Alloying Global and Local Branch History: Taxonomy, Performance, and Analysis

Kevin Skadron, Margaret Martonosi, Douglas W. Clark
Depts. of Computer Science and Electrical Engineering
Princeton University
Princeton, NJ 08540

skadron@cs.princeton.edu, mrm@ee.princeton.edu, doug@cs.princeton.edu

Abstract

The need for accurate conditional-branch prediction is well known: mispredictions waste large numbers of cycles and also waste power on mis-speculated computation. A number of studies have explored ways to improve the prediction accuracy of two-level predictors, but have considered exclusively global or local history. Because most programs benefit from having both global and local history available, other work has proposed hybrid predictors that combine a global-history with a local-history component.

Building on prior work that categorizes the types of branch mispredictions, this paper presents a taxonomy of mispredictions. Using the taxonomy, the paper shows why a two-level predictor that *alloys* local and global history in the same predictor index provides superior performance. By making both types of history available all the time, an alloyed scheme obtains accuracies competitive with hybrid schemes, even though organized like conventional two-level predictors. Alloying has the advantage that it eliminates the need to make a choice between components, since the history types are merged. For large predictors used in high-performance processors, an alloyed organization provides competitive performance that suffers less variation from program to program than alternatives. For the smaller predictors needed in space- and power-constrained environments, alloying outperforms alternative organizations.

1 Introduction

Predicting the direction of conditional branches has received intense study in recent years, much of it focusing on dynamic, two-level predictors that use history about prior branch outcomes [18, 30]. Yet branch mispredictions remain an important performance bottleneck, because each misprediction results in five, ten, or more cycles of wasted instruction fetch. Even in out-of-order execution systems, mispredictions usually create pipeline bubbles, because the instruction window can rarely expose sufficient ILP to overlap these long penalties. Ranganathan and Jouppi predict that branch prediction will be the most important bottleneck in processors by 2010, worse even than the memory system [10]. Mispredictions also waste power on wrong-path instructions that are eventually squashed [14].

Most proposed predictor organizations today are either refinements of two-level predictors [13, 15, 16, 25], or hybrid organizations combining two or more components [7, 9, 15]. By tracking branch history, two-level predictors can identify branch behavior patterns that a *bimodal*, single-level table—one bias counter per branch site [23]—cannot expose. The history can be either *global*, tracking outcomes from different branches in the same history register, or *local*, tracking history separately for each branch. Global history can expose correlation among branches, while local history is well suited for branches that follow a consistent pattern. In a hybrid organization like the one in Figure 1, the components are themselves often two-level components; in this way, the overall predictor can make both global and local history available to the program.

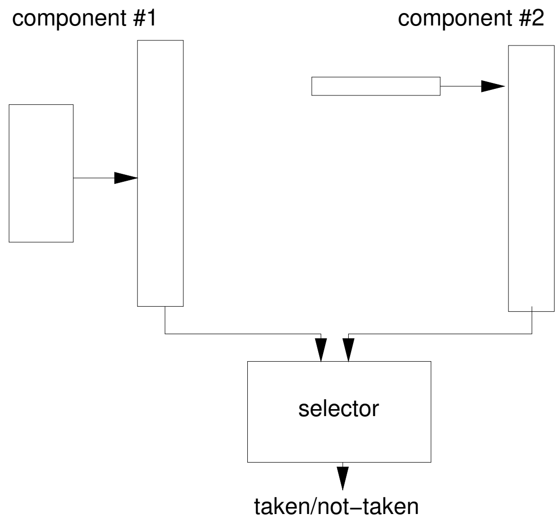

Figure 1: The organization of a hybrid predictor with two components. If the hybrid predictor statically assigns branches to one or the other component, the selector simply chooses the appropriate component depending on the branch's PC.

Both organizations have drawbacks. If designers choose a single two-level predictor, they must determine whether to use global or local history—but for most programs, there will be some branches where the choice leads to poor performance. Our results show the importance of such *wrong history* mispredictions by presenting a taxonomy of mispredictions which also identifies destructive interference in the table of bias counters (the *pattern history table* or PHT) and long training times as other important contributors to programs' misprediction rates. This taxonomy unifies and extends prior efforts to characterize mispredictions [3, 8, 11, 13, 15, 21, 25, 26, 30, 31]

Hybrid organizations make both local and global history available. Because they accomplish this by including multiple components, they also require a selection mechanism to choose which predictor to believe at any moment. Whether static or dynamic, the selector can fail, especially when individual branch sites frequently alternate between using the two types of history. A static selector necessarily encounters mispredictions for these branches, and a dynamic selector, in addition to requiring additional space, can have difficulty learning how to steer branches to the correct component.

Alloying. Based on our taxonomy and resulting data, we propose a pseudo-hybrid organization that *alloys* global and local history in the same predictor index, attacking the drawbacks of both organizations. This is a small change to the organization of a two-level predictor, yet it makes both local and global history available and thus provides hybrid-like capabilities without the need for a selection mechanism. Figure 2 shows the organization we propose.

We use our taxonomy to compare our proposed alloyed organization—which we term MAs—against two well-established, two-level predictors: GAs [18] and PAs [30].[1] Because history patterns often fail to

[1]In this naming scheme, the first letter indicates the type of history (G = global; P = per-address or local), the second indicates whether the PHT is adaptive or static, and the third indicates whether the PHT consists of one global column, or a set of PHTs selected by the address bits. The "M" in MAs stands for "merged history."

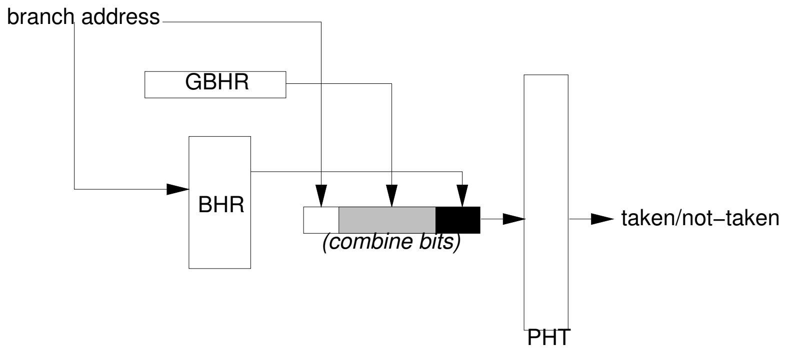

Figure 2: The organization of a two-level predictor with an alloyed index. This predictor combines local history from the per-branch history table (BHT) and global history from the global branch-history register (GBHR) with some address bits to compose the PHT index.

uniquely assign branches to a particular bias counter, GAs and PAs reduce PHT interference by sacrificing some history bits for the anti-aliasing ability of address bits. Alloying can use its richer mixture of bits to accomplish the same effect, and because it uses less of each type of history, tends to exhibit shorter training time, especially compared to GAs.

As mentioned earlier, a number of studies have proposed refinements to two-level prediction—especially to global-history based predictors. These primarily focus on reducing PHT interference. We forego these proposals, examining the older GAs and PAs organizations and the corresponding MAs organization. Because all three compose their index via concatenation, the comparison is fair, and all three would benefit from the newer techniques.

We also compare alloying to hybrid prediction. Alloying eliminates selection failures—there is no longer any selection to be made—but suffers from a corresponding rise in interference within its unified structure.

Generally speaking, alloying represents a medium between two types of branch prediction research. Proposed enhancements to two-level predictors have used exclusively local or global history. Work describing possible hybrid organizations has provided both types of history, but by using multiple existing two-level predictors and selecting among them. Alloying combines these approaches, as depicted in Figure 3. One other study is noteworthy in this regard: the application by Emer and Gloy [6] of genetic algorithms to grow branch predictor structures. This technique can combine different types of information by applying genetics-like mutations to the predictor structure, but seems not to have yielded a structure merging global and local history.

Contributions. Specifically, this paper:

- Develops a taxonomy of misprediction types and extends prior work showing the importance of PHT interference and wrong history.
- Shows that not only do programs as a whole need both global and local history, but so do individual branches.
- Proposes MAs, which alloys both global and local history in the same PHT index. When compared both against solo two-level predictors and against hybrid predictors, MAs provides excellent performance for a range of predictor sizes. In particular, MAs is the best overall choice for space- and power-constrained environments like embedded processors.

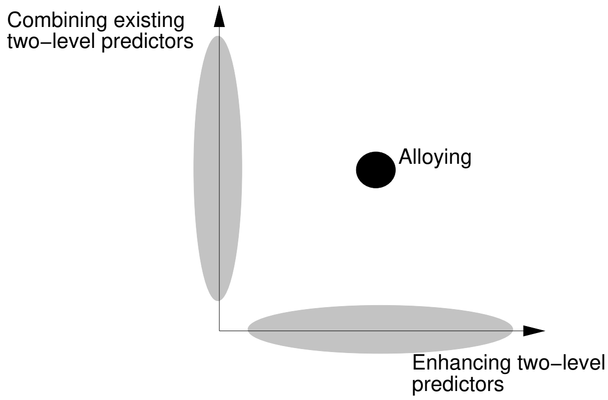

Figure 3: Alloying in relation to other proposed history-based branch-prediction organizations.

The paper is organized as follows. The next section presents our taxonomy of mispredictions, and Section 3 then describes our simulation environment and benchmark suite. Section 4 uses the taxonomy to compare MAs against GAs and PAs. Then, because a hybrid predictor is another way to make both global and local history available, Section 5 compares alloying to two varieties of hybrid prediction and Section 6 briefly compares small sizes of all the predictor organizations. Section 7 presents related work, and Section 8 concludes the paper.

2 Taxonomy

As mentioned, many studies have measured the importance of PHT interference as a source of mispredictions [3, 13, 15, 16, 21, 25, 26, 31], and other work has shown the importance of wrong-history mispredictions [8]. This section unifies the different misprediction types in a single taxonomy, includes the effects of training time [11], and describes a one-pass method for counting them. We also estimate how often branches require both local and global history simultaneously. In establishing categories, we focus specifically on mispredictions in history-based predictors like GAs, PAs, or MAs for a fixed history length.

Destructive PHT conflicts. All three predictors can suffer from PHT interference: different branches that map to the same PHT counter yet have opposite directions. A branch that mispredicts in a baseline organization (GAs, PAs, or MAs) but predicts correctly with an interference-free PHT (GAp, PAp, or MAp) is an instance of *destructive interference*.[2]

Aliasing in the BHT can also cause destructive or constructive interference. The BHT's design incorporates some additional variables that are beyond the scope of this taxonomy, so here we assume an interference-free BHT. Section 4.2 examines overall prediction accuracies when using a finite, set-associative BHT.

Training-induced mispredictions. If a misprediction is not caused by PHT interference, it can instead occur because the predictor has not yet learned the branch's behavior. This happens especially at the beginning of a program or after a context switch, but also occurs as programs transition from one phase to another. We have yet to devise a precise method for measuring training mispredictions, but propose that their contribution can be estimated with a bimodal predictor as follows.

[2]Note that constructive interference can also occur, so the expected gain from eliminating PHT conflicts would only be the difference between the two values.

PHT conflicts must first be eliminated as above. Then, if a branch mispredicts in a particular organization but predicts correctly in a simpler organization, we conclude that the former has not yet fully learned the branch's behavior. The simpler predictor we use as a reference is a bimodal predictor.

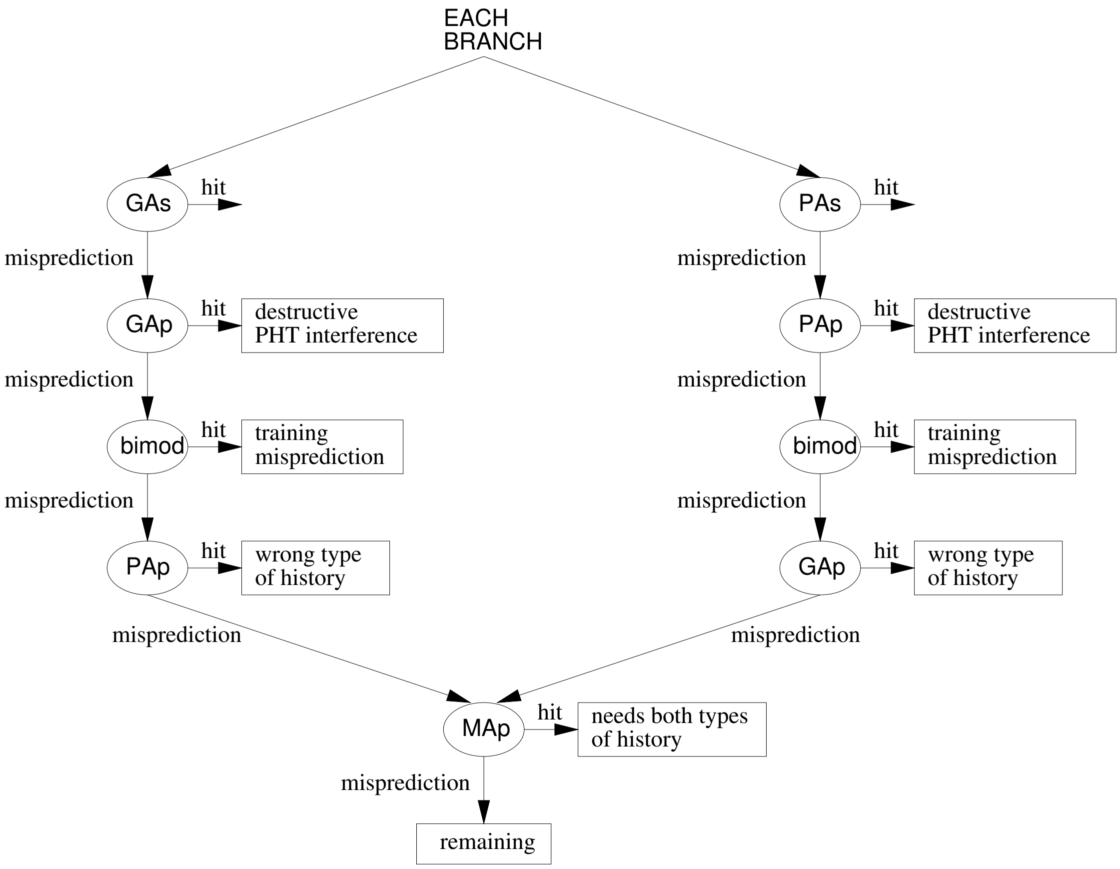

Figure 4: A flowchart depicting how we categorize misprediction types. Each dynamic branch flows down both sides until it is either labeled or falls through.

If a branch mispredicts in an n-bit-history GAp, PAp, or MAp predictor but predicts correctly in an $n-1$ bit GAp or PAp predictor, the misprediction occurred either because the n bit predictor has not yet reached steady state for the branch in question, or possibly due to a lucky guess by the $n-1$ bit predictor. Training mispredictions in an $n-1$ bit predictor can in turn be identified with an $n-2$ bit predictor. The extreme, an $n-n$ bit predictor, is the bimodal predictor. To avoid this potentially prohibitive sequence of tests, we collapse them: any misprediction that does not result from a conflict and which the bimodal predictor can eliminate, we deem a *training misprediction*. Needless to say, this is only an approximation.

Wrong type of history. Mispredictions can also occur because the predictor tracks the wrong type of history for the branch in question: global instead of local, or vice-versa. Our measurements separate "true" wrong-history mispredictions from those merely caused by interference. True wrong-history misses occur when a non-conflict-miss/non-training-miss is correctly predicted using the other type of history. These mispredictions can potentially be completely eliminated by MAs, because it provides both types of history.

Needs both types of history. For some branches, neither type of history alone suffices; instead the branch *needs both* types of information simultaneously. We estimate the frequency of such mispredictions using an

MAs predictor, inferring that mispredictions which do not fall into the preceding categories and which MAs can eliminate are need-both mispredictions. The number of need-both mispredictions is always the same for both GAs and PAs.

Figure 4 shows the sequence of tests just described. Each dynamic branch can be categorized for both GAs and PAs by following the sequence of tests described until it predicts correctly, or falls through as a misprediction that could not be categorized. These *remaining* branches are either inherently difficult to predict, or need longer history.

Conflict and training mispredictions can also be categorized for MAs using an MAp predictor and a bimodal predictor.

3 Simulation and Benchmark Details

The next section compares GAs, PAs, and MAs using the taxonomy we have presented, but first we describe our simulation environment and benchmark suite.

3.1 Simulator

Our simulations use the instruction-level branch-predictor simulator from the University of Wisconsin's SimpleScalar toolkit v2.0 [1]. SimpleScalar provides a toolbox of simulation components—like a branch-predictor module and a statistics-gathering module—and several simulators, including an instruction-level branch-prediction simulator which we have augmented in various ways to obtain the measurements in this paper. The simulator interprets executables compiled by *gcc* version 2.6.3, targeting a portable, virtual instruction set (PISA) that most closely resembles MIPS IV [19]. Simulations do not model kernel behavior or context switches, instead performing operating-system calls by proxy, but otherwise all non-kernel behavior, including library code, is simulated.

Our results only report conditional-branch accuracies. Unconditional branches do not use the direction-prediction portion of the branch predictor. Their performance depends on the branch target buffer's (BTB) behavior, whose design we do not explore here. We therefore report prediction accuracies in terms of the percentage of conditional branches whose *direction* has been correctly predicted, regardless of whether they miss in the BTB.

Changes in conditional-branch prediction accuracy do not necessarily translate directly into corresponding changes in overall performance. That depends on the frequency of branches within the code, the degree of instruction-level parallelism, the average misprediction penalty, and other factors. Nevertheless, all other things being equal, better prediction improves performance. We restrict our measurements to prediction type and accuracy in order to focus on different predictors' ability to exploit predictability.

3.2 Benchmarks

Our evaluations use not only the SPECint95 benchmarks [27], but also 4 other primarily integer benchmarks and three SPECfp95 benchmarks. *Gnuchess* comes from the IBS benchmark suite [28] and is GNU's implementation chess. The benchmark sets the computer to play itself and deactivates the X interface. *Wolf* is the timberwolf circuit router and comes from Smith's Unix-Utils benchmark suite [24]. 1.7% of its instructions are floating-point operations. We were unable to simulate other members of the IBS or Unix-Utils suite due to difficulties compiling them for the SimpleScalar PISA or because they use the X library, which is not part of the SimpleScalar 2.0 toolkit.

Radiosity and *volrend* were chosen from the SPLASH2 suite [29] of parallel applications for shared memory because these two programs have significant misprediction rates and interesting behavior. We have compiled them for a single processor using the null macros. *Radiosity* computes the equilibrium distribution of light in a scene. and *volrend* renders a three-dimensional volume using a ray casting technique.

Finally, we include three SPECfp benchmarks, *tomcatv, mgrid*, and *hydro2d*. They all have tiny misprediction rates compared to the other applications, but these programs serve as verification that MAs does not harm the performance of easily-predicted programs.

	Warmup insts	Simulated insts	Conditional branch static sites	Conditional branch dynamic refs
go	925 M	1 B	5,331	112 M
m88ksim	0	1 B	968	162 M
gcc (cc1)	0	1.242 B	20,783	190 M
compress	1648 M	965 M	203	151 M
li (xlisp)	0	1 B	676	154 M
ijpeg	823 M	1 B	1,415	58 M
perl	600 M	1 B	614	129 M
vortex	2450 M	1 B	3,203	124 M
tomcatv	2950 M	1 B	186	130 M
mgrid	475 M	1 B	923	154 M
hydro2d	375 M	1 B	1,205	189 M
gnuchess	0	1 B	1,127	96 M
wolf	0	164 M	2,993	26 M
radiosity	300 M	1 B	183	92 M
volrend	125 M	1 B	660	70 M

Table 1: Benchmark summary. "Warmup insts" indicates the length of the preliminary phase of simulation, before statistics-gathering is activated. Also note that, for compress, we only simulate the compress phase.

Some benchmarks, like *compress*, have substantial initial phases in which they generate data (as in *compress*), read in data, or perform other actions that differ from the main body of the execution. Simulations produce substantially unrepresentative results if this initial phase comprises too much of the simulation [22]. For these programs, the simulator therefore begins gathering statistics only after the initial phase has passed. During this preliminary phase of simulation, branches are still presented to the predictor so that it can warm up. Benchmarks like *gcc* that have a minimal initial phase perform no simulator warmup.

All benchmarks are compiled using `gcc -O3 -funroll-loops` (`-O3` includes inlining). Most benchmarks are simulated for 1 billion instructions. The SPEC programs use "ref" inputs. Some benchmarks come with multiple reference inputs, in which case one has generally been chosen. For *go*, we chose a playing level of 50 and a 21x21 board with the `9stone21` input. For *m88ksim*, we used the `dhrystone` input; for *gcc*, `cccp.i`, which we simulate to completion. For *ijpeg* we use `vigo.ppm`, and for *perl*, the scrabble game. For *xlisp*, we ran the program with the 9-queens problem (the SPEC92 ref input). Compress repeats a compress-uncompress phase over and over; we simulate a single compress phase to completion.

We run *gnuchess* at a level of 10, and provide *wolf* with the supplied `cbd` input, which we run to completion. *Radiosity* uses the default parameters and the default "room" model, and *volrend* uses the largest provided input. The benchmarks are summarized in Table 1, including the simulator warmup interval, each program's static footprint (the number of branch sites exercised during execution), and the number of dynamic branches executed.

4 Using the Taxonomy to Compare Two-Level Predictors

In addition to comparing the overall misprediction rates of GAs, PAs, and alloyed (MAs) predictors, this section explores the behavioral differences among the different two-level organizations by breaking down mispredictions into categories using our taxonomy.

4.1 Performance with Equal PHT sizes

Comparing equal-sized PHTs and hence equal index sizes reveals the "indexing strength" of different possible bit combinations. To get the best comparison, we must examine the best possible GAs, PAs, and MAs predictors for a particular size. The brute-force method, simulating all possible combinations of global, local, and address bits for the desired index size, not only identifies the best configurations, but yields surfaces interesting in their own right. We mapped out these surfaces for *go, m88ksim, gcc*, and *compress*. Figures 5 and 6 present examples of these surfaces for *gcc* and *m88ksim* with a 32K-entry PHT.

A noteworthy feature of the surfaces is that using hill-climbing to find the best combination can fail due to local optima. One benefit of the taxonomy is therefore that understanding the source of mispredictions can make maneuvering through this non-monotonic space somewhat easier. The problem is pronounced for *m88ksim* when choosing a starting point with a small number of local history bits. Starting with 15 global-history bits and asking whether to replace some of them with other types of bits, hill-climbing leads to an index composed of 11 global-history bits, 2 local-history bits, and 2 address bits. The true best configurations, however, are 6 bits global and 9 bits local, and—a tie—no global, 14 bits local, and 1 bit of address. A similar effect occurs with *gcc* if hill-climbing uses an all-local-bit index as a starting point, but the effect is barely perceptible at this scale.

With these maps of the parameter space, we chose one configuration that gives the best overall performance; these choices appear in Table 2. In each case, the PAs and MAs predictors use an interference-free BHT so that equal-length indices can be compared without artifacts from BHT behavior. Figures 7–9 show both the overall misprediction rate—the bars' total height—and the breakdown of mispredictions for three PHT sizes: predictors with 32K counters, 4K counters, and 1K counters.

	GAs	PAs	MAs
32K entries	8g, 7a	14p, 1a	10g, 4p, 1a
4K entries	5g, 7a	10p, 2a	7g, 4p, 1a
1K entries	1g, 9a	10p, 0a	5g, 4p, 1a

Table 2: Predictor configurations used for equal-PHT comparison. "g" indicates the number of global-history bits, "p" local-history bits, and "a" address bits.

The results are strong. For every benchmark and each of the three PHT sizes, our alloying approach, MAs, outperforms GAs, usually by a substantial margin. MAs also outperforms PAs for most benchmarks. The exceptions are *m88ksim, xlisp, gnuchess*, and *tomcatv*. These benchmarks require substantial self-history, longer than that provided by MAs. This behavior can be inferred from the substantial wrong-history component of the GAs bar (the stippled segment): 47% of *xlisp*'s and 16.5% of tomcatv's GAs mispredictions with a 4k PHT, for example.

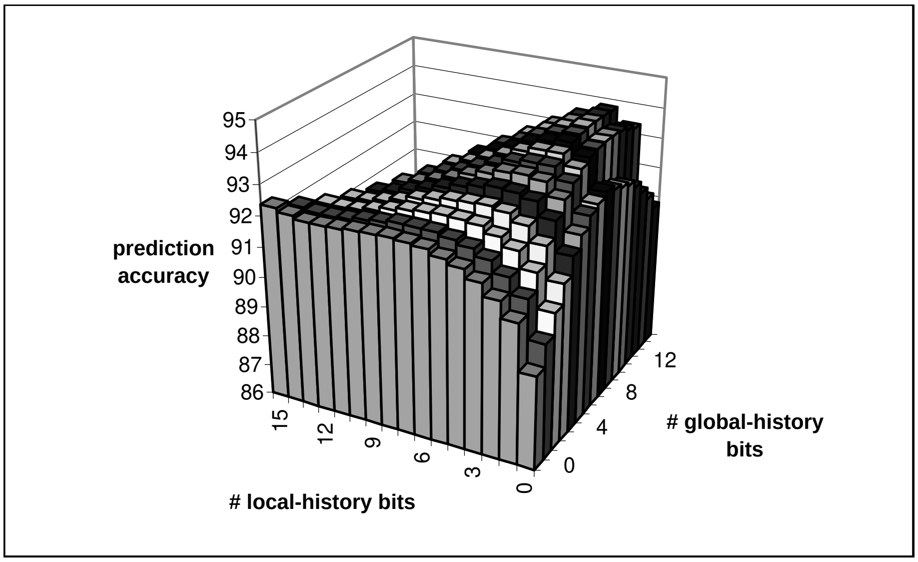

Figure 5: A surface showing, for gcc, the prediction accuracy of all the possible combinations of global-history, local-history, and address bits in a 15-bit index. The number of address bits can be found be subtracting the sum of global- and local-history bits from 15. Only indexes totaling 15 bits are simulated, so the back half of this surface (approaching the coordinates (15,12)) is absent, which is not evident from this perspective.

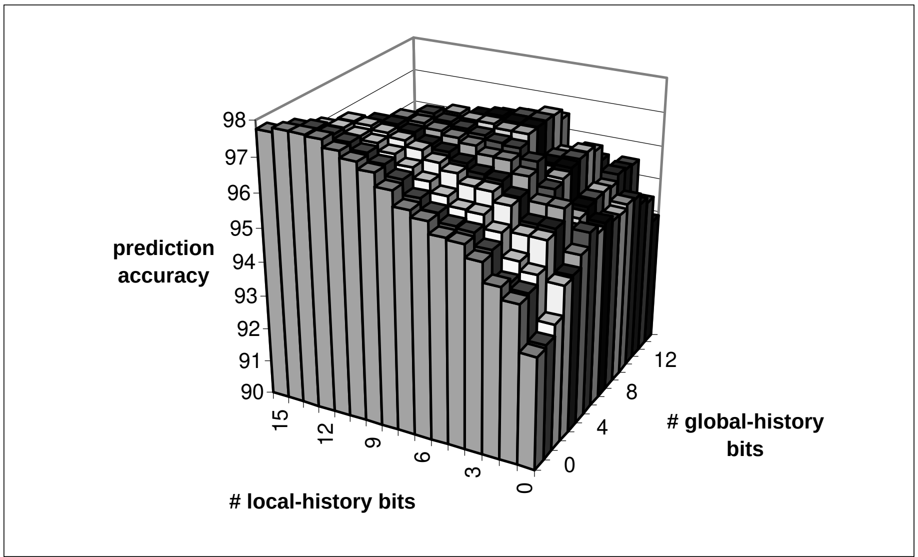

Figure 6: A surface showing, for m88ksim, the prediction accuracy of all the possible combinations of global-history, local-history, and address bits in a 15-bit index.

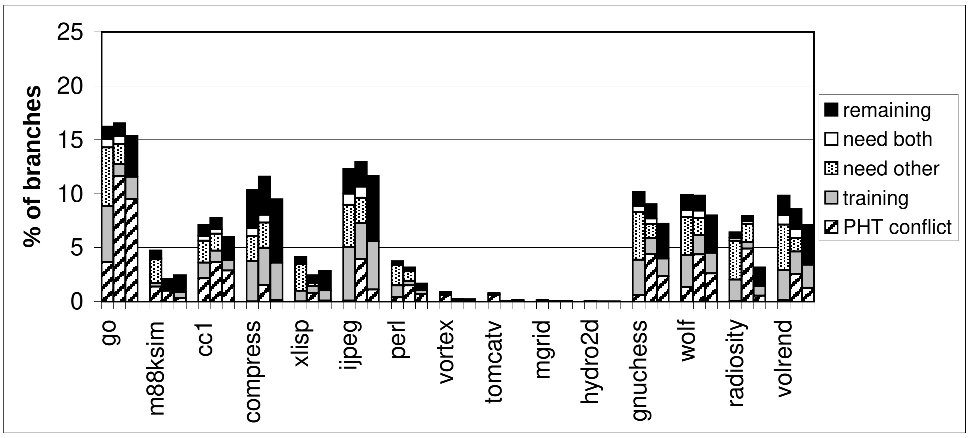

Figure 7: Relative performance of GAs, PAs, and MAs for a 32K-entry PHT and an interference-free BHT, including a breakdown of misprediction types. KEY: For each benchmark, the leftmost bar represents GAs, the center bar PAs, and the rightmost bar MAs.

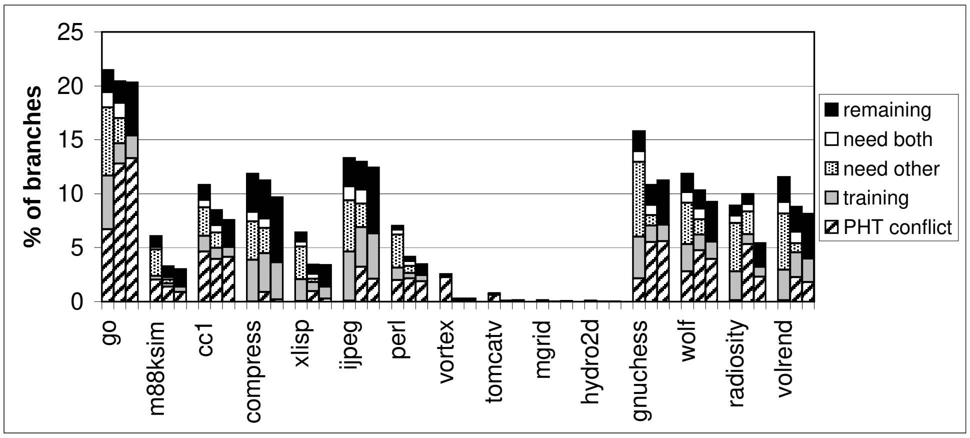

Figure 8: Relative performance of GAs, PAs, and MAs for a 4K-entry PHT and an interference-free BHT, including a breakdown of misprediction types.

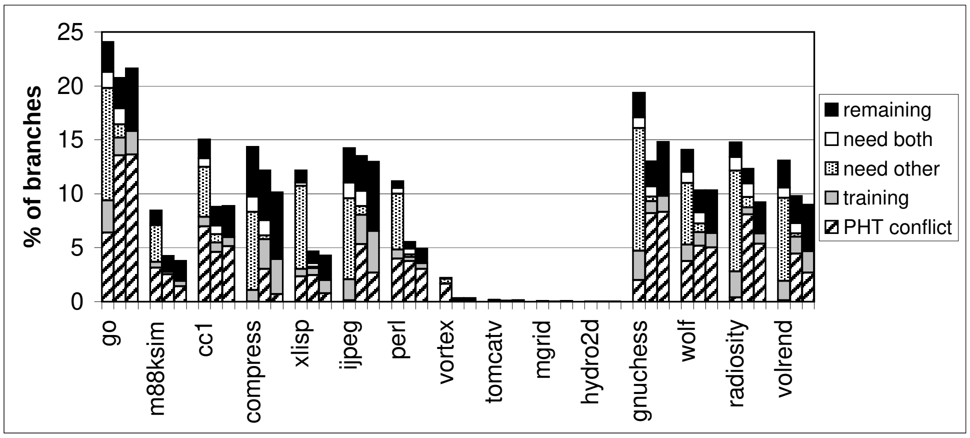

Figure 9: Relative performance of GAs, PAs, and MAs for a 1K-entry PHT and an interference-free BHT, including a breakdown of misprediction types.

	64 Kbits		8 Kbits		2 Kbits	
	GAs	PAs	GAs	PAs	GAs	PAs
go	5.14	6.99	5.26	0.54	10.23	-4.34
m88ksim	48.29	-15.94	50.34	8.18	55.03	10.91
cc1	15.82	22.59	30.13	10.78	40.91	-1.09
compress	8.31	18.26	18.45	13.92	29.42	16.72
xlisp	30.45	-18.63	46.73	0.49	64.76	8.38
ijpeg	5.09	9.79	6.49	4.25	8.98	3.98
perl	54.71	46.30	50.68	16.33	56.00	11.58
vortex	75.23	16.67	88.69	4.26	84.21	0.00
tomcatv	80.07	-133.33	79.58	-44.74	16.67	-35.25
mgrid	56.93	1.67	57.25	-5.36	16.90	-3.51
hydro2d	65.63	12.00	76.77	4.17	27.27	7.69
gnuchess	28.81	19.82	28.88	-3.63	23.56	-13.80
wolf	19.10	18.62	21.97	10.66	26.79	0.29
radiosity	50.47	60.06	38.86	45.48	37.71	25.33
volrend	27.56	16.66	29.67	7.58	31.39	8.25

Table 3: Percentage by which MAs's misprediction rate is better than GAs or PAs for same-sized PHTs.

The number of mispredictions that "need both" (the white segment of each bar) is small: 5–15% of mispredictions. Compared to GAs, the majority of benefit for MAs evidently stems from its ability to address wrong-history mispredictions, which are substantial for every benchmark under GAs. This suggests that many branches need at least a few bits of self-history. Global history does in principle include a branch's own history, but the global history's contents turn over very rapidly. For example, with an 8-bit global history (as used by our 32k-PHT GAs), the global history turns over so quickly that almost no branches can find any self history in the GBHR.

PAs tracks self-history, but cannot observe correlations among different branch sites, so most benchmarks also exhibit a substantial wrong-history component when using a PAs predictor. Exceptions are *m88ksim, xlisp*, and the three SPECfp benchmarks. For 1K counters, the global-history predictor only uses 1 bit of global history, so in this case few benchmarks indicate any substantial need for global history. When benchmarks have minimal need for global history, PAs performs well compared to GAs.

Another factor benefiting MAs when compared to PAs is that MAs almost always exhibits less PHT interference. (The only exception is *gcc* for smaller PHTs.) This is partly because our parameter search produced a PAs configuration using a minimal number of address bits: the benefit of additional local history bits outweighs the increased interference. On the other hand, PAs and MAs both usually suffer from more interference than GAs, because the best GAs configuration incorporates so many address bits.

Overall, MAs consistently performs well compared to conventional two-level predictors. It makes both types of history available all the time, and also reduces PHT interference compared to PAs. The degree to which MAs outperforms (or occasionally underperforms) GAs and PAs appears in Table 3. For example, with a 32K-entry PHT, MAs cuts the GAs misprediction rate by at least 25% in 10 of the benchmarks, and in 11 of the benchmarks with a 1K-entry PHT. Even for benchmarks which already have a high prediction accuracy, gains of this magnitude are significant.

4.2 Performance with Equal Overall Area

The preceding section provided strong results for MAs, but used an interference-free BHT and equal PHT sizes. This gave an answer to the question: what is the most productive way to compose a PHT index of some particular size?

MAs and PAs have a substantial advantage with an interference-free PHT, because they have extra information: n bits of local history for every branch. Figures 10–12 therefore repeat the above experiments, but for equal-size overall configurations.

The configurations used appear in Table 4. We found these by halving the PHT size for PAs and MAs in order to make space available for the BHT, and then exploring a number of BHT configurations, ranging from wide and short (many local-history bits and few BHT entries) to narrow and tall. The importance of BHT contention makes this a more important parameter than local-history length for both PAs and MAs, so programs generally prefer a tall BHT even though this means a very narrow local-history width. This occurs in spite of using a 4-way associative BHT. For 2 Kbits, both PAs and MAs actually use the same number of local-history bits. These configurations are a sharp contrast to those used above, where programs benefited from quite long local histories.

	GAs	PAs			MAs		
		index	BHT	PHT	index	BHT	PHT
64 Kbits	8g, 7a	8p, 6a	4K entries	16K entries	9g, 4p, 3a	8K entries	16K entries
8 Kbits	5g, 7a	4p, 7a	1K entries	2K entries	7g, 2p, 2a	2K entries	2K entries
2 Kbits	1g, 9a	2p, 7a	512 entries	512 entries	3g, 2p, 4a	512 entries	512 entries

Table 4: Predictor configurations used for equal-total-size comparison. BHTs are 4-way associative.

The impact of contention in a finite BHT can be seen not only in the different configurations we used, but in the substantial loss in prediction accuracy for MAs and PAs compared to Figures 7–9. (The GAs results are of course unchanged.) Although MAs loses some advantage compared to GAs, it remains a winner for most benchmarks, and especially at 64 Kbits. Table 5 summarizes the improvement in misprediction rates with MAs over GAs and PAs.

These results show how much further work on reducing BHT contention, like [2, 12], can help push performance toward those in Section 4.1.

4.3 Dynamically Configurable Indexes

Our simulations to compare GAs, PAs, and MAs for the above results used the one best overall configuration. Individual benchmarks, however, have varying preferences; indeed, sometimes contradictory preferences. Even though MAs already performs well, this suggests that an ability to dynamically configure the PHT index's composition—a small change—would be helpful. Then on start-up, each program can set the index to the most beneficial bits from the available global history, local history, and address bits.

As an example of the impact, we consider the two benchmarks, *gcc* and *m88ksim*, for which we presented MAs-performance surfaces in Figures 5 and 6. They prefer dramatically different MAs configurations. *M88ksim* does best with an almost entirely local-history index, while *gcc* does best with a primarily global-history index. In each case, the best configuration for one gives poor performance for the other. Table 6 compares these two cases.

These points are, of course, extreme. Yet most benchmarks prefer an MAs configuration in the general vicinity of the one shown for *gcc*, and the overall, "compromise" configuration used in Section 4.1 is similar

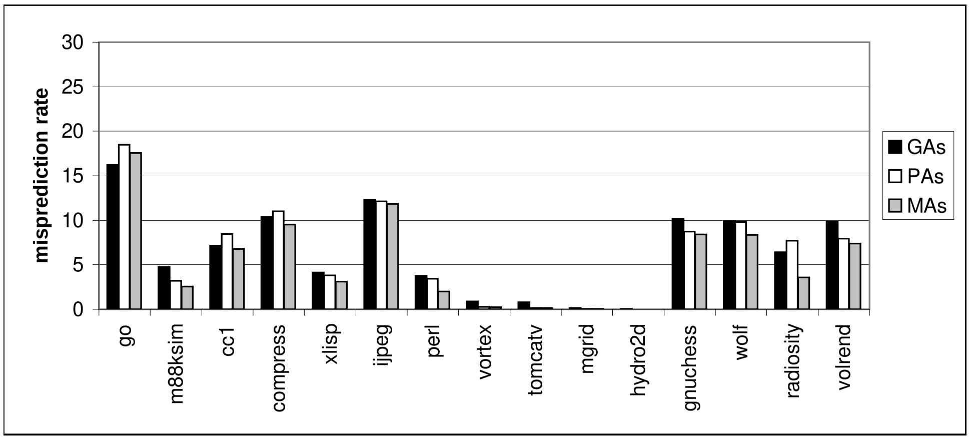

Figure 10: Relative performance of GAs, PAs, and MAs for a 64 Kbits total size, including a finite BHT.

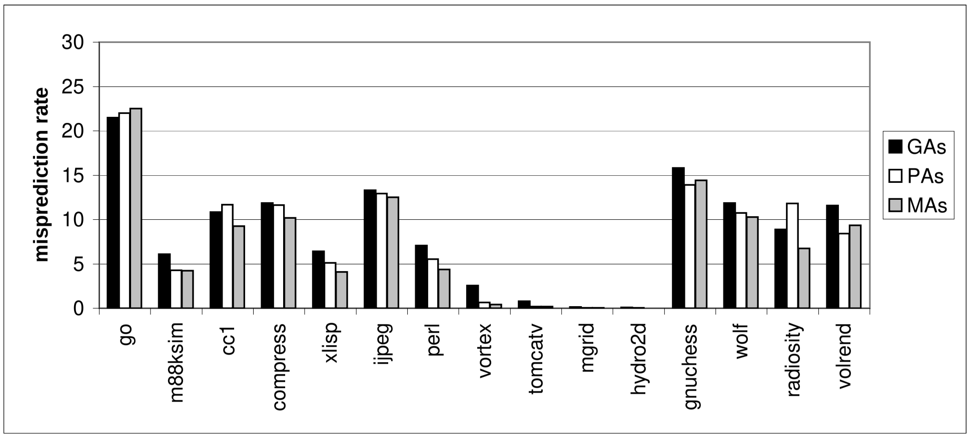

Figure 11: Relative performance of GAs, PAs, and MAs for 8 Kbits total size, including a finite BHT.

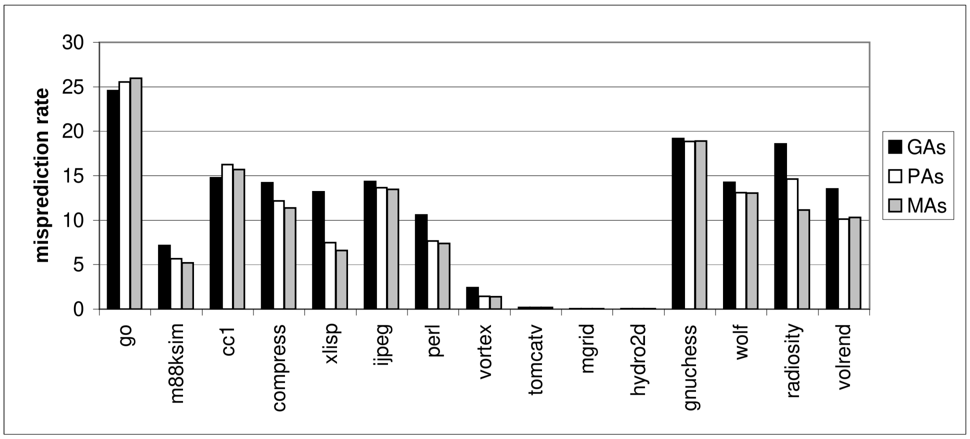

Figure 12: Relative performance of GAs, PAs, and MAs for 2 Kbits total size, including a finite BHT.

	64 Kbits		8 Kbits		2 Kbits	
	GAs	PAs	GAs	PAs	GAs	PAs
go	-8.02	5.09	-4.78	-2.30	-5.60	-1.49
m88ksim	46.04	19.38	30.60	1.26	27.19	8.35
cc1	5.04	19.96	14.89	20.76	-6.42	3.53
compress	7.91	13.31	14.15	12.32	19.75	6.30
xlisp	24.88	18.28	36.03	19.59	49.95	11.52
ijpeg	3.86	2.53	5.94	3.16	6.31	1.55
perl	46.96	41.89	37.81	20.66	30.59	4.07
vortex	72.12	8.33	84.42	40.36	42.76	5.70
tomcatv	80.07	-11.03	78.47	-4.82	11.62	0.57
mgrid	56.93	-5.36	56.52	-1.69	-3.28	0.00
hydro2d	64.06	0.00	76.77	4.17	20.00	0.00
gnuchess	17.58	4.16	8.70	-3.82	1.44	-0.42
wolf	15.65	14.65	13.31	4.53	8.31	0.11
radiosity	44.28	53.53	24.27	42.83	39.99	23.84
volrend	24.99	6.93	19.32	-10.94	23.88	-1.85

Table 5: Percentage by which MAs's misprediction rate is better (or sometimes worse) than GAs or PAs for equal-size configurations.

	misprediction rates			Δ rates	
	m88 best 14p, 1a	gcc best 2p, 10g, 3a	compromise 4p, 10g, 1a	Δ(best–worst)	Δ(best–compromise)
m88ksim	2.12	3.32	2.46	56.60%	16.04%
gcc	7.75	5.7	6	35.96%	5.26%

Table 6: Alternative configurations for gcc and m88ksim and their performance. "Compromise" represents the best overall organization we found.

to *gcc*'s best. Both benchmarks do fairly well with that compromise (also shown in Table 6). But a slight change in the benchmark suite could easily shift the compromise configuration to actually be the best one for *gcc*, which is so poor for *m88ksim*. Some configurations in the immediate vicinity of *gcc*'s best actually cause even worse performance for *m88ksim*: for example, using 13 bits of global history makes *m88ksim*'s misprediction rate nearly double again.

Such a scenario could occur, for example, if *m88ksim* were not in the benchmark suite. This of course argues for care in selecting benchmarks, but no suite can be all-inclusive. This effect is also not restricted to alloyed indices. Benchmarks exhibit contradictory preferences for PAs and especially for GAs. In the latter case, programs which suffer from little PHT interference prefer mostly global history in the index, while others prefer many fewer global bits and want address bits instead.

Allowing each program to determine the index's composition could therefore yield large benefits for programs that behave differently than the benchmark suite used in evaluating the processor design. Choosing a best configuration entails testing the program's prediction accuracy on the possible alternatives, which is a small number for GAs and PAs. A configurable GAs or PAs would then provide full history, but allow the program to choose how many history bits to mask out and replace with address bits.

Configuring an MAs predictor involves a wider range of choices, but these can all be tested in parallel. A configurable organization would perhaps include a BHT of moderate width (4–8 bits), a full length GBHR, and the ability to choose which bits to include. Some programs might even choose to make MAs behave strictly like GAs or PAs.

5 Alloyed History vs. Hybrid Prediction in High-Performance Processors

The breakdown of misprediction types in Section 4.1, as well as MAs's success compared to GAs and PAs, suggest that many programs need a predictor that includes both local and global history. Since only a small fraction of branches benefit from having both types available simultaneously, hybrid predictors that combine both a global-history and a local-history component can eliminate many of the same mispredictions as MAs, and can possibly do a better job than MAs of attacking the PHT-conflict problem. With a static selector like the one proposed by Grunwald, Lindsay, and Zorn [9], branches only update their assigned component, with a substantial corresponding decrease in interference. With a dynamic selector like those proposed by McFarling [15] and Chang, Hao, and Patt [4], if one component suffers from conflicts but the other does not, the hybrid predictor can select the well-behaved component.

An MAs predictor looks almost like a conventional two-level predictor, but by alloying local and global history, provides pseudo-hybrid functionality. This section examines the differences among an MAs predictor, a two-component hybrid predictor with dynamic selection, and a two-component predictor with static selection. The dynamic selector is itself a two-level predictor using global history [4].

5.1 Static vs. Dynamic Choice

The wrong-history category from our taxonomy only indicates whether programs as a whole benefit from having both types of history available. That data does not show whether branch sites within a program have different needs but particular branch sites consistently use the same type of history, or whether particular branch sites dynamically vary the type of history they use. In the latter case, static hybrid suffers. It must assign each branch site to one or the other component, and therefore mispredicts when that branch needs to use the other. Static selection also suffers when the training data is not sufficiently representative. Dynamic hybrid can learn how to steer a branch back and forth, and MAs also deals well with such behavior.

A better understanding of how to design hybrid predictors requires actually measuring how often wrong-history mispredictions occur at each branch site. Figure 13 shows X-Y scatter graphs for an 8 Kbit predictor with an interference-free BHT for *gcc* and *m88ksim*. Each dot represents one branch site; only branches which executed 100,000 times or more are plotted. The graph's x-coordinate gives the percentage of time that branch needs a global predictor; its y-coordinate gives the percentage of time that branch needs a local predictor. If branches consistently use one component, they will lie on one or the other axis. The more mass that lies in the middle of the graph, the more of a disadvantage static-hybrid prediction should experience. *Gcc* has many branches which use both local and global history; *m88ksim* fewer. In both cases, a substantial amount of mass lies at the origin, representing strongly-biased branches and other branches that are easily predicted by either structure (in the absence of interference). Branches that are mispredicted by both structures also lie at the origin.

5.2 Performance Comparison

Here we only consider the largest of the sizes discussed so far, 64 Kbits, because high-performance processors are likely to devote a substantial amount of resources toward obtaining good branch-prediction

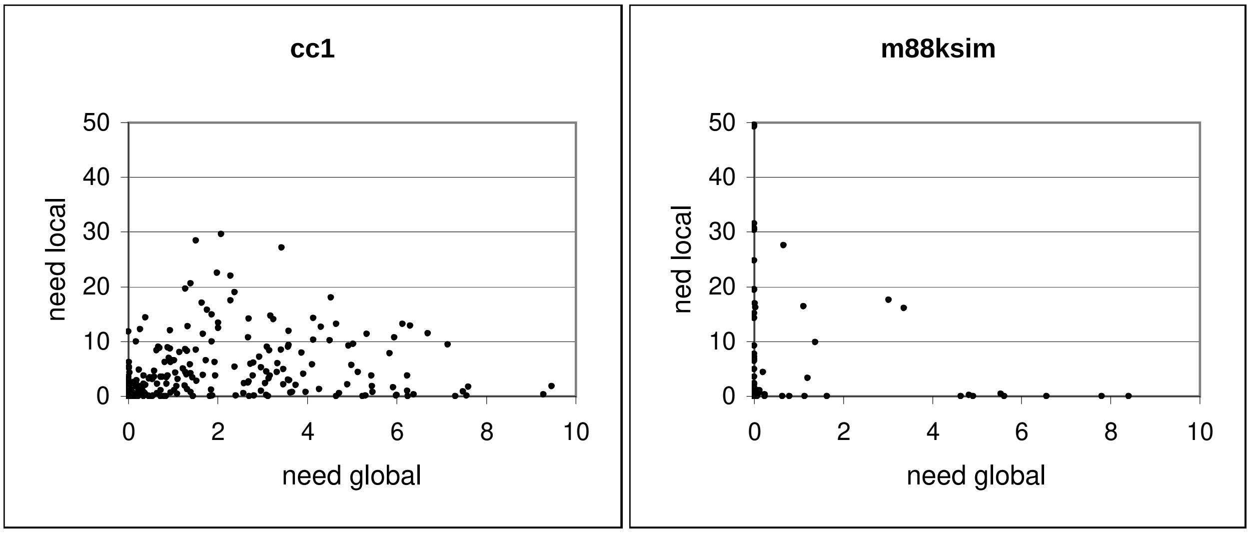

Figure 13: X-Y scatter plots showing, for each branch site, the percentage of time that branch needs a global predictor or a local predictor.

accuracy. Furthermore, as we show in the next section, at smaller sizes like 8 Kbits, a hybrid organization's components are so small that even a stand-alone GAs or PAs is sometimes better.

As before, to get a fair comparison, we want to compare the best configuration for each organization. For dynamic hybrid, we started with the components as close as possible to those used in Section 4.2, but added a selector. We then explored a wide range of configurations surrounding these starting points. The best configuration for a particular application varied widely, but for most programs, the best allocation of resources among the components gives more area to the global-history component than to the selector or the local-history component.

For static hybrid, we again started with the best components from Section 4.2, but deemed that the two components should use the same number of bits. This permitted a smaller search for the best configuration. We trained and simulated using the same input, in order to eliminate unrepresentative training data as a problem.

	GAs		PAs			selector	
	index	PHT	index	BHT	PHT	index	PHT
64 Kbits	7g, 7a	16K entries	8p, 4a	1K entries	4K entries	6g, 7a	8K entries
64 Kbits	7g, 7a	16K entries	13p, 0a	1K entries	8K entries	na	

Table 7: Predictor configurations used for 64 Kbit dynamic and static hybrid predictors.

Figures 14 gives performance figures for the three choices, and Table 8 shows how much better or worse MAs's misprediction rate is. The hybrid configurations we used for these results appear in Table 7. The MAs configurations were taken directly from Section 4.2.

We were unable to make static-hybrid outperform dynamic-hybrid except for *go*, where the reduced interference in the static organization is probably the dominant effect. Our results differ from those reported by Grunwald *et al.* [9], which find that static selection has a significant advantage for smaller sizes, where it can substantially reduce interference compared to dynamic selection.

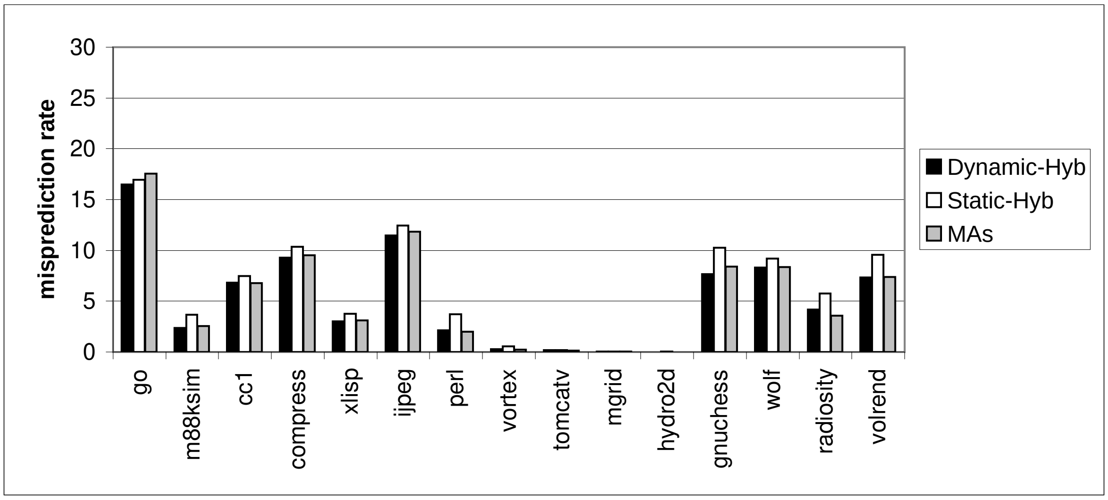

Figure 14: Relative performance of dynamic hybrid, static hybrid, and MAs for 64Kbits total area.

	Dynamic	Static
go	-6.24	-3.35
m88ksim	-8.18	29.99
cc1	0.54	9.35
compress	-2.55	7.86
xlisp	-3.14	17.48
ijpeg	-3.05	4.86
perl	6.66	46.19
vortex	6.92	54.85
tomcatv	3.01	17.86
mgrid	-5.36	13.24
hydro2d	-4.55	23.33
gnuchess	-9.25	18.46
wolf	-0.76	8.98
radiosity	14.53	37.70
volrend	-0.35	22.94

Table 8: Percentage of mispredictions that a 64 Kbit MAs predictor removes compared to dynamic and static hybrid predictors of the same size.

If correct, our results suggest that individual branches are indeed alternating between using both global and local history. These suffer in the static organization, where such alternation is not possible. We have not measured the frequency of this alternation: some branches might change between history types just once.

Dynamic hybrid can mispredict even when one of the underlying components has the right outcome, if the selector *mis-chooses*. Mis-choices are a substantial fraction, even up to 50%, of mispredictions. When MAs outperforms dynamic selection, we find that it usually does so because it correctly predicts some of the branches where mis-choices occur in the hybrid scheme. MAs usually outperforms static selection, but is roughly a tie compared to dynamic selection: sometimes superior, sometimes inferior, usually by small margins in both cases.

6 Alloyed History vs. Hybrid Prediction in Embedded Environments

Not all processors designed today can afford to devote a large area to the branch predictor. For example, power constraints may dictate a smaller chip size, and processors for embedded environments are generally both space- and power-constrained. These environments still require the best branch prediction available for the available size in order to maximize performance. Better branch prediction also reduces power consumption by reducing power expended on mis-speculated computation [14].

	GAs		PAs			selector	
	index	PHT	index	BHT	PHT	index	PHT
dynamic	4g, 7a	2K entries	2p, 7a	512 entries	512 entries	3g, 7a	1K entries
static	4g, 7a	2K entries	2p, 8a	1k entries	1k entries	na	

Table 9: Predictor configurations used for 8 Kbit dynamic and static hybrid predictors.

For these reasons, we revisit the 8 Kbit size and compare all the organizations discussed so far: GAs, PAs, dynamic and static hybrid, and MAs. Again, we use the best configurations we could find for dynamic and static hybrid; these appear in Table 9. Figure 15 presents the results; it just adds the two hybrid predictors (the striped and stippled bars) to the data seen in Figure 11. Table 10 presents the fraction of mispredictions which MAs removes compared to other predictors.

Just as with 64 Kbits, dynamic hybrid usually outperforms static hybrid. *Go* and *gcc* are exceptions. But GAs and PAs sometimes outperform dynamic hybrid. The hybrid predictors must use such small components—dynamic hybrid must fit three structures into the available area—that these components are sometimes hobbled.

MAs can devote all its resources to a single structure while still providing both global and local history, so MAs performs consistently well. We already showed that it does well against GAs and PAs. It also outperforms dynamic and static hybrid for every benchmark except *go*.

Our alloyed-history organization, MAs, therefore shows excellent performance compared against other two-level predictors for a range of sizes and against hybrid predictors for smaller sizes. Compared to a large, 64 Kbit dynamic-hybrid predictor, MAs provides competitive performance. Furthermore, MAs performs consistently well over a wide range of benchmarks.

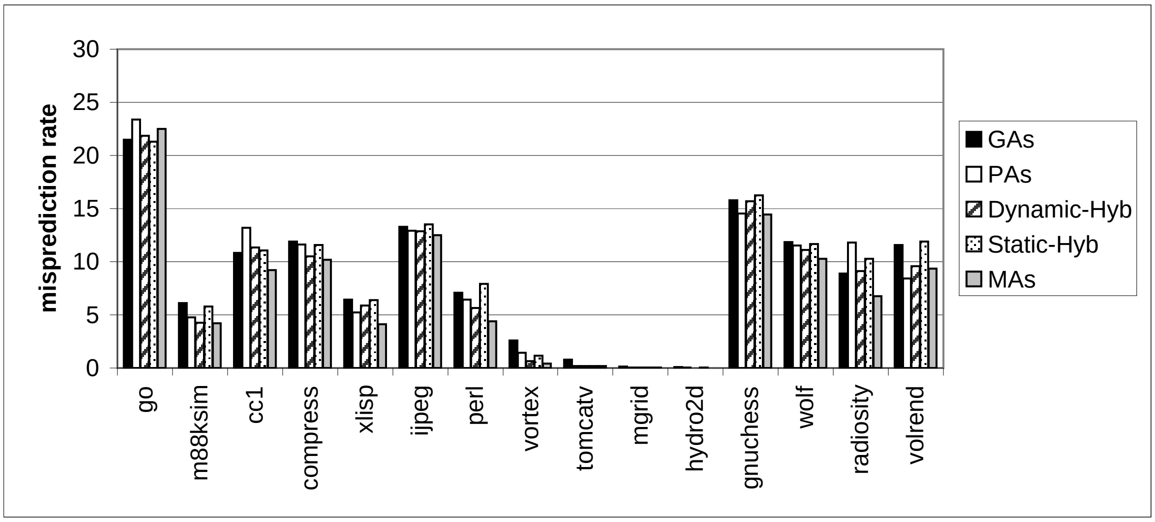

Figure 15: Relative performance of GAs, PAs, dynamic hybrid, static hybrid, and MAs for 8Kbits total area.

7 Related Work

Pan *et al.* [18] observed the tradeoff between including history and address bits in the PHT index, and introduced GAs. Yeh and Patt [30] further examined the balance between history and address bits in GAs and PAs, finding global-history schemes suffer more from interference and are more strongly sensitive to both history length and number of PHTs (address length) than are local-history schemes.

Sechrest *et al.* [21] also observed the value of sacrificing some history bits for address bits, especially for global-history predictors. They further found that whether two shorter strings of history bits and address bits are concatenated, as in GAs/PAs, or full-index-length strings are xor'd, as in *gshare* [15], has minimal effect on performance.

The above papers explore the use of address bits to provide a kind of associativity in the PHT and therefore reduce destructive interference. Other researchers have described a variety of alternative techniques for reducing interference, the simplest of which is the aforementioned gshare. Sprangle *et al.* [25] measure the substantial harm caused by PHT interference and describe an *agree* predictor, which reinterprets the 2-bit counter's meaning as "agree with static prediction" or "disagree with static prediction". Michaud *et al.* [16] measure compulsory, capacity, and conflict aliasing and describe a *skewed* predictor, in which the branches simultaneously exist in multiple PHTs, and each PHT is indexed using a different hash function. A selector chooses which hash function to use. Lee *et al.* [13] observe that conflicting substreams may themselves by strongly biased, just biased in opposite directions. They describe a *bi-mode* predictor (not related to bimodal) that uses a meta-predictor to separate branch substreams of opposite bias into two PHTs. Because weakly biased branches still behave poorly under this approach, Eden and Mudge [5] follow this up with the YAGS predictor, which tries to learn when branches disagree with the predicted bias. Chang *et al.* describe a somewhat different approach: *branch classification*, which identifies highly-biased (either taken or not-taken) branches, which may both cause and suffer from interference. They describe a mechanism to filter these branches out of the predictor reference stream.

Classification also reduces BHT interference. Other work to reduce BHT interference includes instruction reordering [2], and working set analysis [12], which identifies concurrently active branches and, using an approach analogous to register coloring, tries to ensure they use different BHT entries.

Yet other work has focused on characterizing why mispredictions happen in two-level predictors. Talcott

	Two-Level		Hybrid	
	GAs	PAs	Dynamic	Static
go	-4.78	-2.30	-3.00	-5.59
m88ksim	30.60	1.26	0.54	26.90
cc1	14.89	20.76	18.48	16.67
compress	14.15	12.32	2.97	12.03
xlisp	36.03	19.59	30.29	35.75
ijpeg	5.94	3.16	2.64	7.50
perl	37.81	20.66	22.20	44.56
vortex	84.42	40.36	37.77	65.16
tomcatv	78.47	-4.82	6.95	12.56
mgrid	56.52	-1.69	0.00	0.00
hydro2d	76.77	4.17	0.00	63.49
gnuchess	8.70	-3.82	8.12	11.14
wolf	13.31	4.53	7.33	11.86
radiosity	24.27	42.83	26.20	34.49
volrend	19.32	-10.94	2.70	21.55

Table 10: Percentage of mispredictions that MAs removes compared to other 8 Kbit predictors.

et al. [26], Sprangle *et al.* [25], Young *et al.* [31], and Chang *et al.* [3] characterize PHT interference, and show that while both significant amounts of both constructive and destructive interference occur, the destructive interference consistently dominates. Young *et al.* also show, as does Nair [17], that using *path histories* improves performance: recording branch addresses instead of just the branch outcomes recorded by conventional branch histories can better separate branch substreams and reduce harmful interference.

Most recently, Evers *et al.* [8] focused on the correlation characteristics of branches, and found that global history strongly benefits a substantial fraction of branches within programs. Nevertheless, for a given prediction, most of the global history goes unused—adding to interference—and frequently the most useful branch outcomes have already been forced out of the global history. In a related vein, Sazeides and Smith explored how both data and control predictability propagates through the flow of a program in [20]. Emer and Gloy [6] used genetic-like mutations to grow predictors that merge information in a variety of ways.

Some branches within a program benefit more strongly from global history, while others benefit more strongly from local history. so McFarling suggests in [15] a structure that uses two components, with a bimodal, dynamic selector that chooses between them. Chang *et al.* [4] find that the most beneficial components are both two-level predictors: one using global history, the other, local history. They also show that a two-level organization using branch history as the selector outperforms a bimodal selector. Evers *et al.* [7] extend the two-component predictor by proposing a *multi-hybrid* predictor. This organization includes static-prediction components and a dynamic selector design that steers highly biased branches to these simple predictors. The multi-hybrid predictor also includes a bimodal predictor. Most recently, Grunwald *et al.*. [9] have proposed eliminating the selector in favor of a static selection mechanism, an approach we include in this paper's evaluations.

8 Conclusions and Future Work

A great deal of prior branch prediction work has focused on ways to improve two-level predictors using either local or global history, and some recent work has explored ways to combine multiple two-level predictors in one structure and how to choose which component to believe. This paper presents a taxonomy of misprediction types to better understand how to further improve branch prediction design. Our results show that in addition to the widely recognized problems of training time and interference in the pattern history table, *wrong-history* mispredictions are a substantial problem. These occur when a branch requires one type of history (global or local) but the predictor provides the other type. Not just programs as a whole, but also individual branches, suffer from wrong-history mispredictions. Hybrid predictors can attack these categories, especially wrong-history mispredictions, but the selection mechanism can be a further source of problems. In particular, static selection, which assigns branch sites to use one or the other component for the duration of the program, necessarily encounters mispredictions when individual branches suffer from wrong history.

Using these results, we demonstrate the benefits of an *alloyed* predictor that merges local and global history together in a single PHT index. Although such an organization is an incremental change to existing two-level designs, it makes both types of history available all the time. This attacks wrong-history mispredictions, as well as a further category of mispredictions in which branches need *both* types of history simultaneously. Alloying can also reduce PHT interference by using a richer mixture of bits.

Our proposed alloyed predictor, MAs, performs better than solo, two-level predictors of different sizes for most benchmarks. MAs also performs well against hybrid predictors, especially at smaller sizes where the components in a hybrid organization become too small. This is especially interesting for space-constrained environments like embedded processors, where better branch prediction not only helps performance, but reduces power consumption by reducing wasted computation on mis-speculated paths. Overall, our results show MAs to be an approach that yields robust results for a wide range of benchmarks and predictor sizes.

Finally, we presented some preliminary data suggesting that a simple degree of configurability—allowing programs to dictate which bits comprise the predictor index—can improve the performance of all types of predictors, and MAs in particular.

There are several promising avenues for further research. Because MAs is a two-level organization like the GAs and PAs components currently considered in hybrid predictors, MAs might contribute the benefits of alloying when included in a hybrid predictor. In particular, it might work well in a static hybrid organization by helping individual branches that otherwise exhibit wrong-history mispredictions, while dispensing with the need for dynamic selection.

Our hybrid-prediction results also suggest there still exists room for improving both MAs and hybrid prediction by better understanding when simpler two-level components fail and when selection mechanisms fail. Further refinements of branch misprediction categories will help this effort, too.

MAs is only one type of alloyed organization. Just as GAs and PAs can be improved by incorporating techniques for reducing interference, MAs can be, too. The mechanism for composing the alloyed index is a further interesting subject. Researchers have proposed techniques like xor'ing history and address bits [15] that might benefit alloyed predictors as well.

References

[1] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[2] B. Calder and D. Grunwald. Reducing branch costs via branch alignment. In *Proc. ASPLOS-VI*, Oct. 1994.

[3] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *Proc. PACT*, pages 48–57, Oct. 1996.

[4] P.-Y. Chang, E. Hao, and Y. N. Patt. Alternative implementations of hybrid branch predictors. In *Proc. Micro-28*, pages 252–57, Dec. 1995.

[5] A. N. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proc. Micro-31*, Dec. 1998.

[6] J. Emer and N. Gloy. A language for describing predictors and its application to automatic synthesis. In *Proc. ISCA-24*, pages 304–14, June 1997.

[7] M. Evers, P.-Y. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *Proc. ISCA-23*, pages 3–11, May 1996.

[8] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt. An analysis of correlation and predictability: What makes two-level branch predictors work. In *Proc. ISCA-25*, pages 52–61, June 1998.

[9] D. Grunwald, D. Lindsay, and B. Zorn. Static methods in hybrid branch prediction. In *Proc. PACT*, Oct. 1998.

[10] N. P. Jouppi and P. Ranganathan. The relative importance of memory latency, bandwidth, and branch limits to performance. In *The Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, June 1997. http://ayer.CS.Berkeley.EDU/isca97-workshop.

[11] T. Juan, S. Sanjeevan, and J. J. Navarro. Dynamic history-length fitting: A third level of adaptivity for branch prediction. In *Proc. ISCA-25*, pages 156–66, June 1998.

[12] S. Kim and G. Tyson. Analyzing the working set characteristics of branch execution. In *Proc. Micro-31*, Dec. 1998.

[13] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge. The bi-mode branch predictor. In *Proc. Micro-30*, pages 4–13, Dec. 1997.

[14] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. In *Proc. ISCA-25*, pages 132–41, June 1998.

[15] S. McFarling. Combining branch predictors. Tech. Note TN-36, DEC WRL, June 1993.

[16] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proc. ISCA-24*, pages 292–303, June 1997.

[17] R. Nair. Dynamic path-based correlation. In *Proc. Micro-28*, pages 15–23, Dec. 1995.

[18] S.-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proc. ASPLOS-V*, pages 76–84, Oct. 1992.

[19] C. Price. *MIPS IV Instruction Set, Revision 3.1*. MIPS Technologies, Inc., Mountain View, CA, Jan. 1995.

[20] Y. Sazeides and J. E. Smith. Modeling program predictability. In *Proc. ISCA-25*, pages 73–84, June 1998.

[21] S. Sechrest, C.-C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proc. ISCA-23*, pages 22–32, May 1995.

[22] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. A quantitative evaluation of branch prediction's impact on instruction-window size and cache size. *IEEE Trans. Computers*. To appear.

[23] J. E. Smith. A study of branch prediction strategies. In *Proc. ISCA-8*, pages 135–48, May 1981.

[24] M. D. Smith. *Support for Speculative Execution in High-Performance Processors*. PhD thesis, Stanford University, Nov. 1992.

[25] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. In *Proc. ISCA-24*, pages 284–91, June 1997.

[26] A. R. Talcott, M. Nemirovsky, and R. C. Wood. The influence of branch prediction table interference on branch prediction scheme performance. In *Proc. PACT*, pages 89–96, June 1995.

[27] The Standard Performance Evaluation Corporation. WWW Site. http://www.specbench.org, Dec. 1996.

[28] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proc. ISCA-22*, pages 345–56, June 1995.

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. ISCA-22*, pages 24–36, June 1995.

[30] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proc. ISCA-20*, pages 257–66, May 1993.

[31] C. Young, N. Gloy, and M. D. Smith. A comparative analysis of schemes for correlated branch prediction. In *Proc. ISCA-22*, pages 276–86, June 1995.