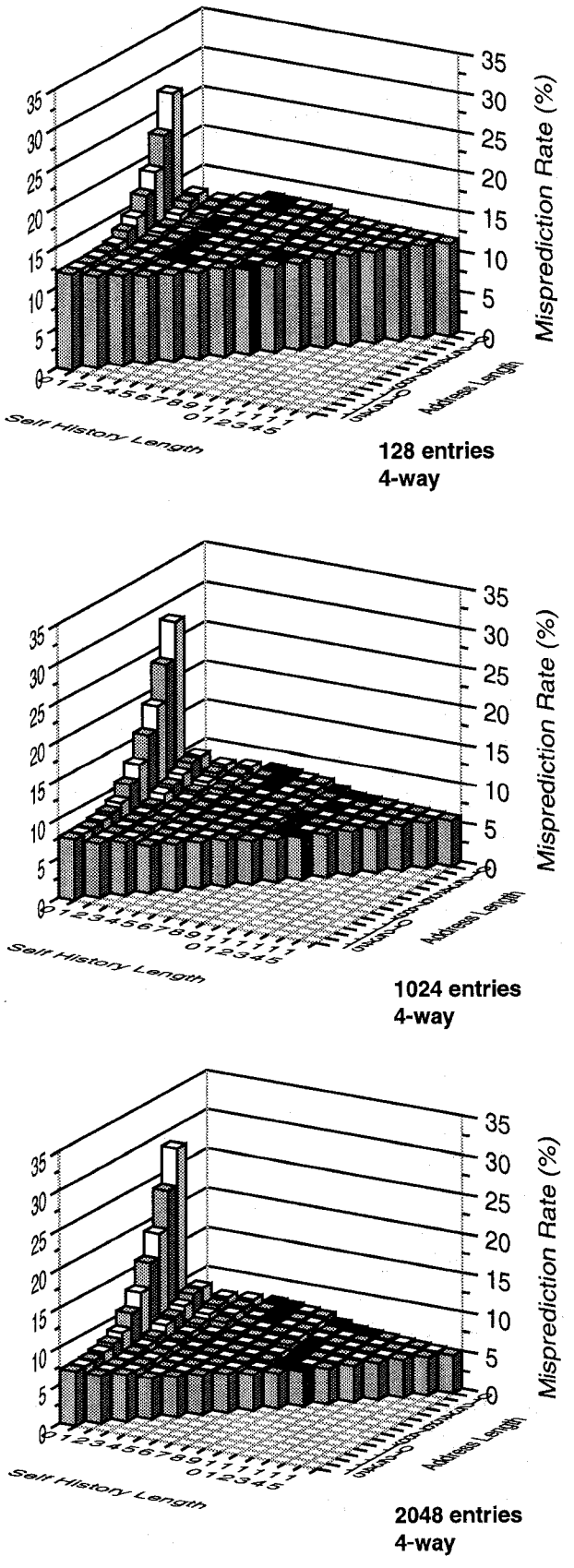



**Figure 10: Mispredict rates for PAs schemes with various first-level tables for mpeg_play**

32,768 only 1.0%. For real_gcc the corresponding differences are 1.9% and 0.7%.

Accurate prediction is possible from a small second-level table if accurate history information is available. Realistic implementations of PAs schemes will store branch histories in a first-level table of some bounded size. Conflicts between branches can result in the pollution of the stored history information. The conflict rates in a direct mapped first-level table are the same as the aliasing rates in a address indexed second-level table. These are given on the left side of Figure 5. These miss rates can be reduced by using some degree of associativity. In Figure 10 we show the misprediction rates that result for mpeg_play using 128-, 1024- and 2048-entry four-way set associative tables in the row selection scheme. In these simulations, we assume that conflicts in the first-level table are detected through tags, and the history reset so a fixed mixture of zeros and ones. (We use the appropriate length prefix of the pattern 0xC3FF, avoiding excessive aliasing for the patterns of all taken or all not taken branches.) With this policy, pollution in the first-level table raises the misprediction rates for most table sizes and configurations more or less uniformly.

The results in Figure 10 show the penalty paid for a small first level table. For a 128-entry first-level table the misprediction rate for the configuration of a single column of $2^{15}$ counters is 6.94% above that for an infinite first-level table. For 1024 entries, this penalty falls to 1.19% and, for 2048 entries, it falls to 0.44%. With only 128 entries in the first table, one is better off relying, even for large configurations, on address bits alone. Given a fixed resource budget, one has to choose in a PAs scheme between throwing additional resources into the first-level table or into the second level table. It is notable that relatively small second-level tables can achieve most of the accuracy possible with the scheme. A 1024-counter prediction table, for example, may well provide adequate performance. Rather than adding counters to the second-level table, it may be most cost effective to add additional entries to the first-level table. For example, 65,536 bits can be used to implement a table of 32,768 counters, or a table of 1024 counters and enough history bits to keep 10 bits of history for 6348 branches. We have omitted here the cost of address tags for the first-level table. In some designs it is possible to integrate the branch history cache with a branch target buffer or with the instruction cache and, thus, avoid having to implement additional tag bits.

Yeh and Patt [YehPatt91] first proposed the PAs organization, and explored the importance of the size of the first-level table. They simulated a 4096 counter PAg scheme with 256-entry, 512-entry and unbounded first-level tables. Their results for gcc emphasized the importance, for that benchmark, of a large first-level table. In a later study [YehPatt93], however, when investigating alternative implementations with a large resource budget, they kept the first-level table at a constant 1024-entry size, with 4-way set associativity. They found that adding large amounts of resources to the second-level predictor table gave almost no benefit. In a later study, Calder and Grunwald[CalderGrunwald94] compared the results for a 2048-entry and 4096-entry GAg scheme with those for a 64-row, 16-column PAs scheme with a 512 entry 4-way set associative first-level table. They found, that the GAg schemes gave better prediction accuracy, but failed to note that additional resources, properly applied, would aid the PAs scheme much more than the GAg schemes.

# 6 Conclusions

Table 3 lists, for each benchmark and for each of the schemes considered, the best configuration for a range of predictor table sizes, along with the corresponding misprediction rate. This table

| Benchmark | Predictors | First-level Table Miss Rate | 512 Counters | 4096 Counters | 32768 Counters |
|---|---|---|---|---|---|
| espresso | GAs | — | $2^6 \times 2^3$ (4.79%) | $2^8 \times 2^4$ (3.99%) | 211 x 24 (3.52%) |
| | gshare | — | $2^8 \times 2^1$ (4.83%) | $2^8 \times 2^4$ (3.82%) | 213 x 22 (3.33%) |
| | PAs(inf) | — | $2^9 \times 2^0$ (14.61%) | $2^{12} \times 2^0$ (4.34%) | 213 x 22 (4.06%) |
| | PAs(1k) | 0.01% | $2^9 \times 2^0$ (4.62%) | $2^{12} \times 2^0$ (4.35%) | 213 x 22 (4.06%) |
| | PAs(128) | 0.44% | $2^9 \times 2^0$ (4.83%) | $2^{12} \times 2^0$ (4.57%) | 213 x 22 (4.28%) |
| mpeg_play | GAs | — | $2^0 \times 2^9$ (10.61%) | $2^6 \times 2^6$ (7.23%) | 29 x 26 (4.95%) |
| | gshare | — | $2^0 \times 2^9$ (10.61%) | $2^8 \times 2^4$ (6.90%) | 211 x 24 (4.58%) |
| | PAs(inf) | — | $2^9 \times 2^0$ (5.41%) | $2^8 \times 2^4$ (4.84%) | 29 x 26 (4.22%) |
| | PAs(2k) | 0.97% | $2^9 \times 2^0$ (5.85%) | $2^8 \times 2^4$ (5.27%) | 29 x 26 (4.67%) |
| | PAs(1k) | 2.68% | $2^9 \times 2^0$ (6.5%) | $2^8 \times 2^4$ (5.92%) | 29 x 26 (5.34%) |
| | PAs(128) | 17.9% | $2^3 \times 2^6$ (11.53%) | $2^3 \times 2^9$ (10.93%) | 27 x 28 (10.53%) |
| real_gcc | GAs | — | $2^0 \times 2^9$ (14.45%) | $2^3 \times 2^9$ (9.59%) | 27 x 28 (6.82%) |
| | gshare | — | $2^0 \times 2^9$ (14.45%) | $2^4 \times 2^8$ (9.52%) | 26 x 29 (6.76%) |
| | PAs(inf) | — | $2^9 \times 2^0$ (7.05%) | $2^{12} \times 2^0$ (6.5%) | 215 x 20 (6.15%) |
| | PAs(2k) | 1.89% | $2^9 \times 2^0$ (8.05%) | $2^{12} \times 2^0$ (7.51%) | 215 x 20 (7.17%) |
| | PAs(1k) | 3 88% | $2^9 \times 2^0$ (9.09%) | $2^{12} \times 2^0$ (8.55%) | 215 x 20 (8.23%) |
| | PAs(128) | 22.28% | $2^2 \times 2^7$ (17.88%) | $2^3 \times 2^9$ (16.78%) | 25 x 210 (16.2%) |

**Table 3: Best configurations for various predictor table sizes for three benchmarks**

points to several conclusions. Although it has been suggested that global history schemes would be able to draw upon information not available from self history, even for large tables, aliasing can undercut this advantage, and PAs schemes can provide more accurate predictions. The advantage of PAs is more pronounced for smaller second-level tables, in which the aliasing rate for global history schemes can be quite high. For smaller table sizes, the global history schemes are, in fact, less effective than a conventional table of address-indexed predictors. To be effective, however, PAs schemes need sufficient resources in the first-level table. Collisions in this first-level table tend to harm prediction rates almost uniformly, without regard to the size or configuration of the second-level table. Table 3 also points out that optimal configuration for a particular scheme and table size will vary considerably with the benchmarks chosen. In the presence of aliasing, global history schemes will typically need more address bits to most effectively predict larger programs. For larger tables, the gshare scheme enjoys a slight advantage over GAs schemes. Self history schemes, on the other hand, will typically perform slightly better using more history bits, but for an actual implementation, the misprediction advantage may be overshadowed by the additional storage expense.

We have reported the results of extensive simulations of a variety of branch prediction approaches and configurations and have shown that the accurate prediction of large programs depends primarily upon the deployment of sufficient resources to keep track of information regarding a large number of branches, whether these resources are placed in a first-level or a second-level table. The small-footprint SPECint92 benchmarks have encouraged the idea that the resource requirements of global history schemes might be limited, but this is not the case. Recent work has begun to examine ways of combining schemes to provide more effective branch prediction. The results presented here suggest that controlling aliasing will be the key to improving prediction accuracy and taking advantage of inter-branch correlations in global schemes. This is much the same conclusion reached by Young *et al.* [YoungGloySmith95]. We have shown, however, that controlling aliasing in the first-level table of a PAs scheme is an alternative route to high prediction accuracy that may well prove more cost effective.

It will be important to recognize the position of branch prediction schemes under design within the larger space of possibilities, lest resources be misapplied. In their widely-used textbook, Hennessy and Patterson [HennessyPatterson95] make the statement, regarding GAs, that "the attraction of this type of correlating branch predictor is that it can yield higher prediction rates than the two-bit scheme and requires only a trivial amount of hardware." We have shown the degree to which this attractive element is subject to subversion for large programs. Branch prediction results require more careful interpretation, with the recognition that the benefits of correlation can easily be drowned by aliasing.

# Acknowledgments

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

# References

**[CalderGrunwald94]**Calder, B., and Grunwald, D., "Fast & Accurate Instruction Fetch and Branch Prediction," *Proceedings of the 21th International Symposium on Computer Architecture*, 2-11, Apr. 1994.

**[CalderGrunwaldEmer95]**Calder, B., Grunwald, D., and Emer, J., "A System Level Perspective on Branch Architecture Performance," *IEEE Micro-28*, Nov. 1995.

**[ChangHaoYehPatt94]**Chang, P., Hao, E., Yeh, T., and Patt, Y., "Branch Classification: a New Mechanism for Improving Branch Predictor Performance," *IEEE Micro-27*, Nov. 1994.

**[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.

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

**[HennessyPatterson95]** Hennessy, J. and Patterson, D., "Computer Architecture: a Quantitative Approach, 2nd ed.," Published by Morgan Kaufmann Publisher Inc., 1995.

**[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.

**[McFarlingHennessy86]**McFarling, S, and Hennessy, J., "Reducing the Cost of Branches," *Proceedings of the 13th International Symposium on Computer Architecture,* 396-403, 1986.

**[McFarling92]**McFarling, S. "Combining Branch Predictors," *WRL Technical Note TN-36*, Jun. 1993.

**[MicroReport95a]**Gwennap, L., "New Algorithm Improves Branch Prediction," *Microprocessor Report,* Mar. 27, 1995.

**[MicroReport95b]**Gwennap, L., "Nx686 Goes Toe-to-Toe with Pentium Pro," *Microprocessor Report,* Oct. 23, 1995.

**[Nair95]** Nair, R."Dynamic Path-Based Branch Correlation," *Proceedings of the 28th Annual International Symposium on Microarchitecture*, 15-23, Nov. 1995.

**[PanSoRahmeh92]**Pan, S.T., So, K., and Rahmeh, J.T., "Improving the Accuracy of Dynamic Branch Predication Using Branch Correlation," *Proc. 5th Annual Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 76-84, Oct. 1992.

**[SechrestLeeMudge95]**Sechrest, S, Lee, C-C, and Mudge, T, "The Role of Adaptivity in Two-Level Adaptive Branch Prediction," *IEEE Micro-28,* Nov. 1995.

**[SechrestLeeMudge96]**Sechrest, S, Lee, C-C, and Mudge, T, "Correlation and Aliasing in Dynamic Branch Predictors: Full Simulation Results," Tech. Report CSE-TR-283-96, Univ. of Michigan, Ann Arbor, MI, Feb. 1996.

**[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.

**[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.

**[YehPatt92a]**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.

**[YehPatt92b]**Yeh, T-Y. and Patt, Y. "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution," *Proceedings of the 25th International Symposium on Microarchitecture*, 129-139, Dec. 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,* 257-266, May 1993.

**[Yeh93]**Yeh, T-Y. "Two-Level Adaptive Branch Prediction and Instruction Fetch Mechanisms for High Performance Superscalar Processors," Ph.D Thesis, Tech. Report CSE-TR-182-93, Univ. of Michigan, Ann Arbor, MI, Oct. 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.