Alloyed Global and Local Branch History: A Robust Solution to Wrong-History Mispredictions

Princeton Dept. of Computer Science Technical Report
TR-606-99

Kevin Skadron
Dept. of Computer Science
University of Virginia

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

Abstract

The need for accurate conditional-branch prediction is well known: mispredictions waste large numbers of cycles, inhibit out-of-order execution, and waste power on mis-speculated computation. Prior work on branch-predictor organization has focused mainly on how to reduce conflicts in the branch-predictor structures, while relatively little work has explored other causes of mispredictions. Some prior work has identified other categories of mispredictions, but this paper organizes these categories into a broad taxonomy of misprediction types. Using the taxonomy, this paper goes on to show that other categories—especially *wrong-history mispredictions*—are often *more* important than conflicts. This is true even if just a very simple conflict-reduction technique is used.

Wrong-history mispredictions arise because current two-level, history-based predictors provide only global or only local history. Their contribution to the overall misprediction rate is substantial because most programs have some branches that require global history and also some that require local history. If only one or the other type of history is available, many branches are therefore penalized. For SPECint95 programs using a global-history predictor, wrong-history mispredictions account for 35–50% of the total misprediction rate. By comparison, conflicts only account for 15-20%. Hybrid predictors are one proposed solution; they use both a global-history and a local-history component. Unfortunately, hybrid predictors only work well with large hardware budgets, because they must subdivide the available area into subcomponents.

Based on these observations, this paper proposes *alloying* local and global history together in a two-level branch predictor structure. This is a simple but superior technique for making global and local history simultaneously available and eliminating wrong-history mispredictions. Unlike hybrid prediction, alloying gives robust performance for branch-predictor hardware budgets ranging from very large to very small. Alloying also consistently outperforms other two-level organizations. In fact, a small alloyed predictor often performs as well as a much larger global-history predictor.

I. Introduction

The question of how better to predict the direction of conditional branches has received intense study in recent years. Two-level [19], [32] and hybrid [16] predictors, which explicitly track prior branch history, have received special attention. Most of this attention examines how to reduce aliasing errors (*conflict mispredictions*), which arise when unrelated branches happen to collide in a particular branch-predictor entry and overwrite each other's state. Conflicts are undeniably important, but a wealth of excellent techniques have been developed to reduce these destructive conflicts in the pattern history table (PHT)[1] of two-level predictors [5], [14], [17], [21], [27]. Even *without* using aggressive anti-aliasing techniques, conflicts only account for 15–20% of mispredictions in global-history predictors and 40–50% in local-history predictors. Naturally, these fractions are much smaller when aggressive conflict-reduction techniques are applied. A complete elimination of conflicts therefore leaves many or most mispredictions unsolved.

Further reductions in conflict mispredictions are indeed becoming difficult, and prediction accuracies still lie only in the 90–97% range. It might seem that misprediction rates of 3–10% should be negligible, but each misprediction results in many wasted cycles: a minimum of 7 wasted cycles in the Alpha 21264 [10], and 11 wasted cycles in the Pentium Pro [9] and its successors. The average misprediction penalty is larger still—10 to 20 cycles—because branches in out-of-order processors like these can spend an arbitrary time in the instruction window waiting to issue. Furthermore, in multi-issue processors, each of these cycles represents a wasted opportunity to issue not one, but possibly several instructions. Even with the ability to issue instructions out of order, the instruction window can rarely expose sufficient ILP to hide such long penalties. Indeed, Jouppi and Ranganathan [11] predict that branch prediction will be the most important bottleneck in processors by 2010, worse even than stalls from the memory system.

This paper therefore looks beyond conflict mispredictions and organizes a number of misprediction types into a *taxonomy* to help characterize their relative importance. The paper then goes on to show that some other misprediction categories are often more important than conflicts, and that *alloyed branch prediction* is an attractive way to attack the especially important category of *wrong-history mispredictions*.

Wrong-history mispredictions. This type of misprediction arises in two-level organizations because current designs track either *global* history (recording outcomes from different branches in a single shared history register) or *local* history (tracking history separately for each branch using a dedicated table), but not both. Global history can expose correlation among branches, while local history is well suited for branches that follow a consistent pattern. Unfortunately, most programs have some branches that do well with global history *and* some branches that do well with local

[1] The PHT is the table of saturating two-bit counters used by most predictor organizations. Different organizations assign branches or branch streams to these two-bit counters differently.

history. If the branch predictor only tracks one or the other, some branches therefore find that the predictor provides the wrong type of history.[2] Evers *et al.* showed this to be important in [7]. Our measurements find that these wrong-history mispredictions are especially severe in global-history predictors, comprising 35–50% of the total misprediction rate.

Hybrid predictors [16] are one way to attack this wrong-history problem. Hybrid predictors combine two or more prediction components, with some way to choose which component to use for each dynamic branch encountered. If one component is a global-history predictor and the other is a local-history predictor, both types of history are now available [3]. This reduces the wrong-history problem if the selection mechanism does an effective job of choosing which component to use for each branch. The selector, however, may itself be a large prediction structure, and in fact seems to perform best if it is a two-level predictor [3], [13]. Figure 1 presents a high-level schematic of a hybrid predictor.

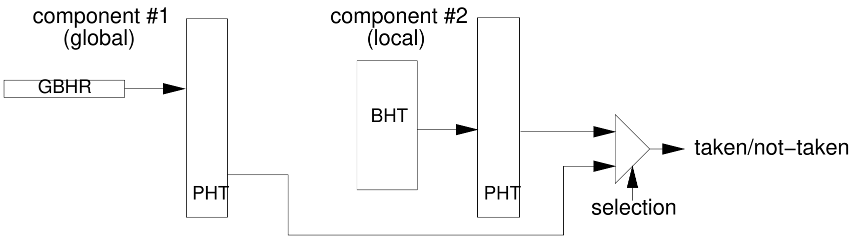

Fig. 1. The organization of a hybrid predictor with two different components. (The left-hand component shows a high-level schematic of a global-history predictor, and the right-hand component shows a local-history predictor.) The selector can be dynamic, requiring a meta-predictor structure, or static, in which case each branch is assigned to a component at compile time.

Hybrid predictors have drawbacks. Designing an effective selection mechanism can be difficult. More importantly, hybrid prediction only works well with a large hardware budget. This problem exists because a hybrid predictor must subdivide the available area into these different components, and each component must be large enough to behave well in its own right. Yet for small hardware budgets, there simply isn't enough hardware available to accomplish this.

Alloying. This paper proposes an alternative—alloying—as a superior way to expose both global and local history and to attack the wrong-history problem. Alloyed prediction performs competitively to an equal-area hybrid predictor for large hardware budgets, and substantially outperforms

[2] By *wrong history*, we do not mean that the actual history bits are incorrect; rather, the *type* of history being tracked is inappropriate for the branch at hand.

hybrid predictors for smaller hardware budgets. In other words, alloying provides *robust* performance for both large and small hardware budgets. Alloying also consistently outperforms other two-level organizations.

Alloying is a simple idea, but to our knowledge it has never appeared in the literature. It is a pseudo-hybrid organization that looks just like a two-level, local-history predictor, and merely adds a global-history register.[3] The predictor then *alloys* global and local history bits into one PHT index. Figure 2 shows the organization we propose. This simple modification attacks the drawbacks of two-level organizations—by exposing both global and local history—and the drawbacks of hybrid organizations—by avoiding the need for a selector and by avoiding the need to subdivide the hardware into multiple branch-prediction components.

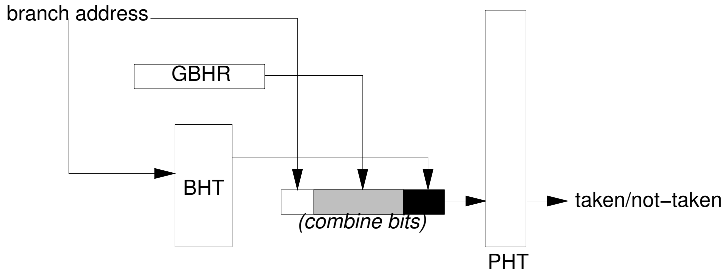

Fig. 2. The organization of a two-level predictor with an alloyed index. This "MAs" predictor combines local history from the per-branch history table (BHT) and global history from the global branch-history register (GBHR) with some address bits to compose the PHT index.

We call the organization shown in this figure *MAs*, because it resembles GAs and PAs predictors[4] [32] in concatenating the different types of bits. GAs and PAs predictors try to reduce conflicts in the PHT by concatenating the history—whether global or local—with some bits from the branch address. In this way, two unrelated branches that share the same prior history should be distinguished and mapped to different PHT entries by their differing branch addresses. MAs does this too, as shown by Figure 2. However, to obtain the same degree of anti-aliasing, MAs typically needs fewer address bits than GAs or PAs. This is because alloying global and local history itself provides some anti-aliasing capability: unrelated branches that alias with one kind of history often

[3]Note that a hybrid predictor typically contains a local-history predictor as one component. But it exposes global history using a separate, full-size component that performs global-history prediction.

[4]In this naming scheme, the first letter gives the type of history tracked: Global, Per-address (local), or Merged (alloyed). The second letter indicates whether the predictor's PHT is Adaptive (*i.e.*, dynamic), or Static. And the third letter indicates the PHT structure: "g" indicates no anti-aliasing, "s" indicates *select* or concatenation-style anti-aliasing, and "p" indicates perfect anti-aliasing (no conflicts ever; GAp, PAp, and MAp are ideal in this regard) [32].

can be distinguished using the other kind of history.

As mentioned before, other anti-aliasing schemes have been proposed that outperform GAs and PAs, although often at the cost of additional complexity [5], [14], [17], [21], [27]. We restrict this particular work to GAs, PAs, and MAs predictors, and hybrid predictors that use them as components. Although we may not use the most aggressive anti-aliasing technique, the comparison remains a sensible one. Comparing predictors that use the same conflict-reduction technique keeps our experiments fair, and should give an accurate picture of alloying's usefulness. A further consideration is that, unlike many proposed anti-aliasing techniques, this "select" or concatenation-style of anti-aliasing is already a well-established technique and appears in current microprocessors like the AMD K6 [4]. Restricting our work to "select" organizations also substantially simplifies the comparison of two-level, alloyed, and hybrid predictors. Finally, "select"-style anti-aliasing makes comparisons between alloying and hybrid straightforward: it is not obvious how to extend most proposed anti-aliasing techniques to hybrid prediction.

Alloying is certainly not restricted to an MAs configuration. It is a variation on conventional two-level, history-based techniques, so like those techniques, alloying can also be implemented with aggressive conflict-reduction techniques: for example, we would have "mshare" instead of "gshare" [16], or "YAMS" instead of "YAGS" [5]. In each case, alloying would presumably benefit slightly less from the anti-aliasing than strictly global- or local-history predictors (because alloying already achieves some anti-aliasing), but alloying would presumably be more effective at removing wrong-history mispredictions.

To quantify the importance of wrong-history mispredictions and motivate alloying, the next section describes a taxonomy of mispredictions and then shows how much each source of mispredictions actually contributes to the overall misprediction rate. Following the taxonomy, Section III briefly describes the simulator and benchmarks used in the rest of this study. Section IV then shows that MAs is consistently the best organization among possible two-level predictors. Section V compares alloyed prediction against the alternative for removing wrong-history mispredictions, hybrid predictors. Section VI discusses this paper's results and briefly explores dynamic reconfigurability for branch predictors. Finally, Section VII describes related work, and Section VIII concludes the paper.

II. A Taxonomy of Mispredictions

A taxonomy of mispredictions has three virtues. It shows the relative importance of different misprediction types. It permits us to tailor branch-prediction solutions to individual misprediction types—a divide-and-conquer approach—rather than devise a single, all-purpose branch predictor. And it provides better understanding of branch-predictor behavior: merely devising a taxonomy yields insight. Indeed, alloying simply suggested itself while we developed this taxonomy.

As mentioned before, a great deal of work has explored ways to prevent conflict mispredictions in two-level predictors. This paper shows that predictors also suffer from other important types of mispredictions. For example, two-level and hybrid predictors are sophisticated structures that can take a long time to learn a branch's behavior, often producing a substantial number of training-time mispredictions [6], [12]. And—as we have already pointed out—most programs suffer severely from wrong-history mispredictions. It is important to understand the relationship among these different sources of mispredictions, but we are aware of no prior work that organizes such misprediction categories into a broad framework and measures the relative importance of so many sources of mispredictions.

We by no means claim the taxonomy is comprehensive: the included categories can presumably be refined, and it lacks some obvious categories: update timing [22], history length, and history pollution [7]. This is partly because our work on taxonomies is still in its beginning stages, partly because we restrict ourselves to GAs and PAs predictors of fixed history length, and partly because measurements using the taxonomy are difficult. This is therefore only a first step toward a rigorous and thorough analysis of what causes branch mispredictions, a step that we hope leads to further research in understanding branch predictability.

Yet even this simple taxonomy is an important contribution. It substantially extends prior efforts at categorizing branch mispredictions, by organizing a number of recognized misprediction types into a single classification scheme and by describing a one-pass method for counting them. It shows the importance of wrong-history mispredictions. It shows the risk of continuing to focus on conflict mispredictions. And it has helped us discover alloying.

Figure 3 shows the sequence of tests used to classify each misprediction. Note that wrong-history mispredictions are only counted after conflict and training-time mispredictions. This provides a

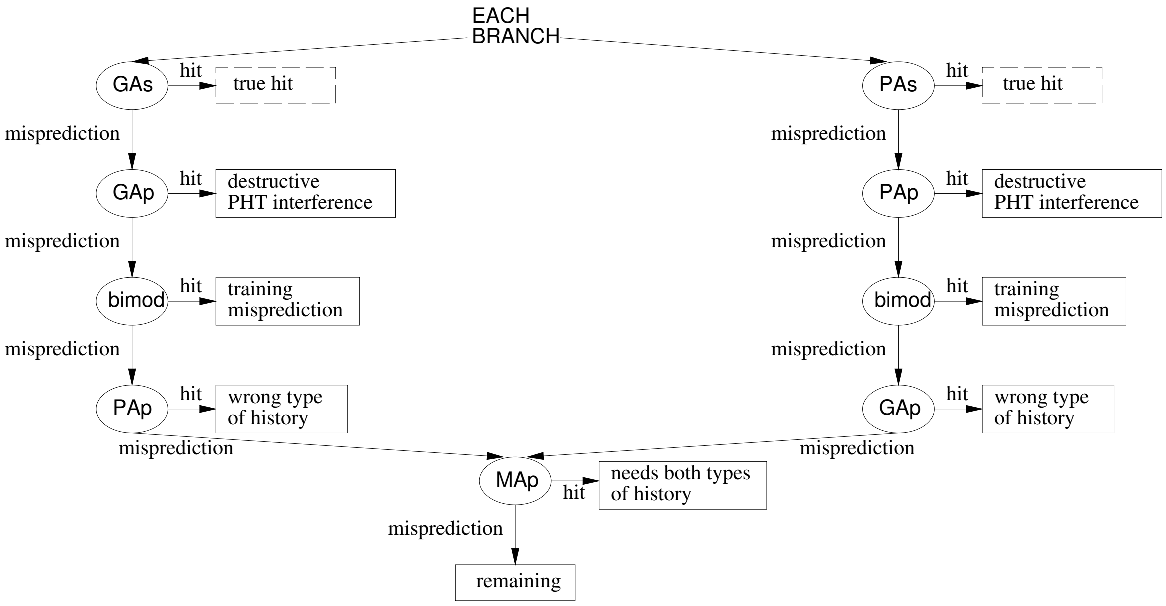

Fig. 3. A flowchart depicting how the taxonomy categorizes misprediction types. Each dynamic branch flows down both sides until it is either categorized or falls through.

measure of "true" wrong-history mispredictions. Measurements are accomplished by running in parallel several predictor organizations of increasing sophistication. If a branch mispredicts in one organization while predicting correctly in another, the difference between the two configurations isolates the misprediction category. The simulator performs the pictured cascade of tests until the branch either predicts correctly, or the misprediction fails all tests. *Remaining* branches are either inherently difficult to predict, or fall into a category not yet included in the taxonomy. The depicted process simultaneously categorizes each dynamic branch's behavior for both GAs and PAs predictors.

A. Taxonomy Categories

Destructive PHT and BHT conflicts. All dynamic predictors that track state can suffer from destructive conflicts when unrelated branches map to the same predictor entry. Destructive PHT (pattern history table) conflicts arise when branches map to the same 2-bit PHT counter and these branches go in opposite directions.[5] These *conflict-mispredictions* can be identified by running a finite and infinite PHT in parallel (GAs and GAp predictors, or PAs and PAp). The two predictors behave the same, except that the infinite PHT cannot suffer from conflicts. A misprediction in the

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

finite PHT that does not occur in the infinite PHT must therefore be a destructive conflict.

Aliasing in the BHT (branch history table) can also cause mispredictions. To simplify an already complicated measurement, here we omit their impact by assuming an interference-free BHT, while all later sections evaluate predictor performance using realistic BHT configurations.

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 the impact of training time can be estimated using a simple bimodal predictor[6] as follows. First eliminate conflict mispredictions. Then training-time mispredictions occur when the main branch predictor fails, while an equal-size bimodal predictor succeeds. The assumption is that if a branch mispredicts, but predicts correctly in the bimodal organization, the branch is predictable; the main predictor just has not yet learned its behavior.

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. These are the *wrong-history mispredictions*, which can be reduced or eliminated by hybrid or MAs predictors. As mentioned, the measurements here separate "true" wrong-history mispredictions from those merely caused by aliasing. We argue that the only true wrong-history mispredictions are those that cannot be solved by eliminating conflict or training-time mispredictions. The above techniques are therefore used first, to eliminate all conflict and training mispredictions. Then, if a misprediction remains in a GAs organization while a PAs organization predicts the branch correctly, global history must be the wrong type of history for this branch instance. Similarly, if PAs fails while GAs succeeds, local history must be the wrong type.

Needs combined history. For some branches, neither type of history alone suffices; instead the branch needs both types of information simultaneously. This occurs if a branch correlates with other branches and also has some self-repeating pattern. The frequency of these *combined-history* mispredictions can be estimated using an MAp predictor: mispredictions which do not fall into the preceding categories and which an MAp organization eliminates are combined-history mispredic-

[6] A bimodal or "2-bit" predictor—proposed by Smith in [25]—is just a bare PHT, indexed by branch address.

tions. Their number is always the same for both GAs and PAs. Note that a hybrid predictor cannot eliminate this type of misprediction.

Remaining mispredictions. Mispredictions that cannot be eliminated using these techniques fall into a “left-over” category. These *remaining* branches are either inherently difficult to predict, or fall into a category not yet included in the taxonomy.

B. Taxonomy Results

The taxonomy measurements use a modified version of SimpleScalar 2.0’s instruction-level *sim-bpred* simulator [1]. Benchmarks are compiled for SimpleScalar’s portable ISA (PISA) using *gcc* version 2.6.3 at maximum optimization, and executed with the reference inputs. Later results in the paper use cycle-level simulation, but the use of instruction-level simulation here permits longer simulations of one billion instructions. For programs like *compress* with a long and unrepresentative startup phase [23], the billion instructions are taken from later in the program; in the case of *compress* in particular, the measurement simply captures one complete compression phase. For programs with a small startup phase—*m88ksim, xlisp*, and *gnuchess*—just the first billion instructions are measured, and *gcc* and *wolf* are short enough to run to completion. Other benchmarks first skip over part of their execution before gathering statistics. More detail on the benchmarks appears in the next section.

Figure 4 presents a breakdown of misprediction types for GAs and PAs predictors of different sizes: 64 Kbits (32K PHT entries), 8 Kbits (4K PHT entries), and 2 Kbits (1K PHT entries). Because these taxonomy measurements use a perfect BHT, its size is not included in the total area (but later sections use realistic BHT configurations). This does mean the total misprediction rate for PAs is understated, and the training time for PAs is slightly overstated. Nevertheless, the bar segments faithfully depict the *relative* importance of PHT conflict, wrong history, combined history, and uncategorizable mispredictions. PAs just lacks an additional segment to show the number of BHT conflicts.

For each branch predictor size, all possible GAs, PAs, and MAs configurations were tested, and the one configuration that performs best overall for the entire benchmark suite is the one used for the experiments. That configuration is reported in Table I. MAp is included because it is used to determine the impact of combined history.

TABLE I

PREDICTOR CONFIGURATIONS USED FOR TAXONOMY MEASUREMENTS.

	GAs/GAp	PAs/PAp	MAp
32K entries	8 global, 7 address	14 local, 1 address	10 global, 4 local, 1 address
4K entries	5 global, 7 address	10 local, 2 address	7 global, 4 local, 1 address
1K entries	1 global, 9 address	10 local, 0 address	5 global, 4 local, 1 address

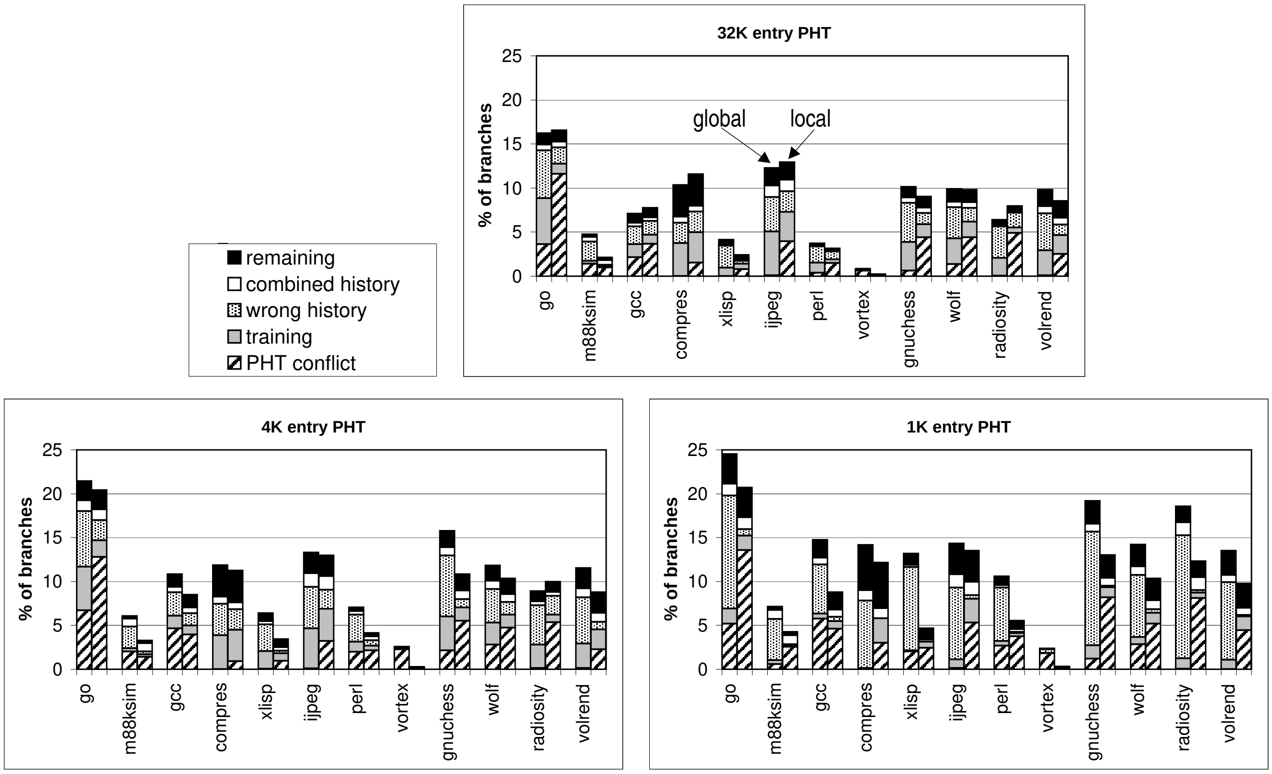

Fig. 4. Breakdown of misprediction types for GAs and PAs with 32K-entry, 8K-entry, and 4K-entry PHTs and an interference-free BHT. KEY: For each benchmark, the left-hand bar represents GAs, and the right-hand bar PAs. Shorter bars mean fewer mispredictions.

As expected, PHT conflicts are important, and as expected, that importance declines with increasing PHT size. Still, even with the simple concatenation-style anti-aliasing used by GAs and PAs, PHT conflicts are often less important than training time and wrong history. This is especially true for global-history predictors. Overall—for each of the three sizes—conflicts comprise an average of 15–20% of mispredictions for the GAs predictor, and 40–52% for the PAs predictor.

Wrong-history mispredictions are instead often the single most important cause of mispredictions for global-history predictors, comprising an average of about 35% of mispredictions for the 8 Kbit and 32 Kbit GAs predictors, and 50% for the 2 Kbit GAs predictor. This is true even though only true wrong-history mispredictions are counted (all conflict and training-time mispredictions are first eliminated). Wrong-history mispredictions are less dominant in local-history predictors, comprising about 14.5% and 17.5% of the mispredictions for the 8- and 32-Kbit PAs predictors, and 3% for the 2 Kbit predictor.

Combined-history mispredictions are usually unimportant (6–7% of mispredictions), although alloying does help eliminate them.

It might seem curious that in the 2 Kbit predictor, wrong-history is especially important for global history, and especially unimportant for local history. (This can be seen in the large wrong-history segments for GAs in Figure 4, and the near-absence of those segments for PAs.) This happens because at that size, the chosen GAs configuration tracks only 1 bit of branch history, while PAs tracks 10 bits. Under these circumstances, PAs frequently has better information than GAs, and GAs almost never has better information than PAs. As the global history grows longer with larger predictors, this problem diminishes. Furthermore, once the global history is long enough, GAs also captures correlation behavior that local history never can. This means that some PAs mispredictions that were uncategorizable at small sizes[7] can be predicted correctly by larger GAs predictors. This converts those mispredictions to wrong-history mispredictions for a large PAs. As a result, PAs's wrong-history component grows with predictor size, and the uncategorizable component shrinks.

III. Simulation and Benchmark Details

A. Simulator

The remainder of this paper uses detailed cycle-level simulation to compare the performance of different branch-predictor configurations. Simulations are performed using HydraScalar, our modified version of SimpleScalar 2.0's *sim-outorder* [1]. HydraScalar has been configured to approximately model an Alpha 21264 [10], [13]. It performs out-of-order execution with a 64-entry instruction window, and issues up to 4 integer and 2 floating-point instructions each cycle. The two-level, non-blocking cache hierarchy has 2-cycle, 64 KByte first-level instruction and data caches and a 12-cycle, unified, 8 MByte second-level cache. Memory latency is 100 cycles, and the TLB miss latency is 30 cycles. The instruction- and data-TLBs each contain 128 entries. We

[7] These are properly labeled uncategorizable, because no small predictor organization can make a correct prediction.

hold the 2-way associative branch target buffer (BTB) fixed at 2K entries and the return-address stack fixed at 32 entries. The stack and branch history are updated speculatively at fetch time with suitable repair mechanisms [22], [24]. Branch mispredictions suffer at least a 7-cycle latency, but branches whose direction is correctly predicted and merely miss in the BTB suffer only a 2-cycle penalty. Indirect-branch mispredictions suffer the full minimum-7-cycle latency.

Some experiments also model an 8-issue processor. This processor resembles in most respects the model just described, but can issue up to 8 integer instructions; as many as 4 of these may instead be floating-point instructions. The instruction window contains 128 entries and the first-level caches are 128 KBytes.

B. Benchmarks

These evaluations use not only the SPECint95 benchmarks [29], but also four other primarily integer benchmarks. The four other integer benchmarks have been added to show that the results in this work are not specific to characteristics of SPECint programs. Table II summarizes the benchmarks' characteristics. All are compiled using `gcc -O3 -funroll-loops` (`-O3` includes inlining) for the SimpleScalar PISA. The SPEC programs use "ref" inputs. Some benchmarks come with multiple reference inputs, in which case one has generally been chosen.

TABLE II
BENCHMARK SUMMARY.

	Warmup insts	Input	Conditional branch static sites	Conditional branch dynamic refs
go	925 M	9stone21	4,627	11.2 M
m88ksim	25 M	dhrystone	231	16.2 M
gcc (cc1)	220 M	cccp.i	14,245	14.7 M
compress	2575 M	ref	205	11.8 M
li (xlisp)	270 M	9queens	271	15.4 M
ijpeg	823 M	vigo.ppm	657	5.1 M
perl	600 M	scrabble	352	12.9 M
vortex	2450 M	ref	3,134	12.2 M
gnuchess	150 M	level 10	665	9.6 M
wolf	50 M	cbd	2,288	15.9 M
radiosity	300 M	room	163	9.4 M
volrend	125 M	largest input	57	6.5 M

"Warmup insts" indicates the length of the preliminary phase of simulation, before statistics-gathering.

Gnuchess comes from the IBS benchmark suite [30] and is GNU's implementation of 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 [26]. 1.7% of its instructions are floating-point operations. *Radiosity* and *volrend* were chosen from the SPLASH2 suite [31] of parallel applications for shared memory because these two programs have significant misprediction rates and interesting behavior. These are compiled 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.

Some benchmarks 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 [23]. The simulator therefore begins gathering statistics much later in the program. During the preliminary phase of simulation, branches and memory references are still presented to the simulator, warming up the predictor and caches. After the warmup phase (also shown in Table II) completes, the simulator runs in full-detail, cycle-level mode for a further 1 million instructions to prime all the processor structures. Then statistics are gathered for the next 100 million instructions.

IV. Comparing Two-Level Prediction Schemes

This section compares the three types of two-level predictors described in this paper: GAs, PAs, and MAs. Unlike the taxonomy results in Section II, these results are based on cycle-level simulations using a finite BHT.

To get the best comparison for each predictor size, the GAs, PAs, and MAs configurations that perform best overall for the entire benchmark suite must be used. Finding the best composition of PHT index bits was done using brute force, simulating all possible combinations of global, local, and address bits for the desired branch-predictor size. Finding equal-area configurations must also account for the BHT's size. We explored all possible BHT configurations for the chosen size, ranging from wide and short (many local-history bits and few BHT entries) to narrow and tall. The importance of BHT contention makes BHT height 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. The configurations chosen appear in Table III.

Figures 5–7 compare the IPC (instructions per cycle) obtained with GAs, PAs, and MAs for branch predictor hardware budgets of 64 Kbits, 8 Kbits, and 2 Kbits. For reference, a simple

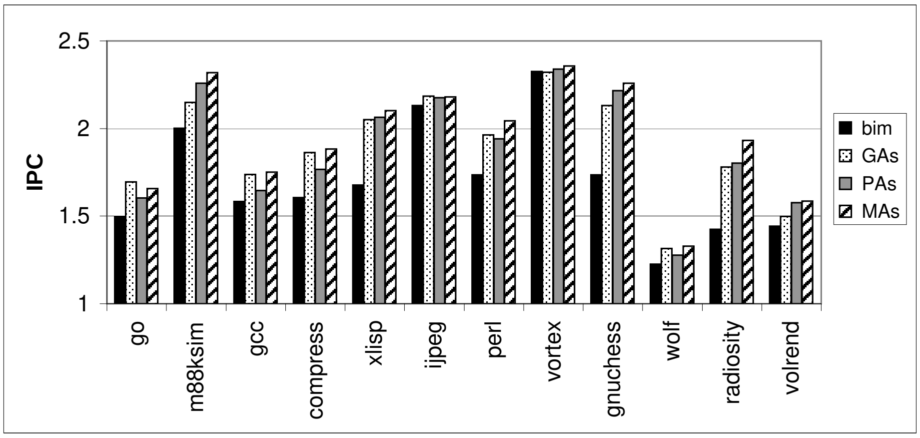

Fig. 5. Relative performance of bimodal ("bim"), GAs, PAs, and MAs for 64 Kbits total size, including a finite BHT. Taller bars represent better performance.

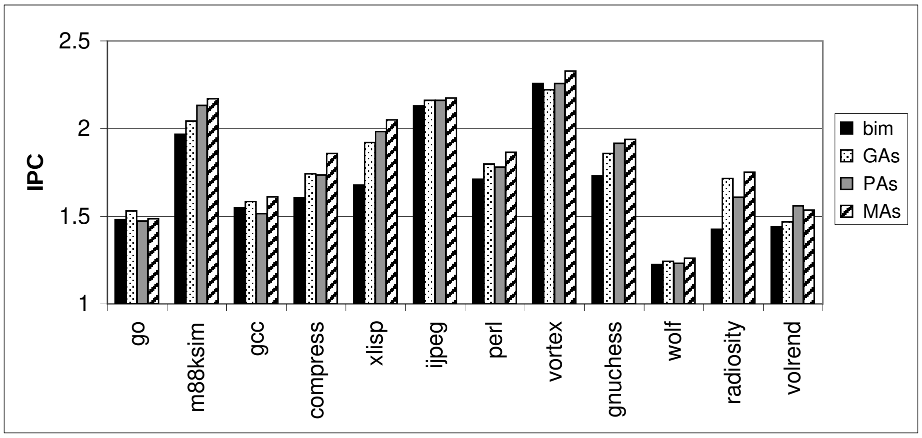

Fig. 6. Relative performance of bimodal, GAs, PAs, and MAs for 8 Kbits total size, including a finite BHT.

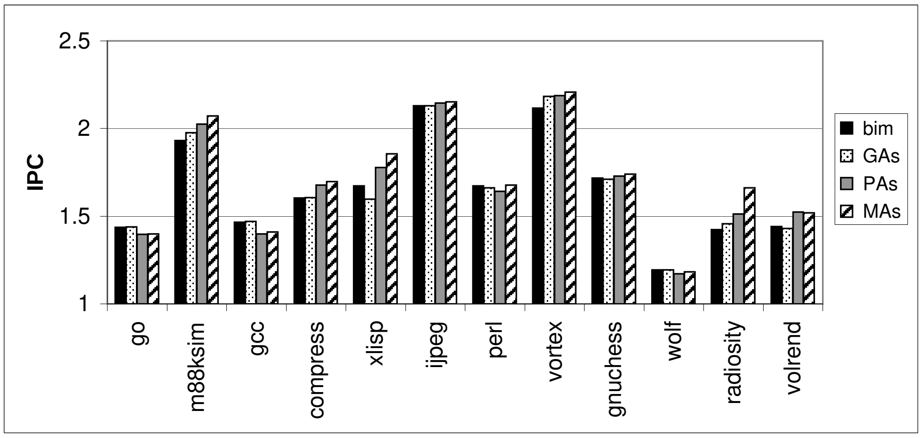

Fig. 7. Relative performance of bimodal, GAs, PAs, and MAs for 2 Kbits total size, including a finite BHT.

TABLE III
PREDICTOR CONFIGURATIONS USED FOR EQUAL-TOTAL-SIZE COMPARISON.

	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

"g" indicates the number of global-history bits, "p" local-history bits, and "a" address bits.

bimodal predictor of the appropriate size is shown as well. The results are for the 4-issue processor.

The results are strong: MAs consistently outperforms the alternatives. For many benchmarks, MAs is better by a substantial margin of 4–8%. For the 64 Kbit and 8 Kbit sizes, MAs is always better than bimodal and PAs. MAs also outperforms GAs for most benchmarks: 10 out of 12 for 64 Kbits and 11 out of 12 for 8 Kbits. For 2 Kbits, MAs outperforms bimodal and GAs for 9 out of 12 benchmarks, and PAs for 11 out of 12 benchmarks (the twelfth is a tie). Table IV shows MAs's speedup compared to each of the other organizations. The speedups are even better in an 8-issue configuration: Table V reports the average speedups for 64 Kbits.

TABLE IV
SPEEDUP OF MAs OVER EACH LISTED PREDICTOR ORGANIZATION FOR A 4-ISSUE PROCESSOR.

	64 Kbits			8 Kbits			2 Kbits		
	bimodal	GAs	PAs	bimodal	GAs	PAs	bimodal	GAs	PAs
go	1.108	0.978	1.034	1.003	0.971	1.010	0.973	0.972	1.001
m88ksim	1.158	1.080	1.027	1.104	1.063	1.018	1.073	1.049	1.024
gcc	1.107	1.008	1.065	1.039	1.017	1.063	0.962	0.959	1.009
compress	1.172	1.010	1.066	1.157	1.066	1.071	1.058	1.057	1.012
xlisp	1.253	1.025	1.019	1.222	1.068	1.034	1.108	1.161	1.044
ijpeg	1.023	0.999	1.002	1.020	1.006	1.006	1.010	1.010	1.004
perl	1.177	1.041	1.052	1.090	1.037	1.047	1.002	1.010	1.021
vortex	1.014	1.015	1.008	1.031	1.048	1.032	1.043	1.011	1.010
gnuchess	1.301	1.060	1.019	1.119	1.043	1.011	1.013	1.016	1.007
wolf	1.084	1.010	1.041	1.029	1.013	1.024	0.991	0.990	1.009
radiosity	1.356	1.085	1.072	1.229	1.020	1.088	1.166	1.140	1.099
volrend	1.099	1.058	1.005	1.064	1.045	0.984	1.053	1.062	0.997
MEAN	1.154	1.031	1.034	1.092	1.033	1.032	1.038	1.036	1.020

We also evaluated *gshare*-style [16] versions, where the history and address strings are xor'd together. Like Sechrest *et al.* [21], we found little added benefit. Xor'ing actually helps MAs slightly more than GAs or PAs.

These speedups translate into substantial reductions in the misprediction rate. For some bench-

TABLE V

MEAN SPEEDUP OF MAS OVER EACH LISTED PREDICTOR ORGANIZATION (AT 64 KBITS) FOR AN 8-ISSUE PROCESSOR.

64 Kbits		
bimodal	GAs	PAs
1.227	1.046	1.050

TABLE VI

MEAN REDUCTION IN MISPREDICTION RATE ACHIEVED BY MAS.

64 Kbits		8 Kbits		2 Kbits	
GAs	PAs	GAs	PAs	GAs	PAs
23.1%	22.8%	19.6%	16.9%	11.8%	6.8%

marks like *m88ksim, perl*, and *vortex*, a 64 Kbit MAs halves the misprediction rate compared to an equivalent-area GAs! Table VI reports how much overall MAs reduces the misprediction rate compared to GAs and PAs. Only the average is reported to save space, because the improvement in misprediction rate is approximately proportional to the speedup.[8] The reduction in mispredictions is also mostly independent of issue width.

In summary, MAs outperforms PAs, because MAs augments plain local-history prediction with some global history. MAs also almost always outperforms GAs because MAs needs only a few local-history bits, so it can still track a long global history string. Indeed, MAs can track a *longer* global history than GAs, because MAs needs only a few address bits; combining global and local history already reduces aliasing,

A final observation is that MAs is often so much better that a substantially smaller MAs configuration can be used. For example, compared to a 64 Kbit GAs predictor, a mere 8 Kbit MAs ($\frac{1}{8}$ the size!) does as well or better for 5 benchmarks: *m88ksim, compress, xlisp, ijpeg*, and *vortex*. A 2 Kbit MAs does as well for *volrend*! The IPC of an 8 Kbit MAs is within 5% of the 64 Kbit GAs for a further three benchmarks. A much smaller MAs might therefore plausibly replace larger GAs predictors, and could be especially appealing for embedded or other low-cost processors.

[8] The relationship is not strictly proportional, because it also depends on the frequency of branches within the program. *Vortex*, for example, obtains smaller speedups than its 50%+ reduction in mispredictions would suggest, since it has one of the lower branch frequencies in the SPEC suite.

V. Alloyed History vs. Hybrid Prediction

The previous section argues that, among two-level predictors, alloying is a superior choice. Of course, GAs and PAs are restricted to one type of history, and suffer from wrong-history mispredictions. We also must evaluate alloying against hybrid prediction, which like alloying can attack the wrong-history problem.

Unlike hybrid, alloying can eliminate "combined-history mispredictions," but these are only a small problem, as Figure 4 showed. Otherwise, hybrid predictors that combine both a global-history and a local-history component can eliminate many of the same mispredictions as MAs. Hybrid prediction with dynamic selection has the potential to do a better job than MAs of attacking the PHT-conflict problem, by migrating branches from one component to the other to reduce conflicts.

As with the comparison of two-level predictors, a fair comparison of MAs against hybrid prediction requires finding the best hybrid predictor for each size. This first requires choosing between a dynamic and a static selector.

All the evaluations in this section only consider sizes of 64 Kbits and 8 Kbits; hybrid prediction is not feasible at 2 Kbit, because the components would simply be too small. Indeed, the data in Section V-B show that 8 Kbits is also too small for a hybrid predictor. But an alloyed predictor works well even down to 2 Kbits.

A. *Static vs. Dynamic Choice in Hybrid Predictors*

Hybrid predictors can use either static or dynamic selection to choose which predictor component to use for each branch. Branches that do well with global history are directed to the global-history component, and branches that do well with local history are directed to the local-history component.

Grunwald *et al.* [8] argue that static selection outperforms dynamic selection. Profiling can identify which component branches prefer, and a static assignment can be made at compile time. This dispenses with the dynamic selector, and the extra area can be used to make the prediction components larger. By assigning a branch to one or the other component, static selection also has the advantage that branches only cause conflicts in one component. Static selection does require instruction set support, and also requires high-quality training data to make the profiling accurate.

Yet some *individual* branches dynamically alternate between needing global history and needing local history. Because a static selector assigns each branch to one or the other, it penalizes any branches that alternate between history types, while dynamic selection can accommodate such behavior. The taxonomy's wrong-history data in Figure 4 only provides cumulative data for programs as a whole. Here we measure whether individual branch sites dynamically vary the type of history they use, using 4K entry PHTs and interference-free BHTs. We find that a significant number of branches do in fact switch between using global and local history.

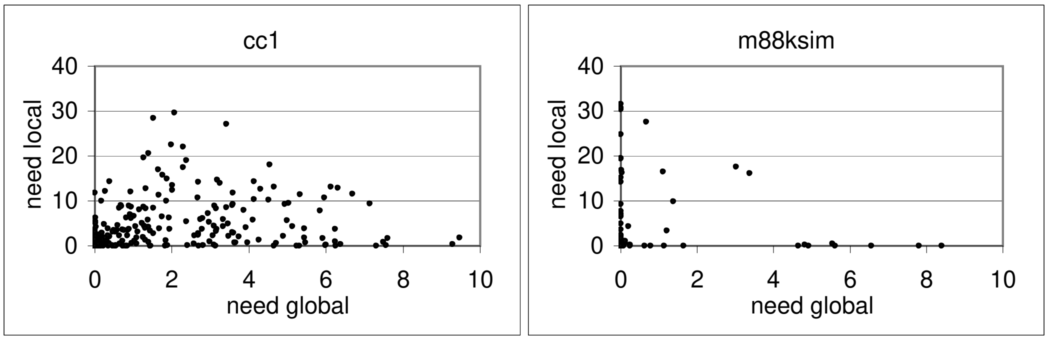

Fig. 8. X-Y scatter plots showing, for each branch site, the percentage of time that branch needs a global predictor or a local predictor.

Figure 8 presents X-Y scatter graphs for two benchmarks, *gcc* and *m88ksim*. Each point represents one branch site; only branches executed 100,000 times or more are plotted. The plotted points are not otherwise weighted by frequency. The graph's x-coordinate gives the percentage of time that a branch can only be predicted with global history; its y-coordinate gives the percentage of time that a branch can only be predicted with local history. Branches that consistently use one component lie on one or the other axis. In both cases, a substantial amount of mass lies at the origin: branches that are easily predicted by either structure, and branches that are mispredicted by both structures. Mass that lies in the middle of the graph indicates branches that need access to both types of history. These branches are penalized in a static-hybrid predictor. *Gcc* has many such branches; *m88ksim* has fewer, but they execute a huge number of times.

These data show that individual branches do alternate between using global and local history (we have not measured the frequency of this alternation: some branches might change between history types just once). Unless static selection can do a substantially better job of eliminating conflicts, dynamic selection ought to outperform static selection.

To see which is better in practice, we compared the performance of dynamic-selection and static-selection hybrid predictors at 64 Kbits and 8 Kbits, this time using realistic (finite) BHTs. To make the comparison fair, we used the predictor configurations that perform best overall for the benchmark suite, first testing a wide range of component sizes, selector sizes, and history lengths. While exploring configuration options, we made sure to test hybrid organizations that replace either the global- or local-history component or the selector with a simple, 2-bit, bimodal structure of various sizes. This benefited some benchmarks, especially for small hybrid predictors, but for both 64 Kbits and 8 Kbits, the organization that was best overall did not use any bimodal structures. The configurations eventually chosen appear in Tables VII and VIII.

TABLE VII
PREDICTOR CONFIGURATIONS USED FOR 64 KBIT DYNAMIC AND STATIC HYBRID PREDICTORS.

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

TABLE VIII
PREDICTOR CONFIGURATIONS USED FOR 8 KBIT DYNAMIC AND STATIC HYBRID PREDICTORS.

	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	

For 8 Kbit predictors, where conflicts are a more serious problem, static selection is better for a few benchmarks, but only by a small margin. Dynamic selection is still better for 8 of the 12 benchmarks. For 64 Kbits, dynamic selection is almost uniformly superior—static selection is better for only a single benchmark. Curiously, these results contradict those reported by Grunwald *et al.* [8]. They find that more benchmarks prefer static selection than dynamic selection. We suspect that the difference either arises from area calculations—they use a 4-way associative BHT and do not count the BHT tags against the area—or from training methodology. They apparently use shorter inputs and do no warmup, a procedure which penalizes the dynamic selector.

B. Hybrid vs. Alloyed Prediction

The above results strongly suggest that dynamic selection outperforms static selection. Since MAs also outperforms a static-selection hybrid predictor—for both 64 Kbits and 8Kbits for every benchmark but one—we use a dynamic-hybrid predictor as the best comparison against MAs.

To get a fair comparison, we must compare the best configuration for each organization. These were already found as part the preceding experiments: the dynamic-hybrid configurations are the same as those reported in Tables VII and VIII; the MAs configurations are the same as those reported in Table III of Section IV.

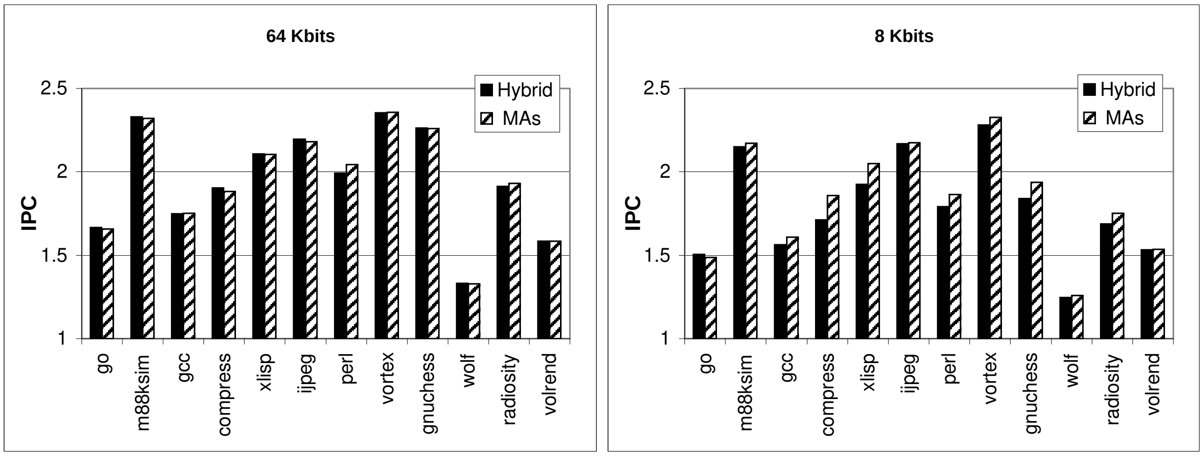

Fig. 9. Relative performance of dynamic-selection hybrid and MAs for 64 Kbits and 8 Kbits total area. Taller bars represent better performance.

TABLE IX
SPEEDUP OF MAS OVER DYNAMIC-SELECTION HYBRID FOR A 4-ISSUE PROCESSOR.

	64 Kbits	8 Kbits
go	0.996	0.987
m88ksim	0.995	1.010
gcc	1.001	1.031
compress	0.988	1.085
xlisp	0.997	1.065
ijpeg	0.993	1.002
perl	1.025	1.040
vortex	1.001	1.020
gnuchess	0.998	1.053
wolf	0.998	1.010
radiosity	1.010	1.038
volrend	1.001	1.003
MEAN	1.000	1.029

Figure 9 gives performance figures for the two choices, and Table IX reports speedups. At 64 Kbits, hybrid prediction and MAs provide roughly equivalent performance. Hybrid prediction is slightly better than MAs for five of the benchmarks, but the average speedup over all the benchmarks is 1.0. Both are doing a good job of eliminating wrong-history mispredictions. Hybrid performs well because 64 Kbits is enough area to subdivide into the different components.

The picture is different for an 8 Kbit predictor. Here, MAs is superior for all but one benchmark, and usually by a substantial margin. The overall speedup for MAs compared to hybrid is 2.9%. This corresponds to an average 15% reduction in mispredictions. At small sizes like 8 Kbits, a hybrid predictor—whether using dynamic or static selection—simply does not have enough area to subdivide into smaller components. For five of the benchmarks, even GAs outperforms hybrid prediction.

In summary, MAs performs at least as well as hybrid prediction for large hardware budgets, and substantially outperforms hybrid prediction for smaller hardware budgets.

VI. Discussion

A. *The Importance of Small Branch Predictors*

Some readers may wonder why 8 Kbit and 2 Kbit branch predictors are of any interest today, when some processors now use much larger predictors. For example, the Alpha 21264's predictor is about 28 Kbits [13].

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 cost constraints likewise. Processors for embedded environments are generally both space- and power-constrained.[9] Even high-performance processors may need to use smaller branch predictors, if a larger predictor would worsen the chip's cycle time. According to *Microprocessor Report* [4], for example, the AMD K7 uses a simple 8 Kbit bimodal predictor for exactly this reason.

These smaller branch-prediction environments still require the best branch prediction available, because prediction accuracy remains a powerful lever over performance. One might think a simple bimodal organization would be the best choice. Figures 6 and 7 show that this is not so. Alloying's robust performance therefore makes it attractive for a range of processors, from high-performance processors with large branch-prediction budgets to small embedded processors.

[9] Better branch prediction also reduces power consumption by reducing power expended on mis-speculated computation [15].

B. Configurability

We mention a final interesting aspect of alloyed predictors. While exploring the design space to find the best configuration for each predictor, we observed that the best *overall* configuration is often not the best for individual programs. *M88ksim*, for example, benefits from a large local-history width, whereas most programs do better with a small local-history width (and either address bits or global-history bits comprising the rest of the PHT index, depending on whether alloying is used). For a 64 Kbit predictor, the choice of configuration can alter *m88ksim's* prediction accuracy by as much as 50%, and for *gcc*, by as much as 35%.

Although the magnitude of the variation is impressive, it should not be surprising that one-size-fits-all configurations are not the best choice for any individual benchmark. Yet this suggests that processors might allow dynamic configurability of the PHT index. More substantial reconfigurability is of course possible, but this tiny degree of reconfigurability has tiny cost, while providing substantial performance boosts for programs that do not behave like the "average benchmark."

The dynamic history-length fitting of Juan *et al.* [12] is an aggressive means for accomplishing this. It varies the history length while a program runs, but requires significant hardware support. It would suffice to set the PHT index's configuration once, when a program starts. No hardware is needed to find the best configuration: it can be determined via profiling that simulates the program's behavior on different possible branch-predictor configurations. If the processor makes both a global-history register and a BHT of moderate width are available, profiling can select a configuration from organizations spanning a range from GAs through various MAs configurations and on to PAs. Only a single GBHR, BHT, and PHT are needed (in essence, an MAs configuration); the program chooses which GBHR and BHT bits to use for indexing the PHT. Facilitating this would require only some state in the processor control register and a two- or three-input multiplexor on each index bit of the PHT. For organizations that already include a BHT, the only significant cost is then the added gate delay associated with these multiplexors. In more aggressive implementations, the multiplexor could be replaced by any combinational circuit with a sufficiently short delay, allowing a more sophisticated operation—like xor, or perhaps even hardware supporting user-programmable functions—on the input bits.

MAs therefore has a further advantage compared to hybrid prediction. Not only does it provide robust performance, but it is easily made reconfigurable with the addition of a tiny amount of hardware. Making a hybrid predictor reconfigurable, on the other hand, is a substantially more involved undertaking.

VII. Related Work

We are unaware of any published work describing alloying, but a great deal of literature explores how to best design global- and local-history predictors, especially in avoiding PHT conflicts. Pan *et al.* [19] observed that a tradeoff exists between including history and address bits in the PHT index, and introduced GAs. Yeh and Patt [32] further examined the balance between history and address bits for GAs and PAs. They found that global-history schemes suffer more from aliasing and are more strongly sensitive to both history length and 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, and in addition found that xor'ing history bits and address bits, as in *gshare* [16], provides little benefit.

Other researchers have described a variety of more aggressive techniques for reducing conflict mispredictions. Sprangle *et al.* [27] identified the substantial harm caused by PHT interference and described an *agree* predictor, which reinterprets the 2-bit counter's meaning as "agree with static prediction" or "disagree with static prediction". Michaud *et al.* [17] measured compulsory, capacity, and conflict aliasing and described a *skewed* predictor, in which the branches simultaneously exist in multiple PHTs, and each PHT is indexed using a different hash function. A voting function combines the multiple PHT results to generate a prediction. Lee *et al.* [14] observed that conflicting substreams may be strongly biased, just in opposite directions. They described 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] extended the bi-mode scheme with the YAGS predictor, which tries to learn when branches disagree with bi-mode's bias.

It is important to note that MAs can easily be extended to incorporate these anti-aliasing schemes. The original schemes use only one type of history, while an alloyed version of these aggressive schemes exposes both types of history, so we would expect alloying to remain superior.

Chang *et al.* described a different approach: *branch classification*, which identifies highly-biased branches. These may both cause and suffer from conflicts, so they described a mechanism to filter them out of the predictor reference stream. Classification also reduces BHT conflicts.

Yet other work has focused on characterizing why mispredictions happen in two-level predictors. Talcott *et al.* [28], Sprangle *et al.* [27], Young *et al.* [33], and Chang *et al.* [2] characterized PHT interference, and showed that while both significant amounts of both constructive and destructive interference occur, the destructive interference consistently dominates. Young *et al.* also showed, as did Nair [18], 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.* [7] focused on the correlation characteristics of branches, and found that many branches do benefit from global history. Yet for a given prediction, most of the global history bits go 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 propagate through a program in [20].

Hybrid prediction was originally proposed by McFarling [16]. Chang *et al.* [3] extended his work, finding that the most beneficial components are a global-history predictor and a local-history predictor. They also showed that a global-history selector outperforms a bimodal selector. Evers *et al.* [6] further extended the two-component predictor by proposing a *multi-hybrid* predictor. This organization includes sophisticated prediction structures, a simple bimodal predictor, static-prediction components, and a dynamic selector design that steers branches to the appropriate component. Most recently, Grunwald *et al.*. [8] have proposed eliminating the selector in favor of a static selection mechanism. Our results suggest this does not work well for two-component hybrid predictors. But the selection mechanism becomes more complex in multi-hybrid predictors, and in this case static selection might be beneficial.

VIII. Conclusions and Future Work

A great deal of prior branch prediction work has focused on ways to improve two-level predictors that use either local or global history, but not both. Such has mainly focused on reducing conflict mispredictions due to aliasing in the pattern history table.

This paper has presented a taxonomy of misprediction types to better understand what categories of mispredictions are important. Using the taxonomy, we have shown that conflict mispredictions are important, but other categories are often more important. In particular, *wrong-history* mispredictions are often the most important source of mispredictions, comprising up to 50% of the total mispredictions. Wrong history occurs when a branch requires one type of history (global or local) but the predictor provides the other type. Hybrid predictors can attack these mispredictions, but an *alloyed* predictor is superior for several reasons.

An alloyed predictor merges local and global history bits together in a single PHT index. Although such an organization is a minor 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 but less important category of mispredictions in which branches need both types of history simultaneously. Alloying can also reduce PHT aliasing, because branches that alias with one type of history are often distinguished by the other type of history.

Our proposed alloyed predictor, MAs, substantially outperforms other solo, two-level predictors—and even compares well against those with a larger area. MAs also performs well against hybrid predictors, especially at smaller sizes where the components in a hybrid organization become too small. Small predictors are interesting for constrained environments like embedded or low-cost processors, and design exigencies sometimes force even high-performance processors to use smaller predictors. Overall, our results show MAs to be an approach that yields robust results for a wide range of benchmarks and predictor sizes. Finally, it is important to note that MAs is only one type of alloyed organization. Just as GAs and PAs can be improved by incorporating techniques for reducing conflicts, MAs can be, too. For example, MAs retains its superior performance when bit-strings are xor'd as in *gshare* [16].

There are several promising avenues for further research. We have briefly suggested how an MAs organization makes a reconfigurable branch predictor easy to implement, and this merits further investigation. Also, MAs might provide further benefits as part of a hybrid predictor. In particular, it might work well with static selection, which assigns branches at compile time to either the global-history or local-history component, but penalizes a significant number of branches that alternate between the history types. MAs could solve this problem by exposing both types of history in one component, which might obviate the need for dynamic selection. A third area for future work is in finding more effective hashing functions for alloying the global and local history bits. Finally, additional work on understanding branch-prediction behavior will certainly be beneficial.

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] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *Proc. PACT '96*, pages 48–57, Oct. 1996.

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

[4] K. Diefendorff. K7 challenges intel. *Microprocessor Report*, pages 1, 6–11, Oct. 26 1998.

[5] A. N. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proc. Micro-31*, pages 69–77, Dec. 1998.

[6] 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.

[7] 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.

[8] D. Grunwald, D. Lindsay, and B. Zorn. Static methods in hybrid branch prediction. In *Proc. PACT '98*, pages 222–29, Oct. 1998.

[9] L. Gwennap. Intel's P6 uses decoupled superscalar design. *Microprocessor Report*, pages 9–15, Feb. 16, 1995.

[10] L. Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, pages 11–16, Oct. 28, 1996.

[11] 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.

[12] 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.

[13] R. E. Kessler, E. J. McLellan, and D. A. Webb. The Alpha 21264 microprocessor architecture. In *Proc. ICCD 1998*, pages 90–95, Oct. 1998.

[14] 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.

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

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

[17] 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.

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

[19] 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.

[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. *Characterizing and Removing Branch Mispredictions*. PhD thesis, Princeton Univ. Dept. of Computer Science, June 1999.

[23] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Trans. Computers*. To appear.

[24] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Improving prediction for procedure returns with return-address-stack repair mechanisms. In *Proc. Micro-31*, pages 259–71, Dec. 1998.

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

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

[27] 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.

[28] A. R. Talcott, M. Nemirovsky, and R. C. Wood. The influence of branch prediction table interference on branch prediction scheme performance. In *Proc. PACT '95*, pages 89–96, June 1995.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.