Alloyed Branch History: Combining Global and Local Branch History for Robust Performance

UNIV. OF VIRGINIA DEPT. OF COMPUTER SCIENCE TECH. REPORT CS-2002-21

Zhijian Lu, John Lach, Mircea R. Stan, †Kevin Skadron
Dept. of Electrical and Computer Engineering and †Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

This paper introduces *alloyed* prediction, a new two-level predictor organization that combines global and local history in the same structure, combining the advantages of two-level predictors and hybrid predictors. The alloyed organization is motivated by measurements showing that *wrong-history mispredictions* are even more important than conflict-induced mispredictions. Wrong-history mispredictions arise because current two-level, history-based predictors provide only global or only local history. The contribution of wrong-history to the overall misprediction rate is substantial because most programs have some branches that require global history and others that require local history.

This paper explores several ways to implement alloyed prediction, including the previously proposed *bi-mode* organization. Simulations show that *mshare* is the best alloyed organization among those we examine, and that mshare gives reliably good prediction compared to bimodal ("two-bit"), two-level, and hybrid predictors. The robust performance of alloying across a range of predictor sizes stems from its ability to attack wrong-history mispredictions at even very small sizes without subdividing the branch predictor into smaller and less effective components.

Keywords

Branch prediction, mispredictions, branch history, alloying, bi-mode prediction, aliasing, conflicts, two-level prediction, combined history, hybrid prediction, combining predictors, tournament predictors, taxonomy, categorization, classification.

I. INTRODUCTION

The question of how to better predict the direction of conditional branches has received intense study in recent years. The need for accurate conditional-branch prediction is well known: mispredictions waste large numbers of cycles, inhibit out-of-order and parallel execution, and waste energy on mis-speculated computation. Continued work on better prediction of conditional branches is important, because prediction accuracy still lies only in the 90–97% range for most benchmarks. 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 17 wasted cycles in the Pentium 4 [8]. The minimum misprediction penalty will only continue to grow if pipelines continue to get longer, as recent work [11], [12], [35] suggests. The *average* misprediction penalty is larger still—10 to 13 cycles for a system like the 21264 [30], longer yet for systems like the Pentium 4 and future long pipelines—because branches in out-of-order processors such as these can spend an arbitrary time in the instruction window waiting to issue. In multi-issue processors, each of these cycles represent a wasted opportunity to issue not one, but possibly several instructions. Furthermore, even with the ability to issue instructions out of order, the instruction window can rarely expose sufficient instruction-level parallelism (ILP) to hide such long misprediction penalties. Finally, branch mispredictions also inhibit the effectiveness of other processor structures [30] and the ability to exploit even wider issue capabilities. Indeed, Jouppi and Ranganathan [17] claim that branch prediction will be the most restrictive bottleneck in processors by 2010, worse even than stalls from the increasingly high performance memory system.

In pursuing better prediction, two-level [26], [41] and hybrid [24] predictors[1], which explicitly track prior branch history, have received special attention. Most of this attention has been focused on reducing aliasing errors (*conflict mispredictions*), which arise when unrelated branches happen to collide in a particular branch-predictor entry and overwrite each other's state. A wealth of effective techniques have been developed to reduce conflict occurrence in the pattern history table (PHT)[2] of two-level predictors [5], [25], [27], [36]. Even *without* using aggressive anti-aliasing techniques, conflicts account for only 15–20% of mispredictions in global-history predictors and 40–50% in local-history predictors.

Work on hybrid predictors with global- and local-history-based components [3], [9] implicitly acknowledges another source of mispredictions: predictors that do not track the necessary type of history. Branches that need local history usually behave poorly when the predictor tracks only global history, and vice versa. We call these predictions *wrong-history mispredictions*, and they are often more common than conflict mispredictions. 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. As noted in Section II, for SPECint95 programs using a global-history predictor, wrong-history mispredictions account for 35–50% of the total misprediction rate. Hybrid predictors have been developed specifically to combat this type of misprediction, but we propose *alloyed prediction* as a more generally useful alternative.

Predictors must also work well at small sizes, as fast clock speeds and comparatively slower wires constrain the size of a predictor that can be accessed in a single cycle, a requirement of most

[1] Also called combining or tournament predictors.

[2] 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.

current architectures. Indeed, Jiménez, Keckler, and Lin [15] point out that the table size that can be accessed within a single cycle is beginning to shrink. They calculate that in a 100nm micron process for example, only structures smaller than 1–2 Kbytes will have single-cycle access times.

Unfortunately, the organizations that are the most effective at reducing conflict mispredictions usually require large structures to spread out the branches. The same is true for existing organizations that combat wrong-history mispredictions. While multi-predictor organizations (with a small fast first predictor and a larger, more accurate backup predictor) are a possible solution [15], [21], it is important to also have predictors that work well at small sizes for designs that cannot afford such duplication for cost, complexity, or power reasons. This is especially true for lower-cost embedded processors where cost and power are among the most important factors. Branch predictors must provide the highest possible performance while meeting this myriad of increasingly strict specifications.

These considerations are all excellent motivation for alloyed prediction, which combines (or "alloys") both global and local history bits in the same PHT index. Alloyed prediction looks like a conventional, two-level PAs[3] predictor with an added global-history register. This simple change, however, exposes both types of history in the same structure, which makes alloying a new form of hybrid prediction. Alloyed history not only attacks wrong-history mispredictions, but its use of multiple types of history bits provides anti-aliasing and hence attacks conflict mispredictions as well. *Bi-mode prediction* [23], one of the best conflict-reducing predictors, achieves its benefits this way, because its "choice predictor" is really tracking local history. As we show here, the bi-mode organization is actually one specific way to implement alloyed prediction, but other alloyed organizations that can use as many local-history bits as the structure size allows generally perform better.

The combination of global and local history enables the alloyed approach to behave well for both small and large predictor sizes, because, unlike hybrid prediction, alloying can provide both types of history and achieve its benefits without subdividing the available hardware budget into separate components that are too small to be effective. Overall, alloying is a new approach that combines the best features of various previous organizations to achieve robust performance across a range of predictor sizes.

This paper first motivates alloyed prediction by using a simple taxonomy of mispredictions [32]

[3] In this naming scheme, described in [41], the first letter gives the type of history tracked: Global or Per-address (local). 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 and PAp are ideal in this regard).

to measure wrong-history mispredictions. This taxonomy is used to explain the performance of different predictors and to establish the need for alloying. Section III describes alloying in more detail and compares its organization to hybrid and bi-mode predictors. Section IV briefly describes the simulator and benchmarks used in the rest of this study. Then Section V compares and analyzes the performance of alloyed predictors against more conventional types of two-level prediction (GAs and PAs [41]), bi-mode prediction, hybrid prediction, and the original dynamic predictor, bimodal prediction [33] (in which the PHT is directly indexed by the branch address to select a two-bit counter—not to be confused with bi-mode prediction [23]). Related work is discussed in Section VI, and Section VII presents conclusions and future work.

II. A Taxonomy of Mispredictions

To illustrate the need for alloyed prediction, we categorize mispredictions into several major classes: destructive conflicts, training, wrong history, and "other". This taxonomy shows the relative importance of these different misprediction types and in particular, the importance of wrong history.

As mentioned above, 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. 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 the many sources of mispredictions.

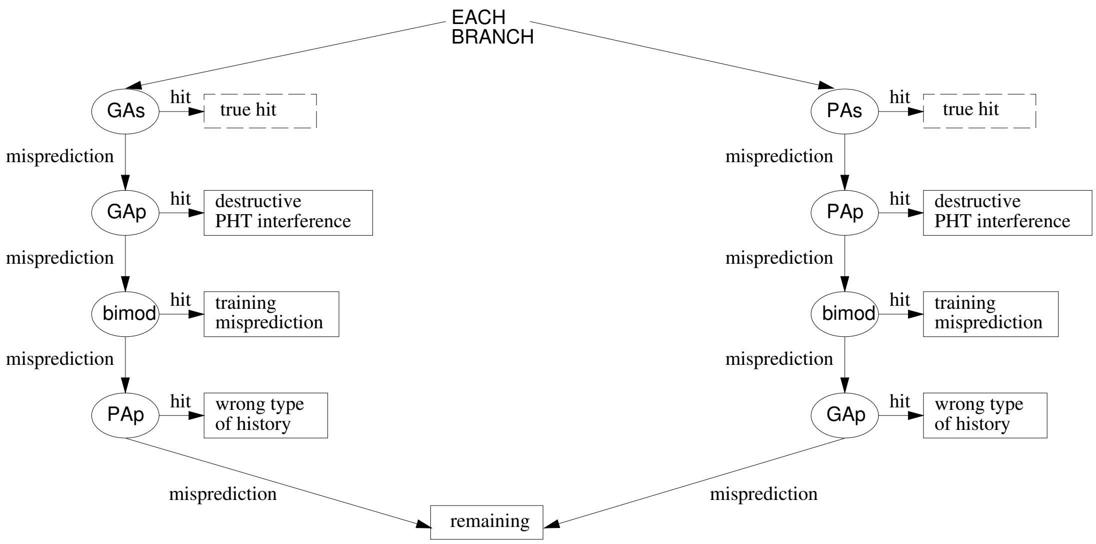

Fig. 1. A flowchart depicting how the taxonomy categorizes misprediction types. Each dynamic branch flows down both sides until it is either categorized or falls through.

Figure 1 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 ensures that only mispredictions which cannot have been caused by conflict or training effects can be counted as 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.

We by no means claim the taxonomy is comprehensive: the included categories can presumably be refined, and it lacks some obvious categories: update timing [31], history length, and history pollution [7]. Although this taxonomy 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. that is not the focus of this work. The point of this section is not to develop a complete taxonomy but rather to illustrate the importance of wrong-history mispredictions and the need for alloying.

A. Taxonomy Categories

Destructive PHT and BHT conflicts. All dynamic predictors that track state suffer from destructive conflicts when unrelated branches map to the same predictor entry. Destructive PHT conflicts arise when branches map to the same two-bit PHT counter and they are biased in opposite directions.[4] 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 does not suffer from conflicts. A misprediction in the 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, but all later sections evaluate predictor performance using realistic BHT configurations.

Training-induced mispredictions. If a misprediction is not caused by PHT conflict, it can instead occur because the predictor has not yet learned the branch's behavior. This happens especially at

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

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 yet tractable method for measuring *training mispredictions*, so for the illustrative purpose in which this taxonomy is being used, we estimate training mispredictions using a simple bimodal predictor as follows. First eliminate conflict mispredictions by using an infinite PHT. Then a measure of training-time mispredictions can be obtained by observing when the main branch predictor fails, while an idealized bimodal predictor succeeds. The assumption is that if a branch mispredicts in the GAp or PAp but predicts correctly in the bimodal organization, the branch is predictable; the main predictor just has not yet learned its behavior. This admittedly neglects the time it takes the bimodal predictor to train (not long for a two bit saturating counter), but it provides a good estimate of the effect of training-induced mispredictions.

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*.

Global history can expose correlation among branches, while local history is well suited for branches that follow a consistent pattern. However, most programs have some branches that do well with global history *and* some branches that do well with local history. If the branch predictor only tracks one or the other, some branches find that the predictor provides the wrong type of history. 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.

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 conflicts or training-time issues. The above techniques are therefore used first, eliminating 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 (when eliminating the "correct prediction by chance" factor). Conversely, if PAs fails while GAs succeeds, local history must be the wrong type.

A possible drawback of using both types of history is that the measurement of wrong-history mispredictions is tied to the anticipated predictor size. Yet any predictor under consideration will have some finite size, and the behavior of the branches is dictated by the maximum history length that size can entertain. Some wrong-history mispredictions will therefore occur, even though they might be eliminated by a more idealized organization. At the limit, one might consider infinite history or prediction by partial matching [4]. This would not measure wrong-history, but rather the intrinsic predictability of a branch. Our approach characterizes the degree to which *a particular predictor size* produces wrong-history mispredictions and a different history type for the same predictor size could remove those mispredictions and therefore illustrates the need for hybrid or alloyed prediction.

Remaining mispredictions. Mispredictions that cannot be eliminated using these techniques fall into a "left-over" category. These *remaining* mispredictions are either inherently difficult to remove 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]. The benchmarks (described in detail in Section IV-B) 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 [30], 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.

Figure 2 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 that 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 conflicts, 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 and PAs configurations were tested, and the

TABLE I
PREDICTOR CONFIGURATIONS USED FOR TAXONOMY MEASUREMENTS.

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

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.

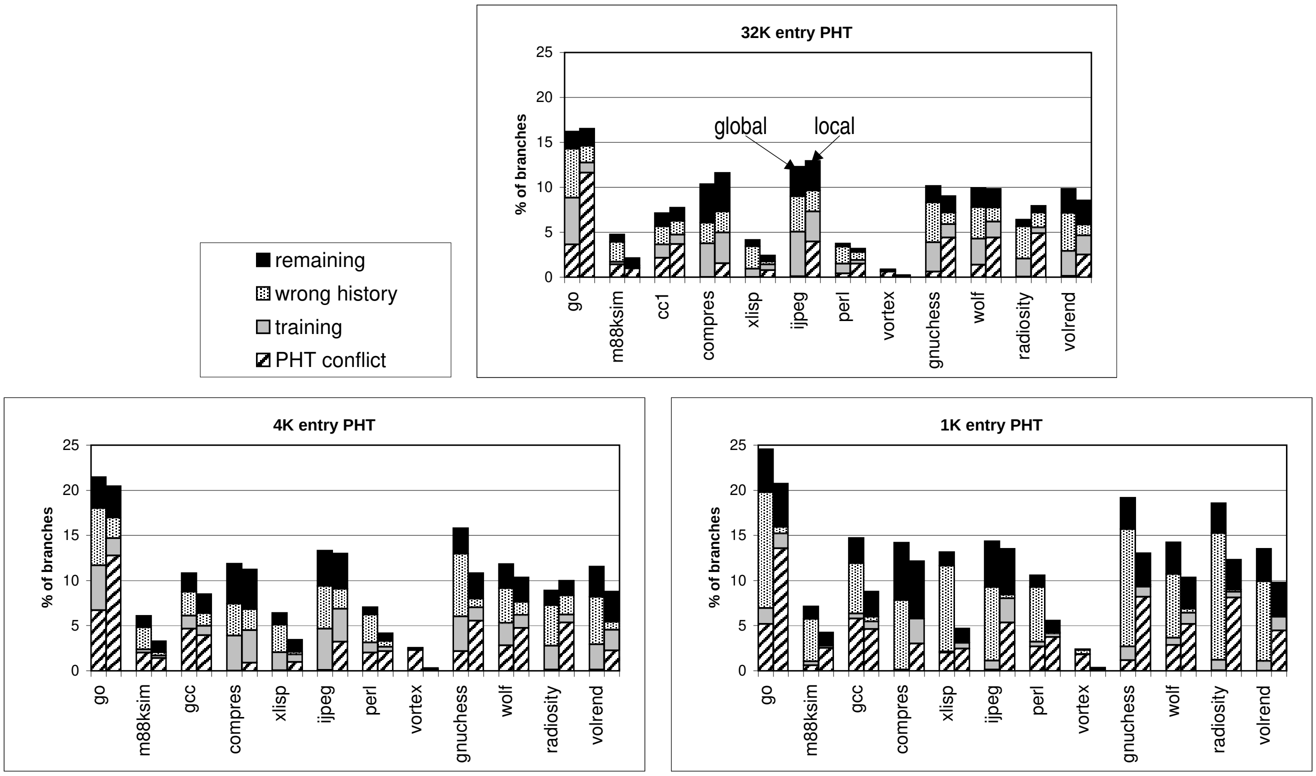

Fig. 2. Breakdown of misprediction types for GAs and PAs with 32K-entry, 8K-entry, and 4K-entry PHTs and an interference-free BHT. KEY: For each benchmark, the left-hand bar represents GAs, and the right-hand bar PAs. Shorter bars mean fewer mispredictions.

As expected, PHT conflicts are important, and as also 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.

In most cases, wrong-history is the most common 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.

Another view of the importance of wrong-history mispredictions can be found by using a hybrid predictor with a global and local component [3] (see Fig. 4 and recording for each static branch the number of times it chooses the local or global prediction component. To perform this test, we use a 24 Kbit hybrid predictor where each component is 8 Kbits. Figure 3 shows the results of these measurements for two benchmarks, *m88ksim* and *go* and also for the average over all our benchmarks (the center column of graphs). Each graph shows the distribution of branches' preference for global vs. local history, with branches preferring local history 100% of the time in the rightmost bin and branches preferring global history 100% of the time in the leftmost bin. The top row presents the distribution for static branches (*i.e.* branch sites) and the bottom row presents the distribution for dynamic branch instances. *M88ksim* and *go* are chosen because they represent the two extremes, with *m88ksim* having almost all of its branches consistently preferring the same type of history, and *go* having a large fraction of branches without a strong preference. Even when looking at static branches, on average 25% of the branches do not have a strong preference, and a large fraction of dynamic branch executions vary between needing global and local history.

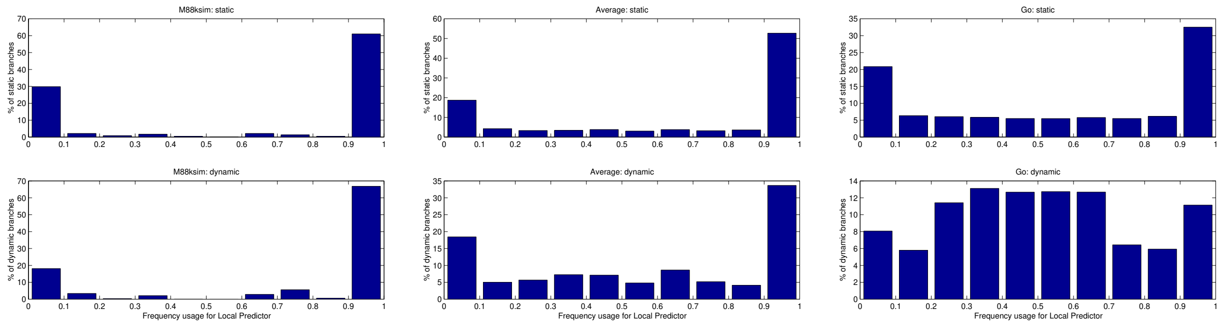

Fig. 3. Distribution of static (top row) and dynamic (bottom row) branches' preference for local vs. global history in a hybrid predictor. The preference for local history is given by the x-axis. The 40%–50% bin, for example, includes all branches that require local history to make a correct prediction between 40 and 50 percent of the time, and otherwise prefer global history. Results are given for m88ksim and go, which represent the two extremes of this distribution, and for the average over all of our benchmarks.

Figure 3 illustrates two points. First, what we already know from the preceding data, that most benchmarks have some benchmarks requiring global history and some requiring local history; an effective predictor must provide both. More importantly, with the exception of a few benchmarks like *m88ksim*, many dynamic branches do not have a consistent preference for local or global history: in the graph presenting the average over all the benchmarks, about 50% of the mass lies between the two endpoints, and for *go*, 80% of the mass lies between the two endpoints. If a branch switches often between the two types, the selector in a hybrid predictor may have difficulty keeping up. Alloyed prediction is ideally suited for this behavior.

This taxonomy has established the high frequency of wrong-history mispredictions. The following section discusses alloyed-history predictors that reduce such mispredictions and perform well for a wide range of predictor sizes.

III. Alloyed History Predictors

A. Hybrid Predictors

Hybrid predictors [24] are one way to attack wrong-history mispredictions. 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 therefore 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. Figure 4 presents a high-level schematic of a hybrid predictor that combines global and local prediction components.

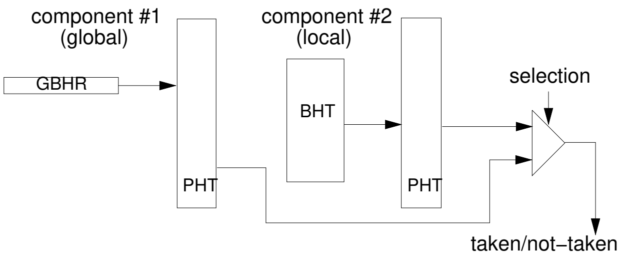

Fig. 4. The organization of a hybrid predictor with two different components. (The left-hand component is a global-history predictor, and the right-hand component is a local-history predictor.) The selector can be dynamic, requiring a meta-predictor structure, or static, in which case each branch is assigned to a component at compile time.

While hybrid predictors do help with the large percentage of wrong-history mispredictions (as shown by the taxonomy in Section II-B), hybrid predictors have drawbacks. Designing an effective selection mechanism can be difficult. More importantly, as our results later in this paper show, 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 and smaller components. If the total hardware budget is too small, the subcomponents will be smaller yet and ineffective as a result, yielding poor overall behavior.

B. Bi-Mode Predictors

The bi-mode predictor, proposed by Lee, Chen and Mudge [23] and shown in Figure 5, was developed to attack destructive interference between branches that map to the same PHT entry but have opposite biases (*i.e.*, one is taken, one is not taken). Branches that alias but have the same bias are harmless. The bi-mode predictor therefore maintains two PHTs, one for branches with a bias toward taken, one for branches with a bias toward not taken. These PHTs are indexed in the *gshare* [24] manner of XORing a global-branch-history string with bits from the branch PC. A *choice predictor*, indexed only by the branch PC, uses two-bit counters to learn each branch's bias and therefore indicate which PHT the branch should use.

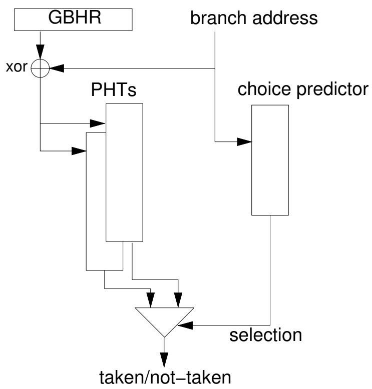

Fig. 5. The organization of a bi-mode predictor. The "choice predictor" uses two-bit counters to learn for each branch whether it is biased toward taken or not taken. This value is then used to assign the branch to one of the two PHTs.

Note that bi-mode prediction is different from *bimodal* prediction, a common name for the simple table of two-bit up-down saturating counters proposed by Smith in 1981 [33] and used in a variety of processors throughout the 1990s.

C. Alloyed Predictors

This paper proposes an alternative—*alloying*—as a superior way to expose both global and local history to attack the wrong-history problem while still minimizing aliasing.

It is a pseudo-hybrid organization that looks just like a two-level, local-history predictor, and merely adds a global-history register. The predictor then *alloys* global and local history bits into one PHT index. Figure 6 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 eliminating both the need for a selector and the need to subdivide the hardware into multiple branch-prediction components.

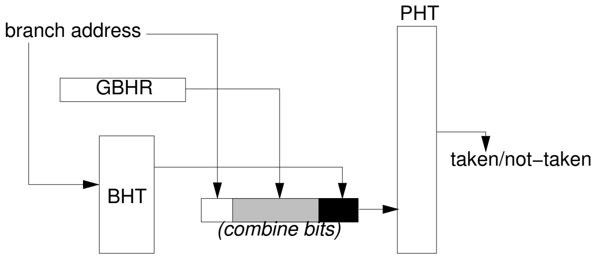

Fig. 6. The organization of a two-level predictor with an alloyed index. This "MAs" predictor combines local history from the per-branch history table (BHT) and global history from the global branch-history register (GBHR) with some address bits to compose the PHT index.

We call the organization shown in this figure *MAs*, because it resembles GAs and PAs predictors in combining via concatenation the different types of bits (the "M" stands for "merged" history).[5] 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 6. 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 can be distinguished by the other kind of history.

Alloying is essentially a generalization of the bi-mode predictor. However, at first glance, the bi-mode predictor seems quite different from the MAs predictor above in Figure 6. Rearrangement, however, shows the similarity. This can be seen in Figure 7, which redraws the bi-mode predictor to resemble MAs. If the choice predictor is viewed as the BHT of a local-history predictor and the two direction PHTs are viewed as logical halves of a physically unified table, the similarity between bi-mode and alloying can be seen. The choice predictor is tracking per-branch—*i.e.*, local—history, and the high-order bit of its two-bit counter is used as the highest-order index bit, thereby selecting which half of the PHT to use.

[5] In Sections V-B and V-C we extend the alloyed scheme to XOR the global-history and branch-address bits, creating what could be called an *mshare* predictor.

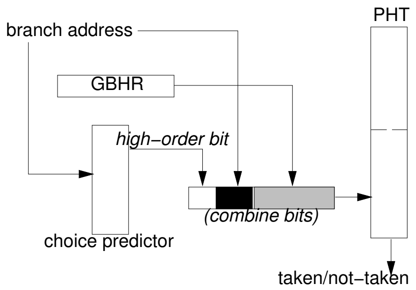

Fig. 7. A bi-mode predictor reorganized to resemble the MAs predictor depicted in Figure 6.

D. Access-Time Considerations

As stated above, most current architectures require a conditional branch prediction to be made in a single clock cycle. At first glance, the MAs organization, as it was described above, would appear to have a longer access time than a conventional two-level predictor or even a conventional hybrid predictor. This is because the MAs predictor would perform two table lookups in series. First it would probe the BHT, in order to get the local-history bits to be combined with the global-history and address bits. Only then could the PHT be accessed.

However, if the number of local-history bits is small, this problem can be avoided. The PHT can be broken into multiple physical tables, accessed in parallel, similar to the bi-mode organization (Figure 5). The local history bits are then used as the selector on a multiplexor that chooses the outcome from the appropriate table.[6] This organization is shown in Figure 8. It permits the PHT and BHT lookups to proceed in parallel.

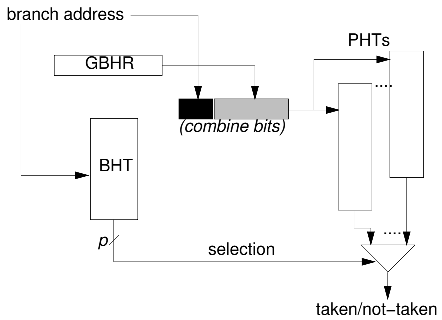

Fig. 8. An MAs predictor rearranged to permit simultaneous PHT and BHT access. The original, unified PHT is broken into 2^p separate tables, all accessed simultaneously. The p local-history bits are then used to select which value to use for the final prediction.

Breaking the PHT into multiple banks like this is also a natural solution to access-time concerns for large PHTs, because each smaller bank will have a faster access time. A further consideration is

[6] This still works for an XOR scheme–mshare–if only the global-history and branch-address bits are XOR'd.

that the multiple simultaneous table accesses will dissipate somewhat more power than the single access to one large table. These considerations are true for any organization, not just MAs. The roles of lookup time and power were not further evaluated, as we felt them to be beyond the scope of this one paper.

IV. Simulation and Benchmark Details

A. Simulator

This paper uses both instruction-level and detailed cycle-level simulation to compare the performance of different branch-predictor configurations. Cycle-level simulations are performed using HydraScalar [28], which is based on SimpleScalar 2.0 [1] but substantially extends the detail of the way it treats branch prediction and pipeline modeling. Simulations do not model kernel behavior or context switches, instead performing operating-system calls by proxy; but all non-kernel behavior, including library code, is simulated.

For these experiments, HydraScalar's out-of-order simulator has been configured to approximately model an Alpha 21264 [21]. It performs out-of-order execution with a 64-entry instruction window, an eight-stage pipeline with stages for decoding, renaming, and enqueuing of instructions, and issue capability of up to 4 integer and 2 floating-point instructions per cycle. The two-level, non-blocking cache hierarchy has two-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 hold the two-way associative branch target buffer (BTB) fixed at 2K entries and the return-address stack fixed at 32 entries. Branch mispredictions are resolved at writeback time, and HydraScalar models multiple layers of misprediction with full detail. Branch predictor updates take place at commit, but the stack and branch history are updated speculatively at fetch time with suitable repair mechanisms if the speculative updates are later found to be incorrect [18], [21], [29], [31]. 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. The predictor we model makes a prediction for each branch fetched, but within a group of fetched instructions, those that follow the first predicted-taken branch are discarded, as control must now jump to a new location. This effectively means that the fetch engine fetches through not-taken branches but stops at taken branches.

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 the SPECint95 benchmarks [37] and four other primarily integer benchmarks. Table II summarizes the benchmarks' characteristics. All are compiled using *gcc* version 2.6.3 for the SimpleScalar PISA, with optimization set at `-O3 -funroll-loops` (`-O3` includes inlining). The SPEC programs use "ref" inputs. Some benchmarks come with multiple reference inputs, in which case one has generally been chosen. *Xlisp* is an exception; it used the 9-queens input. Gnuchess was set to level 10, and the SPLASH benchmarks used the largest input.

TABLE II
BENCHMARK SUMMARY.

	Warmup	Conditional branch counts			
		100 M insts		1 B insts	
	insts	static	dyn.	static	dyn.
go	925 M	4,627	11.2 M	5,331	112 M
m88ksim	25 M	231	16.2 M	968	162 M
gcc (cc1)	220 M	14,245	14.7 M	20,783	190 M
compress	2575 M	205	11.8 M	203	151 M
li (xlisp)	270 M	271	15.4 M	676	154 M
ijpeg	823 M	657	5.1 M	1,415	58 M
perl	600 M	352	12.9 M	614	129 M
vortex	2450 M	3,134	12.2 M	3,203	124 M
gnuchess	150 M	665	9.6 M	1,127	96 M
wolf	50 M	2,288	15.9 M	2,993	26 M
radiosity	300 M	163	9.4 M	183	92 M
volrend	125 M	57	6.5 M	660	70 M

Data is given for simulations of both 100 million and 1 billion instructions. "Warmup insts" indicates the length of the preliminary phase of simulation, before statistics-gathering.

Gnuchess comes from the IBS benchmark suite [39]; *wolf* is the timberwolf circuit router and comes from Smith's Unix-Utils benchmark suite [34], and 1.7% of its instructions are floating-point operations. *Radiosity* and *volrend* were chosen from the SPLASH2 suite [40] of parallel applications for shared memory because these two have significant misprediction rates. *Radiosity* computes the equilibrium distribution of light in a scene and *volrend* renders a three-dimensional volume using a ray-casting technique.

Some benchmarks come with multiple reference inputs, in which case one has generally been chosen. For *go*, we choose a playing level of 50 and a 21x21 board with the `9stone21` input. For *m88ksim*, we use the `dhrystone` input; for *gcc*, `cccp.i`; for *ijpeg*, `vigo.ppm`; and for *perl*, we use the scrabble game. But for *xlisp*, we run the program with all the supplied LISP files as arguments.

Simulations are fast-forwarded to a representative portion of the program's execution. The fast-forward length is presented in Table II. Then statistics are gathered for the next 100 million instructions for cycle-level simulations and 1 billion instructions for instruction-level simulations; in the latter case, *gcc* and *wolf* are short enough to run to completion. Table II also presents the branch coverage—the number of static and dynamic branches encountered—during simulation.

C. Simulation Length for Cycle-Level Simulations

Running the SPEC benchmarks to completion with the "ref" inputs on a cycle-level simulator is prohibitive for the number of simulations required by this study. Using the shorter "test" or "train" inputs, on the other hand, risks unrepresentative results, because some of these inputs are simplistic. Instead, we perform full-detail simulation for a representative, 100 million instruction segment of the program's execution with the "ref" input. Cycle-level simulations are run in a fast mode to reach the chosen simulation window. In this fast mode no microarchitectural simulation takes place; only the caches and branch predictor are updated. Table II includes the length of the fast-mode ("warmup") phase for each benchmark, including 1 million instructions in which simulation runs in full detail but statistics are not yet collected, in order to prime other structures like the instruction window.

To ensure that our chosen segment produces representative results we follow the approach described in [30]. We gather data on branch misprediction rate and cache miss rate for the entire program's execution, then identify a candidate simulation window and test its validity using cycle-level simulation. For a range of cache and branch-predictor configurations, we compare the program's IPC during the chosen 100 million instruction window to the program's IPC for a much larger instruction window. This is a one-time cost that can be amortized over an arbitrary number of studies that use these benchmarks. The comparison of IPC across multiple configurations gives us IPC *surfaces* that permit us not only to verify the IPC itself, but also the validity of the relationship among branch-predictor configuration, cache configuration, and IPC.

We have found that the single, most important factor when sampling this way is to avoid the program's initial phases, which might exhibit unusual behavior. *Compress*, for example, exhibits very different behavior for its first 1.5 billion instructions. This is solely an artifact of the SPEC95 benchmark version of *compress*; during this initial phase, the program generates the data that it will subsequently compress or decompress. The branch misprediction rate during this phase is approximately twice as high as during the rest of the program. *Perl* and *vortex* are other programs with markedly different initial phases, and most of the SPEC95 benchmarks exhibit some startup behavior.

V. Performance of Alloyed Prediction

A. *Comparison Against GAs, PAs, and Bimodal*

This section compares the three most basic types of two-level predictors described in this paper: GAs, PAs, and MAs, plus the classic two-bit or bimodal approach. The purpose of this comparison is to show the advantage of alloying as an effective implementation of two-level prediction. (Bimodal prediction is included for reference purposes.) We subsequently compare alloyed prediction to more aggressive branch-predictor configurations.

In our comparison of two-level predictors, we first compare misprediction rate and IPC for individual benchmarks; then explain alloyed prediction's superior performance.

A.1 Misprediction and IPC Comparison

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 (i.e. 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. Bit concatenation was used for the MAs history combining. The configurations chosen appear in Table III.

TABLE III

Predictor configurations used for equal-total-size comparison. "g" indicates the number of global-history bits, "p" local-history bits, and "a" address bits.

	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

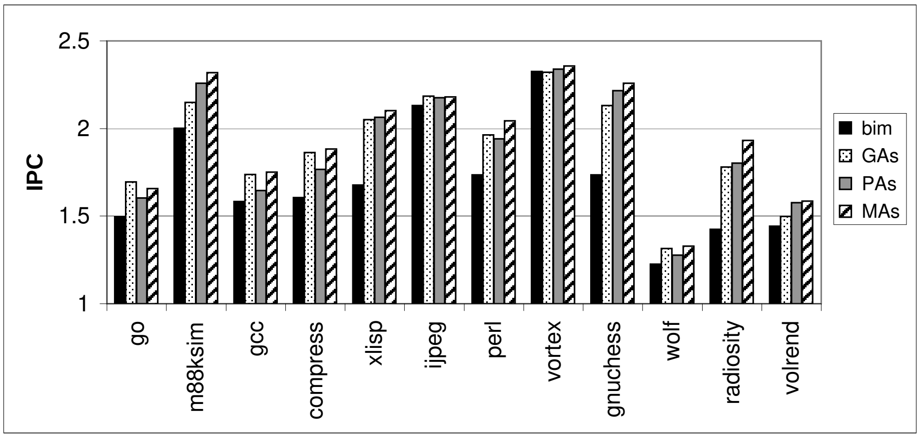

Fig. 9. Relative performance of bimodal ("bim"), GAs, PAs, and MAs for 64 Kbits total size, including a finite BHT. Taller bars represent better performance.

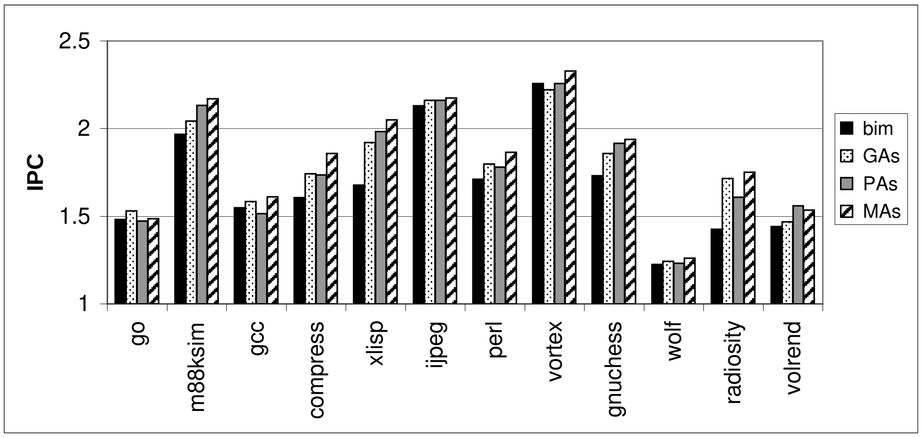

Fig. 10. Relative performance of bimodal, GAs, PAs, and MAs for 8 Kbits total size, including a finite BHT.

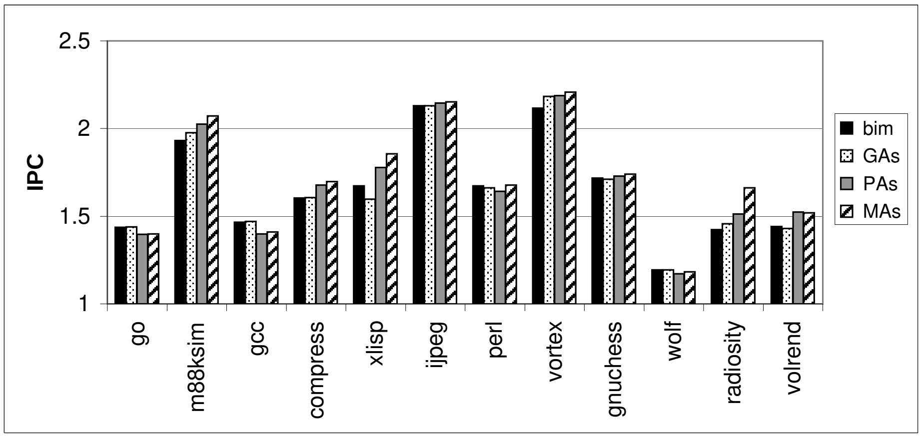

Fig. 11. Relative performance of bimodal, GAs, PAs, and MAs for 2 Kbits total size, including a finite BHT.

Figures 9–11 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 bimodal predictor of the appropriate size is shown as well. The results are for a 4-issue processor.

TABLE IV

MEAN 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
1.154	1.031	1.034	1.092	1.033	1.032	1.038	1.036	1.020

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%

As the taxonomy results and dynamic branch history preference distribution suggest, MAs consistently outperforms GAs and PAs. For many benchmarks, MAs is better by a substantial margin of 4–8% in IPC and as much as 85% in misprediction rate. 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. Table V reveals that the speedups are even better in an 8-issue configuration for 64 Kbit predictors.

These speedups resulted from substantial reductions in the misprediction rate. For some benchmarks (e.g. *m88ksim, perl*, and *vortex*), a 64 Kbit MAs *halves* the misprediction rate compared to an equivalent-area GAs. Table VI reports MAs's mean misprediction rate reduction compared to GAs and PAs. Note that the reduction in mispredictions is mostly independent of issue width.

During the exhaustive search of branch predictor configurations for the above experiments, an interesting property we observe about MAs is that it is the most robust two-level branch predictor.

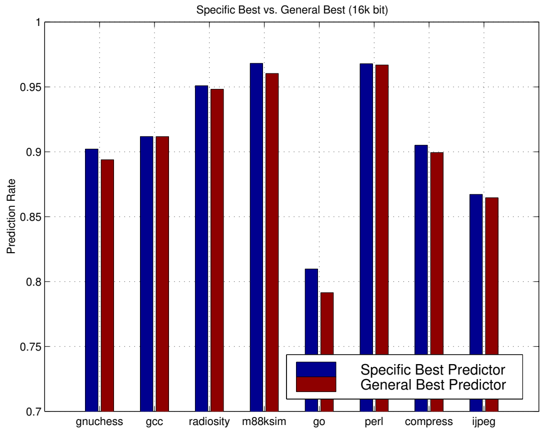

Fig. 12. Performance comparison between benchmark-specific best predictor and general best predictor among different benchmarks.

It is well known that PAs and GAs are very sensitive to history length. Different benchmarks will have the best prediction rate for different history lengths. In [19], Juan *et al.* proposed a way to dynamically adjust the history length of the GAs to achieve the best results for individual programs. Our results show that this kind of adaptation will be unnecessary for MAs.

Figure 12 collates the results from our exhaustive search and compares the best predictor configuration for each benchmark against the one configuration that is best on average across the entire benchmark suite. All predictors we tested were 16 Kbit. We explored all possible two-level predictor configurations (i.e. PAs, GAs, MAs, and we also added data for gshare) of the same size. Except for *go*, all benchmark-specific best predictors are MAs with slightly different lengths of local or global history bits, and of course the general best predictor is the MAs predictor used above. The specific best predictor for *go* is a GAs with long global history but the best specific MAs configuration for *go* gives close performance.

What the results in Figure 12 show is that alloying makes the predictor less sensitive to workload. If GAs, PAs, and gshare are the only design options, it is hard to select a single branch-predictor configuration because different benchmarks have widely varying preferences. But when MAs is a design choice, the alloying of both global history and local history eliminates much of this variability: the specific best predictor turns out to be MAs in almost all cases, and in all cases that we examined, the specific best does not have a large advantage over the general best MAs predictor. This makes MAs a robust choice across workloads and, as we saw above in Figures 9–11, across sizes.

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. In addition, MAs is often so much better than the others that a substantially smaller MAs configuration can be used. For example, compared to a 64 Kbit GAs predictor, an 8 Kbit MAs does as well or better for 5 benchmarks: *m88ksim, compress, xlisp, ijpeg*, and *vortex*. A 2 Kbit MAs does as well as the other 64 Kbit predictors for *volrend*. The IPC of an 8 Kbit MAs is within 5% of the 64 Kbit GAs for three other benchmarks. A much smaller MAs might therefore plausibly replace larger GAs predictors, and could be especially appealing for embedded or other low-cost processors.

A.2 Analysis of "Preference" for History Type

The preceding results suggest strongly that alloyed branch prediction helps both by making both history types available. To confirm this and better understand the behavior of alloying, consider again the distribution of static and dynamic branches preferring local or global history shown in Figure 3. Although a large percentage of static branches prefer either local or global history, the majority of dynamic branches are from the middle bins (*i.e.*, they use both types of history during execution). We compare the prediction rate of three same-size predictors, namely local predictor, global predictor and alloyed predictor, for dynamic branches in the same bin. The result is shown in Figure 13. As in Figure 3, the branches are classified into different bins according to the frequency each branch uses the local predictor. Branches falling in the leftmost bin prefer the global predictor, and vice versa. As expected, the global or local predictor performs best for the branches in the respective extreme bins. However, when we move to the middle bins, both the local and global predictors significantly degrade in performance. Conversely, though the alloyed predictor is not the best for the bins near either end, it keeps its performance across all bins without dramatic degradation. Recall that the branches in the middle bins occupy a substantial portion of the dynamic branch stream. This is why the alloyed predictor outperforms the other two in terms of the whole program.

A.3 Interference-Free Prediction and Upper Bounds

Recently, researchers have modeled branch prediction using Markovian chains and showed that two-level branch prediction is a simplified version of prediction by partial matching [4]. Thus, an open question at the theoretical level is how to define the states in the Markovian chains such that we will have high confidence about the transitions between states (very low/high transition prob-

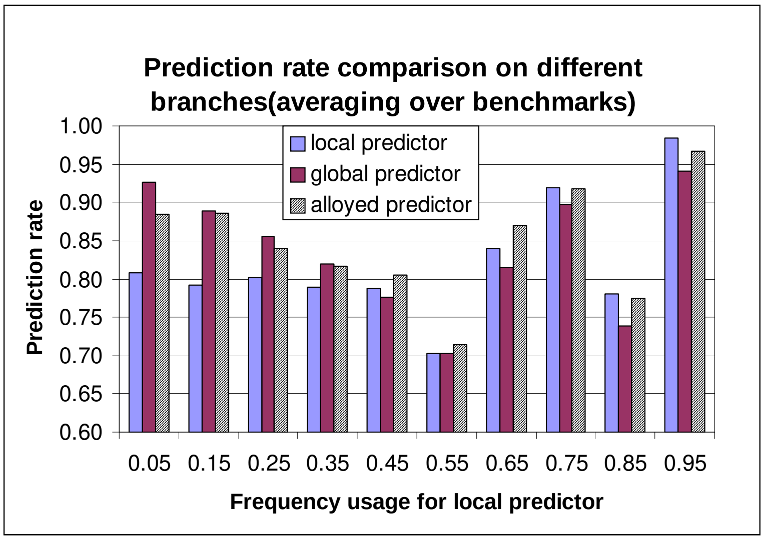

Fig. 13. Dynamic and static distribution of branches preferring local or global predictor

ability). In PAs and GAs, we try to defines the states by only using the local or global branching histories. MAs makes use of the combination of local and global to define different states in the Markovian chains. In order to see the impact of these state coding schemes on MAs prediction performance, we use an interference-free predictor (i.e. each branch has its own PHT) to implement PAs, GAs, and MAs with different history length configurations.

Figure 14 shows our simulation results. For each benchmark, the lowest curve represents the prediction accuracy of a GAs predictor with history length varying along the x-axis. Along the y-axis, the number of global-history bits is zero, and each curve's intersection with the y-axis represents the prediction accuracy of a PAs with different local history length. Other points are the prediction rate for MAs with different local and global history combinations. We can see two trends. First, as one adds more and more bits of a particular history type, diminishing returns quickly set in. Second, as a general rule, if the total history length is fixed, then combinations of local and global history usually give higher performance than only local or global history alone.

Another interesting phenomenon from Figure 14 is that, for most benchmarks, the performance gain for longer histories diminishes beyond a certain amount of history. It seems that all curves in the same benchmark will converge on the same performance limits. We believe that this is the upper bound for the Markovian chain model with states encoded only by branching history. In order to further improve predictor performance, we have to incorporate other branch or program information to encode the states in the Markovian chain model. When we compare the upper bounds shown in Figure 14 with the performance of a small predictor (about 16 Kbit), we find that,

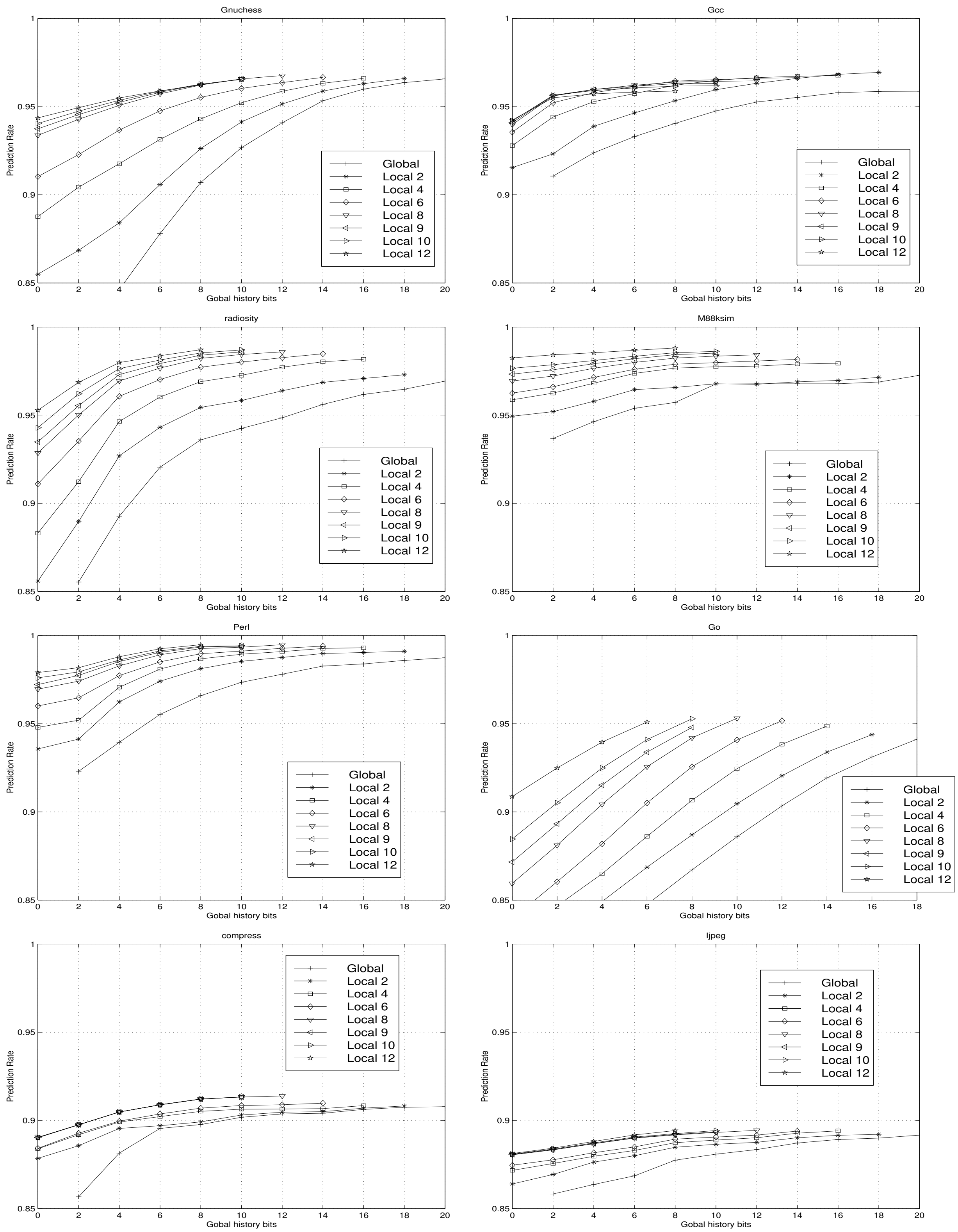

Fig. 14. Prediction rate for interference-less predictor

for some benchmarks (e.g. *Ijpeg*), the small predictor can achieve a prediction rate very close to the upper bound. However, the prediction rate for small predictors on other benchmarks (e.g. *Go*) is significantly lower than the upper bounds shown in Figure 14.

B. Comparison Against Bi-Mode 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 greatly suffer from wrong-history mispredictions and from aliasing. In the next subsection (Sec. V-C), we will evaluate alloying against hybrid prediction, which, like alloying, can attack the wrong-history problem. In this section, we evaluate alloying against bi-mode prediction, which is one of the most aggressive anti-aliasing predictor organizations proposed to date. Furthermore, since we have found that alloying is actually a generalization of bi-mode, we must determine whether bi-mode already captures all the benefits of alloying.

We compare alloying with bi-mode for different predictor sizes. In order to obtain a fair comparison, we must find the best configuration for a certain size within a wide range of different combinations of local and global histories. We again performed experiments to find the best configuration by exhaustive search. Instead of always keeping the choice predictor the same size as the direction PHT table as proposed by the original bi-mode paper [23], we allow different table sizes for the choice predictor in order to obtain more configuration combinations for the bi-mode structure. In addition, as in the original bi-mode proposal, we find that bi-mode benefits somewhat using XOR-style indexing, so we use gshare-style indexing for alloyed prediction as well. The alloyed gshare index consists of the XOR of the global history and branch-address bits; the local-history bits do not participate for timing reasons as discussed in Section III-D. We call this indexing scheme *mshare*. Finally, due to the different structures in mshare and bi-mode, we cannot always compare performance for the exact same predictor size. Instead, we plot the prediction accuracy for each configuration tested and interpolate.

Figure 15 gives the misprediction rates for mshare vs. bi-mode predictors for a range of total predictor sizes up to 150 Kbits. Given the extensive simulation requirements of these results, we confine our study to eight benchmarks and only instruction-level simulations.

In four (*gnuchess, radiosity, perl*, and *m88ksim*) out of the eight benchmarks, bi-mode is better than mshare when the predictor size is small. This is because small mshare predictors usually use two-bit local history, whereas the choice table in the bi-mode predictor uses two-bit saturating

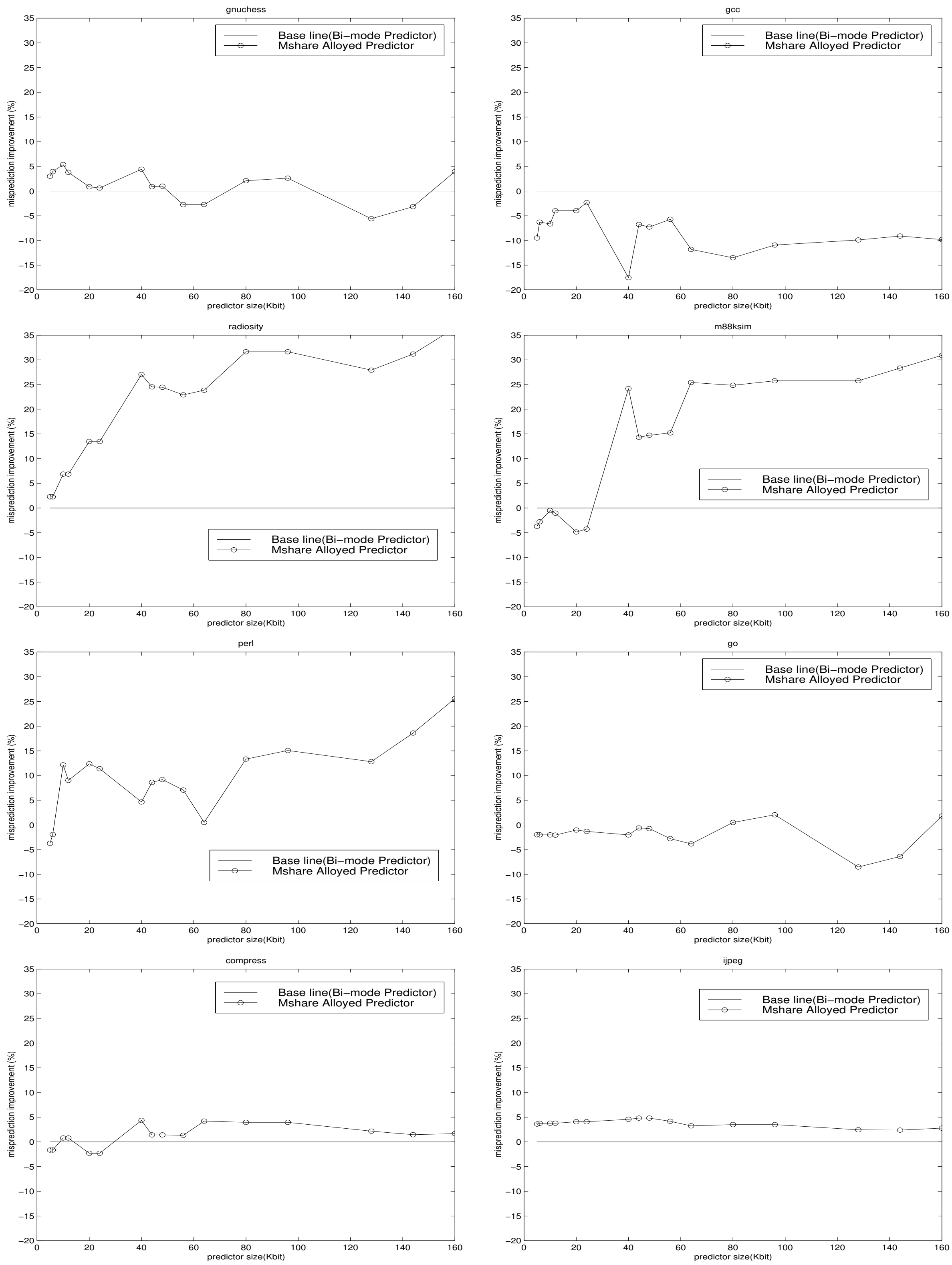

Fig. 15. Misprediction rate comparison between mshare and bi-mode predictor with different predictor size. Each graph plots the gain/loss in misprediction rate conferred by mshare relative to bi-mode (the baseline in each graph). In each case, the configuration used is the one that is best overall for our eight benchmarks.

counters that serve to accumulate longer local history information. When the predictor size increases, the mshare predictor performs better than the bi-mode predictor in six out of eight benchmarks. This is because the mshare can hold longer local history. The bi-mode predictor performs worse in this case because the local history information contained in the two-bit counter does not do as good a job of distinguishing among branch outcomes.

We also found that, for *gcc* when the mshare predictor is forced to use only two-bit local history (even in large predictor sizes), it closely tracks the performance of the bi-mode predictor, for which bi-mode is the benchmark-specific best predictor. Figure 16 shows this result. It can be inferred that *gcc* in an alloyed configuration (regardless of whether that alloyed configuration is mshare or bi-mode) does best with approximately two bits of local history and as long a global history as possible.

To summarize our comparison of alloyed prediction and bi-mode prediction, we plot in Figure 17 the average benefit of the general-best alloyed predictor configuration against the general-best bi-mode predictor configuration. This is simply an average of the same misprediction rates that were used to derive Figure 15. We see that for the very smallest predictors—less than 8 Kbits—bi-mode confers a slight benefit due to the use of saturating counters. Otherwise, removing the restriction of saturating two-bit counters and allowing the alloyed predictor to use more local history bits confers a significant advantage.

C. Comparison Against Hybrid Prediction

Hybrid predictors that combine both a global history and a local-history component can also attack wrong-history mispredictions and eliminate many of the same mispredictions as alloyed prediction (whether MAs, mshare, or bi-mode). But 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. On the other hand, hybrid prediction is prone to mis-selection, *i.e.*, failures in the selector, which alloyed prediction will not suffer.

Again, a fair comparison of alloyed prediction against hybrid prediction requires finding the best predictor for each size by testing a wide range of component sizes, selector sizes and history lengths. This also requires choosing between a dynamic and a static selector for the hybrid predictor. Static selection [9] reduces the area spent on the selector and permits selective updates of just the component used by each branch. Both of these considerations reduce conflicts. However, static selection also eliminates the hybrid predictor's ability to dynamically migrate a branch

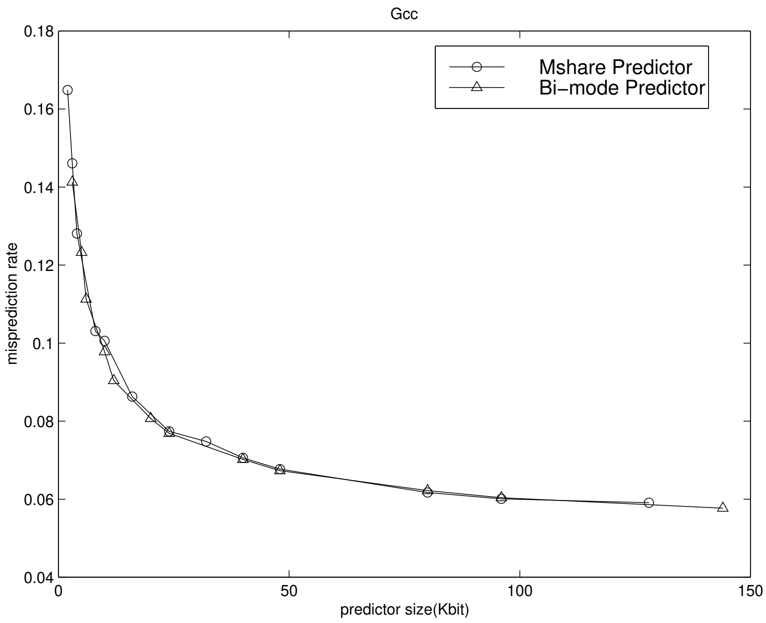

Fig. 16. Misprediction rate for the benchmark gcc, comparing between mshare and bi-mode predictors, as a function of different predictor size when mshare is forced to only use two-bit local history.

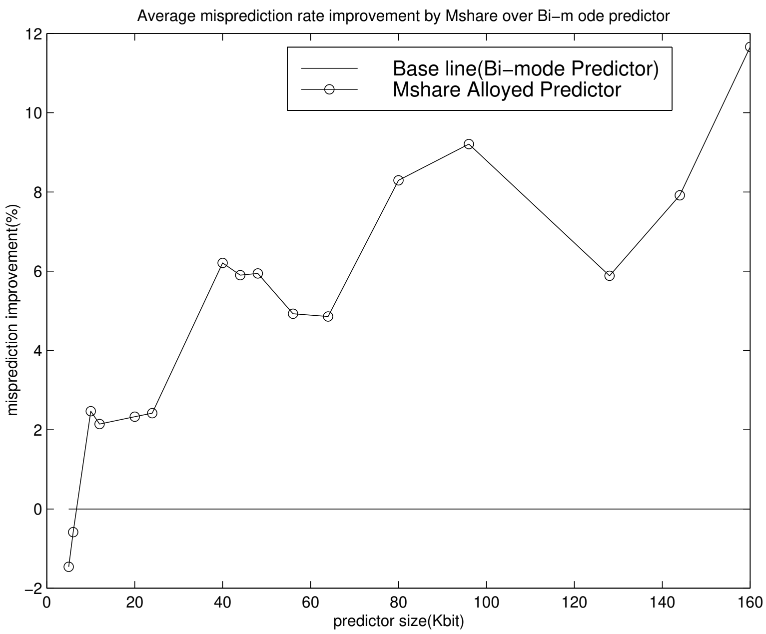

Fig. 17. Average misprediction rate improvement by mshare over bi-mode predictor at different predictor size

between components in response to conflicts or changes in the branch's behavior, permanently assigning each branch to use either global or local history. We found in Section II-B that most programs indeed have a significant number of branches that do not have a strong preference for local vs. global history, and these branches are penalized in a static scheme. As a result, we found that static selection is rarely superior to dynamic selection, even for small predictors.

As in the previous comparison of mshare and bi-mode, we exhaustively search possible hybrid-predictor configurations. We confine ourselves to dynamic selection but allow the size of the selector and predictor components to vary with respect to each other. Also as before, Figure 18 compares general-best mshare and hybrid predictors by showing how much advantage/disadvantage mshare confers at different predictor sizes. The results are averaged in Figure 19. At small sizes, mshare performs much better than hybrid. This is because the selector table in the hybrid predictor consumes a large portion of the total size, and the PHTs in the hybrid predictor are too small. When the predictor size increases, hybrid predictors have larger PHTs, and the hybrid prediction's ability to dynamically switch between the GAs and PAs components begin to show hybrid's ability to attack the wrong history problem, so the performance of the two grows closer.

Nevertheless, across a range of sizes, alloyed prediction confers substantial advantage for several benchmarks while hybrid prediction is not substantially better for any benchmark except *m88ksim* at small sizes. This is due to *m88ksim*'s strong preference for local history, which is sufficiently strong that even a small hybrid predictor can satisfy it. Hybrid is also slightly better for *gcc* at large sizes, because *gcc* prefers long global history and a large hybrid predictor can provide more global history than can a large alloyed predictor.

For three other benchmarks, *go, compress*, and *ijpeg*, there is no clear trend as to which predictor is better. For *gnuchess*, mshare is better for most sizes. And for the remaining two benchmarks, *radiosity* and *perl*, mshare is dramatically better. Overall, Figure 19 shows that alloying is at least 4% better, and at sizes below 16 Kbits, as much as 12–14% better.

To summarize the previous results, alloyed prediction is generally a better choice than hybrid prediction. Although in some cases it confers little advantage or is slightly worse when compared to a hybrid predictor, even in these cases it is a consistently reliable performer; and for other benchmarks alloying confers large improvements—as much as a 30–40% reduction in mispredictions. We therefore argue that this robust behavior makes alloyed prediction an attractive choice across a wide range of sizes.

VI. 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

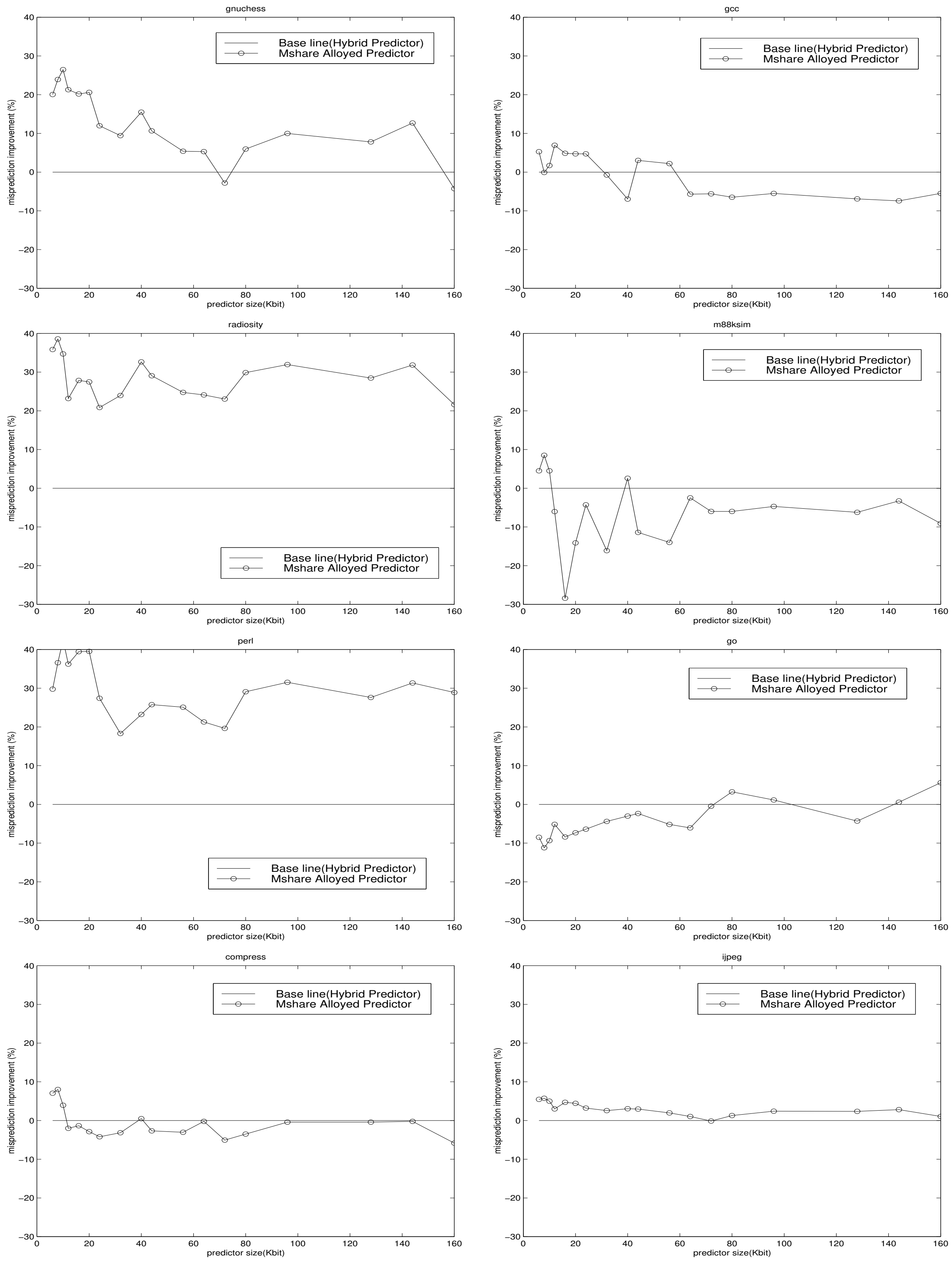

Fig. 18. Misprediction rate comparison between mshare and hybrid predictor with different predictor size

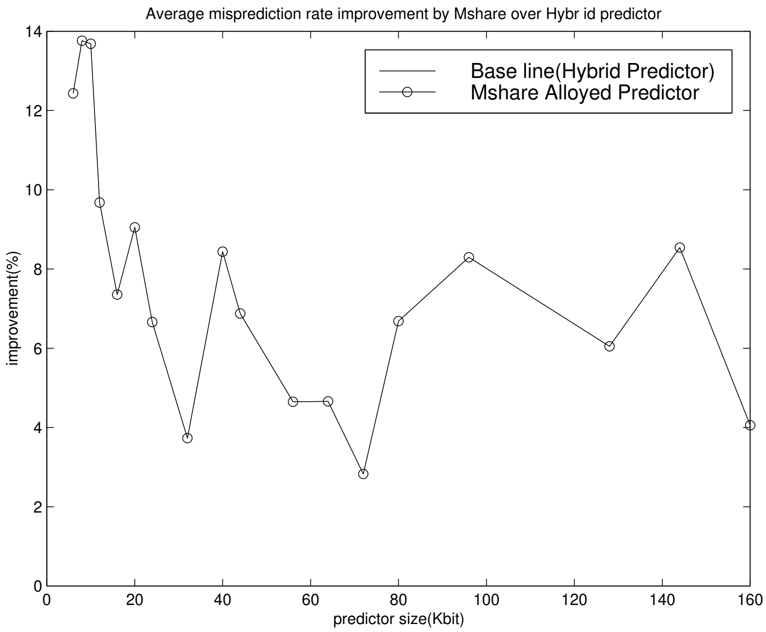

Fig. 19. Average misprediction rate improvement by mshare over hybrid predictor at different predictor size

et al. [26] observed that a tradeoff exists between including history and address bits in the PHT index, and introduced GAs. Yeh and Patt [41] 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.* [27] also observed the value of sacrificing some history bits for address bits, especially for global-history predictors, and in addition found that XORing history bits and address bits, as in *gshare* [24], provides little benefit.

Other researchers have described a variety of more aggressive techniques for reducing conflict mispredictions. Sprangle *et al.* described an *agree* predictor in [36]. Michaud *et al.* [25] introduced 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.* [23] and Eden and Mudge [5] observed that conflicting substreams may be strongly biased, just in opposite directions. They described the bi-mode predictor [23] and later a YAGS predictor [5] that use a meta-predictor to separate branch substreams of opposite bias. The YAGS predictor extends the bi-mode organization by learning when branches disagree with bi-mode's bias. Klauser *et al.* [22] combine traditional branch-prediction methods like gshare, bi-mode, or hybrid, with confidence prediction [13] to determine when the normal prediction should be inverted. They find that this is effective at correcting destructive conflicts.

It is important to note that alloying can easily be applied 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.

Yet other work has focused on characterizing why mispredictions happen in two-level predictors, but these continue to focus on PHT interference. Talcott *et al.* [38], Sprangle *et al.* [36], Young *et al.* [42], and Chang *et al.* [2] characterized PHT interference and showed that while significant amounts of both constructive and destructive interference occur, the destructive interference consistently dominates. Evers *et al.* [6] and Juan *et al.* [19] discussed the importance of training time. 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.

Hybrid prediction was originally proposed by McFarling [24]. 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.

Other recent work has proposed new predictor types that target branches that are not handled well by two-level predictors. Examples include perceptrons [16], boolean equations [14], and fourier-based prediction [20]. While none of these techniques is likely to work well as a stand-alone predictor, it is likely that they would make an excellent combination with an alloyed predictor as part of a new type of hybrid design.

VII. Conclusions and Future Work

This paper has shown the *alloying* of global and local history to be a robust way to predict conditional branches. 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 work has mainly focused on reducing conflict mispredictions due to aliasing in the pattern history table. Conflicts are clearly important, but this paper has shown the importance of providing both global and local history to avoid "wrong-history" mispredictions, which are often the most frequent misprediction type, comprising up to 50% of the total mispredictions. Hybrid predictors attack these wrong-history 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, attacking wrong-history mispredictions without subdividing the available area into multiple predictors. Alloying also reduces PHT aliasing, because branches that alias with one type of history are often distinguished by the other type of history.

Specifically, this paper:

- Shows how to combine local and global history into a two-level predictor structure without serializing the lookup of the BHT and PHT.
- Shows that an alloyed predictor substantially outperforms other two-level organizations that use only one type of history (global or local) and also substantially outperforms bimodal (two-bit) prediction.
- Observes that bi-mode prediction is a subset of a larger class of techniques for mixing global and local history that we broadly refer to as alloyed predictors. The saturating two-bit "choice" counters in the bi-mode organization are actually tracking local history.
- When comparing the bi-mode organization to the more general MAs and mshare configurations, we find that the saturating two-bit choice counters confer some advantage for small alloyed organizations, but that otherwise it is beneficial to use longer local histories as the MAs and mshare predictors do.
- Large hybrid predictors also do a good job of attacking wrong-history mispredictions, but when comparing alloyed prediction against hybrid prediction, alloyed prediction still perform 3%–14%.

Overall, our results show alloyed prediction to be an approach that yields robust results for a wide range of benchmarks and predictor sizes.

In terms of future work, it is important to note that we have only examined a few ways to implement alloying. Just as bi-mode, MAs, and mshare are variations of the alloyed approach, other ways to combine global and local history—and possibly other types of data as well, like data values or confidence bits—may give further benefits. Other appealing areas of investigation are adaptive alloyed predictors that dynamically adjust the mixture of bits, and the combination of alloyed predictors with other novel prediction techniques to form new hybrid predictors. And as always, further research to understand the true limitations of history-based branch predictors and the sources of mispredictions is always helpful.

ACKNOWLEDGMENTS

This work was funded in part by the National Science Foundation under grants nos. CCR-0082671 and CCR-0105626, NSF CAREER MIP-9703440, a grant from Intel MRL, and by an exploratory grant from the University of Virginia Fund for Excellence in Science and Technology, and an NDSEG Graduate Fellowship. We would like to thank Doug Clark, Margaret Martonosi, Mikko Lipasti, and Tom Conte for their helpful comments.

REFERENCES

[1] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[2] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *Proceedings of the 1996 International Conference on Parallel Architectures and Compilation Techniques*, pages 48–57, Oct. 1996.

[3] P.-Y. Chang, E. Hao, and Y. N. Patt. Alternative implementations of hybrid branch predictors. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 252–57, Dec. 1995.

[4] I.-C. Chen, J. T. Coffey, and T. N. Mudge. Analysis of branch prediction via data compression. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 128–37, Oct. 1996.

[5] A. N. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, 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 *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, 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 *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 52–61, June 1998.

[8] P. N. Glaskowsky. Pentium 4 (partially) previewed. *Microprocessor Report*, pages 1, 11–13, Aug. 2000.

[9] D. Grunwald, D. Lindsay, and B. Zorn. Static methods in hybrid branch prediction. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pages 222–29, Oct. 1998.

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

[11] A. Hartstein and T. R. Puzak. The optimum pipeline depth for a microprocessor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 7–13, May 2002.

[12] M. S. Hrishikesh et al. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, May 2002.

[13] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning confidence to conditional branch predictions. In *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 142–52, Dec. 1996.

[14] D. A. Jiménez, H. L. Hanson, and C. Lin. Boolean formula-based branch prediction for future technologies. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 97–106, Sep. 2001.

[15] D. A. Jiménez, S. W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 67–77, Dec. 2000.

[16] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, Jan. 2001.

[17] 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.

[18] S. Jourdan, J. Stark, T.-H. Hsing, and Y. N. Patt. Recovery requirements of branch prediction storage structures in the presence of mispredicted-path execution. *International Journal of Parallel Programming*, 25(5):363–83, Oct. 1997.

[19] T. Juan, S. Sanjeevan, and J. J. Navarro. Dynamic history-length fitting: A third level of adaptivity for branch prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 156–66, June 1998.

[20] M. Kampe, P. Stenström, and M. Dubois. The FAB predictor: Using fourier analysis to predict the outcome of conditional branches. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 223–232, Feb. 2002.

[21] R. E. Kessler, E. J. McLellan, and D. A. Webb. The Alpha 21264 microprocessor architecture. In *Proceedings of the 1998 International Conference on Computer Design*, pages 90–95, Oct. 1998.

[22] A. Klauser, S. Manne, and D. Grunwald. Selective branch inversion: Confidence estimation for branch predictors. *International Journal of Parallel Programming*, 29(1):81–110, Feb. 2001.

[23] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 4–13, Dec. 1997.

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

[25] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 292–303, June 1997.

[26] S.-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 76–84, Oct. 1992.

[27] S. Sechrest, C.-C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 22–32, May 1995.

[28] K. Skadron and P. S. Ahuja. Hydrascalar: A multipath-capable simulator. In *Newsletter of the IEEE Technical Committee on Computer Architecture*, pages 65–70, Jan. 2001.

[29] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Improving prediction for procedure returns with return-address-stack repair mechanisms. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pages 259–71, Dec. 1998.

[30] 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 Transactions on Computers*, 48(11):1260–81, Nov. 1999.

[31] K. Skadron, D. W. Clark, and M. Martonosi. Speculative updates of local and global branch history: A quantitative analysis. *Journal of Instruction-Level Parallelism*, Jan. 2000. (http://www.jilp.org/vol2).

[32] K. Skadron, M. Martonosi, and D. W. Clark. A taxonomy of branch mispredictions, and alloyed prediction as a robust solution to wrong-history mispredictions. In *Proceedings of the 2000 International Conference on Parallel Architectures and Compilation Techniques*, pages 199–206, Oct. 2000.

[33] J. E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, pages 135–48, May 1981.

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

[35] E. Sprangle and D. Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 25–34, May 2002.

[36] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 284–91, June 1997.

[37] Standard Performance Evaluation Corporation. SPEC CPU95 Benchmarks. http://www.specbench.org/osg/cpu95.

[38] A. R. Talcott, M. Nemirovsky, and R. C. Wood. The influence of branch prediction table interference on branch prediction scheme performance. In *Proceedings of the 1995 International Conference on Parallel Architectures and Compilation Techniques*, pages 89–96, June 1995.

[39] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 345–56, June 1995.

[40] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[41] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 257–66, May 1993.

[42] C. Young, N. Gloy, and M. D. Smith. A comparative analysis of schemes for correlated branch prediction. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 276–86, June 1995.