A Taxonomy of Branch Mispredictions, and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions

Kevin Skadron
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904
skadron@cs.virginia.edu

Margaret Martonosi Douglas W. Clark
Depts. of Electrical Engineering and Computer Science
Princeton University
Princeton, NJ 08544
mrm@ee.princeton.edu *doug@cs.princeton.edu*

Abstract

The need for accurate conditional-branch prediction is well known: mispredictions waste large numbers of cycles, 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. Based on these observations, this paper proposes* alloying *local and global history together in a two-level branch predictor structure. This simple technique, a generalization of the* bi-mode *predictor, attacks wrong-history mispredictions by making both global and local history simultaneously available. Unlike hybrid prediction, however, alloying gives robust performance for branch-predictor hardware budgets ranging from very large to very small. Finally, this paper shows that* individual *branch references can also suffer wrong-history mispredictions as they alternate between using global and local history, a phenomenon that favors dynamic rather than static selection in hybrid predictors.*

1. Introduction

The question of how better to predict the direction of conditional branches has received intense study in recent years. Two-level [11, 22] and hybrid [10] 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. This paper shows that 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 smaller when aggressive conflict-reduction techniques are applied. A complete elimination of conflicts therefore leaves many or most mispredictions remaining to be solved.

Further reductions in conflict mispredictions are indeed becoming difficult, and prediction accuracies still lie only in the 90–97% range. 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, especially the category of *wrong-history mispredictions*. These arise when the type of history tracked by a two-level predictor—either global or local history—is the wrong type of history for that branch. The paper describes *alloyed branch prediction*—a generalization of the *bi-mode* branch predictor proposed by Lee, Chen, and Mudge [9]—as an attractive way to attack this category of mispredictions. Finally, the paper shows that individual branches dynamically vary between needing local and global history, and demonstrates that static selection in a hybrid predictor is therefore undesirable. Alloying, on the other hand, allows branches to see both global and local history simultaneously. Alloying has the further advantage over conventional hybrid branch predictors that it does not subdivide the available branch-prediction hardware into distinct and much smaller—and thus less effective—components.

As we developed our taxonomy, the category of wrong history mispredictions suggested the development of the alloyed predictor. We found that using an alloyed predictor permitted us to add a category to our taxonomy. We therefore describe the alloyed predictor first, so that we can use the alloyed predictor in the rest of the paper as we develop the taxonomy.

The next section presents the simulation methodology used in this paper. Next, section 3 describes alloyed prediction and presents a brief evaluation of its performance. Section 4 then develops the taxonomy (incorporating alloying) and quantifies the importance of wrong-history mispredictions, and Section 5 further explores the issue of wrong-history mispredictions and how it affects conventional hybrid predictors. It shows that not just static branch instructions, but dynamic branch references can also suffer wrong-history mispredictions. Finally, Section 6 describes related work, and Section 7 concludes the paper.

2. Simulation and Benchmark Details

2.1. 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, our modified, multipath-capable version of SimpleScalar 2.0's *sim-outorder* [1]. HydraScalar has been configured to approximately model an Alpha 21264 [7]. 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. The branch history is updated speculatively at fetch time with suitable repair mechanisms [14], and HydraScalar models multiple layers of misprediction. The branch misprediction latency is 7 cycles. The taxonomy measurements use a modified version of SimpleScalar's instruction-level *sim-bpred* simulator.

2.2. Benchmarks

These evaluations use not only the SPECint95 benchmarks [19], but also four other primarily integer benchmarks. Table 1 summarizes the benchmarks' characteristics. All are compiled

[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.

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.

	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

Table 1. Benchmark summary.

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 [20]; *wolf* is the timberwolf circuit router and comes from Smith’s Unix-Utils benchmark suite [18], and 1.7% of its instructions are floating-point operations. *Radiosity* and *volrend* were chosen from the SPLASH2 suite [21] of parallel applications for shared memory because these two have significant misprediction rates.

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 [13]. The simulator therefore begins gathering statistics much later in the program. During the preliminary phase, branches and memory references are still presented to the simulator, warming up the predictor and caches. After the warmup phase (whose duration is shown in Table 1) 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 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.

3. Alloying: Description & Performance

3.1. Hybrid Predictors

Hybrid predictors [10] are one way to attack the 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 therefore available [2]. 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 1 presents a high-level schematic of a hybrid predictor that combines global and local prediction components.

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 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.

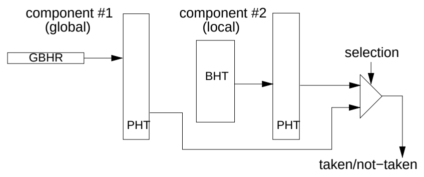

Figure 1. The organization of a hybrid predictor with two different components. (The left-hand component is a global-history predictor, and the right-hand component is a local-history predictor.) The selector can be dynamic, requiring a meta-predictor structure, or static, in which case each branch is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

3.2. 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 [7]. But 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. Despite these constraints, smaller branch-prediction environments still require the best branch prediction available, because prediction accuracy remains a powerful lever over performance. Better prediction accuracy also reduces power wasted on mis-speculated computation. One might think a simple bimodal[2] organization would be the best choice. The data from Section 3.6 and [15] show otherwise.

3.3. Alloyed Predictors

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

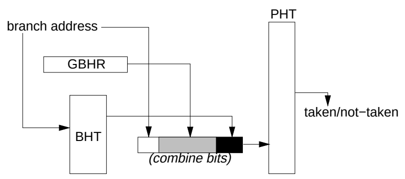

Figure 2. The organization of a two-level predictor with an alloyed index. This “MAs” predictor combines local history from the per-branch history table (BHT) and global history from the global branch-history register (GBHR) with some address bits to compose the PHT index.

Alloying 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 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.

[2]A bimodal or “2-bit” predictor—proposed by Smith in [17]—is just a bare PHT, indexed by branch address.

We call the organization shown in this figure *MAs*, because it resembles GAs and PAs predictors[3] [22] 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 can be distinguished using the other kind of history.

3.4. Alloyed vs. Bi-Mode Prediction

Alloying is a generalization of the *bi-mode* predictor proposed by Lee, Chen and Mudge [9] and shown in Figure 3. As described in [9], the bi-mode predictor seems quite different from the MAs predictor in Figure 2. A careful rearrangement, however, shows the similarity. The bi-mode predictor 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 [10] manner of xor'ing 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 PHTs the branch should use.

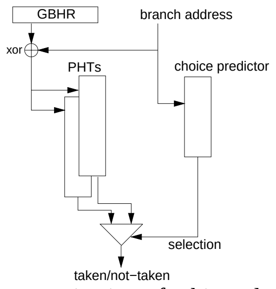

Figure 3. The organization of a bi-mode predictor. The "choice predictor" uses two-bit counters to learn for each branch whether it is biased toward taken or not taken. This value is then used to assign the branch to one of the two PHTs.

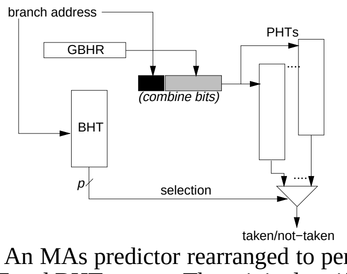

Figure 4. An MAs predictor rearranged to permit simultaneous PHT and BHT access. The original, unified PHT is broken into 2^p separate tables, all accessed simultaneously. The p local-history bits are then used to select which value to use for the final prediction.

[3] 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) [22].

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. In particular, if the bi-mode predictor uses bit-concatenation rather than xor'ing, bi-mode is almost exactly the same as an MAs predictor with one bit of local history.

3.5. Further Alloying Considerations

The MAs organization, as described, may 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 concatenated with the global-history and address bits. Only then could the PHT be accessed. Fortunately, 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 3). The local history bits are then used as the selector on a multiplexor that chooses the outcome from the appropriate table. This organization is shown in Figure 4. It permits the PHT and BHT lookups to proceed in parallel. Such an organization should be feasible for most MAs configurations, since we never found an MAs organization that needed more than 4 local-history bits. While a 16-way multiplexor will have a non-trivial delay in its own right, this delay should be less than that of a table lookup. A further consideration is that the multiple simultaneous table accesses will dissipate somewhat more power than the single access to one large table. Of course, the roles of lookup time and power cannot be evaluated in the experimental framework used for this work.

Despite the concerns over access time, MAs has two major virtues that make it attractive: it reduces wrong-history mispredictions, and it avoids subdividing the branch-prediction hardware into multiple components that may individually be too small to predict effectively. MAs thus gives robust prediction accuracy for a range of sizes. Conventional hybrid predictors perform poorly at small sizes, and conventional two-level predictors reach a domain of diminishing returns too early and therefore perform poorly at large sizes.

As mentioned before, other anti-aliasing schemes have been proposed that outperform GAs and PAs, although often at the cost of additional complexity. We restrict our evaluation of alloying to GAs, PAs, and MAs predictors, and hybrid predictors that use them as components. Comparing predictors that use the same conflict-reduction technique keeps our experiments fair, and should give an accurate picture of alloying's usefulness. Alloying is certainly not restricted to an MAs configuration. 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. A final important consideration for this paper is that "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.

3.6. Performance of Alloyed Prediction

In the interests of space, a detailed evaluation of the performance of alloyed prediction is left to a technical report [15]. That document presents per-benchmark comparisons of MAs against bimodal, GAs, PAs, and hybrid branch predictors, giving both prediction-rate and IPC data. Here we only summarize the results to show that MAs successfully attacks wrong-history mispredictions.

To get the best comparison for each predictor size, the configurations that perform best overall for the entire benchmark suite

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

Table 2. 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			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 3. Predictor configurations used for 64 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	

Table 4. Predictor configurations used for 8 Kbit dynamic and static hybrid predictors.

64 Kbits				8 Kbits				2 Kbits			
bimodal	GAs	PAs	hybrid	bimodal	GAs	PAs	hybrid	bimodal	GAs	PAs	hybrid
1.154	1.031	1.034	1.000	1.092	1.033	1.032	1.029	1.038	1.036	1.020	na

Table 5. Mean speedup of MAs over each listed predictor organization for a 4-issue processor.

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

Table 6. Mean reduction in misprediction rate achieved by MAs.

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 (plots of this design space for *gcc* and *m88ksim* can be found in [16].) 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 GAs, PAs, and MAs configurations chosen appear in Table 2. The hybrid configurations chosen appear in Tables 3 and 4.[4]

Comparison Against Two-Level Predictors. Table 5 reports the average speedup on a 4-issue, out-of-order processor for MAs compared to each alternative: GAs, PAs, and bimodal. Unlike the taxonomy results in Section 4, these results do use a finite BHT. A bimodal predictor of the appropriate size is included to serve as a reference. We also evaluated *gshare*-style [10] versions, where the history and address strings are xor'd together. Like Sechrest *et al.* [12], 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 benchmarks like *m88ksim*, *perl*, and *vortex*, a 64 Kbit MAs halves the misprediction rate compared to an equivalent-area GAs! Table 6 reports how much overall MAs reduces the misprediction rate compared to GAs and PAs. Note that the reduction in mispredictions is mostly independent of issue width.

Comparison Against Hybrid Prediction. 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. Here we complete our brief evaluation of MAs by comparing it against a dynamic-selection hybrid branch predictor [10]. The data are included in Table 5. We only examine configurations of 64 Kbits and 8 Kbits; hybrid prediction is not feasible at 2 Kbits, because the components would simply be too small. Indeed, our data show that 8 Kbits is also too small for a hybrid predictor. An alloyed predictor, in contrast, works well even down to 2 Kbits.

At 64 Kbits, hybrid prediction and MAs provide equivalent performance: the average speedup of MAs over hybrid prediction for all the benchmarks is 1.0, and hybrid never outperforms MAs by more than 1.2%. Both organizations do a good job of eliminating wrong-history mispredictions. Hybrid performs well because 64 Kbits is enough area to subdivide into 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, as high as 8.5%. The exception is *go*, where hybrid is 1.3% better. The overall speedup for MAs compared to hybrid is 2.9%. This seems like a small speedup, but this corresponds to an average 15% reduction in mispredictions. MAs does better at small sizes like 8 Kbits, because a hybrid predictor—whether using dynamic or static selection—simply does not have enough area to subdivide into smaller components.

In summary, MAs performs at least as well as hybrid prediction for large hardware budgets, and outperforms hybrid prediction for smaller hardware budgets. Overall, alloying's robust performance makes it attractive for a range of processors, from high-performance processors with large branch-prediction budgets to small embedded processors.

4. A Taxonomy of Mispredictions

A taxonomy of mispredictions has three virtues. It shows the relative importance of different misprediction types. It permits designers to tailor branch-prediction solutions to individual misprediction types—a divide-and-conquer approach—rather than devise

[4]While exploring configuration options for hybrid prediction, we made sure to test hybrid organizations that use a simple, 2-bit, bimodal structure. 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.

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, and this in turn permitted us to extend the taxonomy. We feel this taxonomy is perhaps the most important contribution of this paper.

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 [4]. 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.

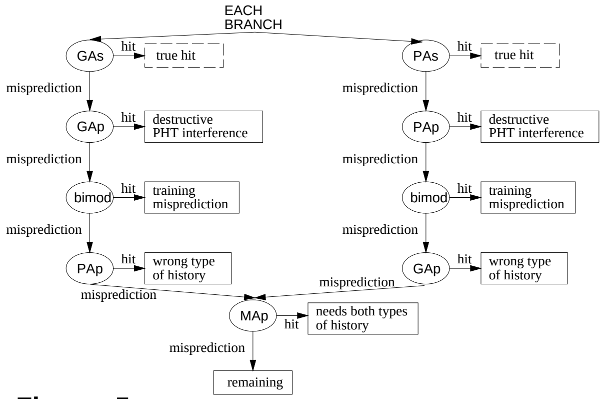

Figure 5. A flowchart depicting how the taxonomy categorizes misprediction types. Each dynamic branch flows down both sides until it is either categorized or falls through.

Figure 5 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 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.

We by no means claim the taxonomy is comprehensive: the included categories can presumably be refined, and it lacks some obvious categories: update timing [14], history length, and history pollution [5]. 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. We expect that readers will find many ways to improve this taxonomy, and this is exactly why we seek to disseminate it.

Yet even this simple taxonomy is an important contribution. It shows the importance of wrong-history mispredictions. It shows the risk of continuing to focus on conflict mispredictions. It has helped us discover alloying. And most importantly, 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.

4.1. 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 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. This also provides better comparability of GAs and PAs results.

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. First eliminate conflict mispredictions. Then training-time mispredictions occur when the main branch predictor fails but an idealized bimodal predictor succeeds.

It may seem odd for a simple bimodal predictor to follow a global-history predictor in our cascade of tests. But recall that we are not comparing the two. We only use the bimodal predictor to indicate if a branch reference could be predicted by some very simple organization. The assumption is that if a branch mispredicts using global history, but predicts correctly in the bimodal organization, the branch is predictable; the global history predictor just has not yet learned its behavior. On the other hand, if not even a simple predictor can predict a reference, then the problem is not training time.

This procedure admittedly neglects the time it takes the bimodal predictor to train. Yet a bimodal predictor is fast-training, and so we feel it provides a good estimate of the effect of training-induced mispredictions. A better method for measuring training mispredictions would be a clear contribution to this taxonomy.

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. Unfortunately, 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 therefore find that the predictor provides the wrong type of history.[6] Evers *et al.* showed this to be important in [5]. 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-

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

[6]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.

	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

Table 7. Predictor configurations used for taxonomy measurements.

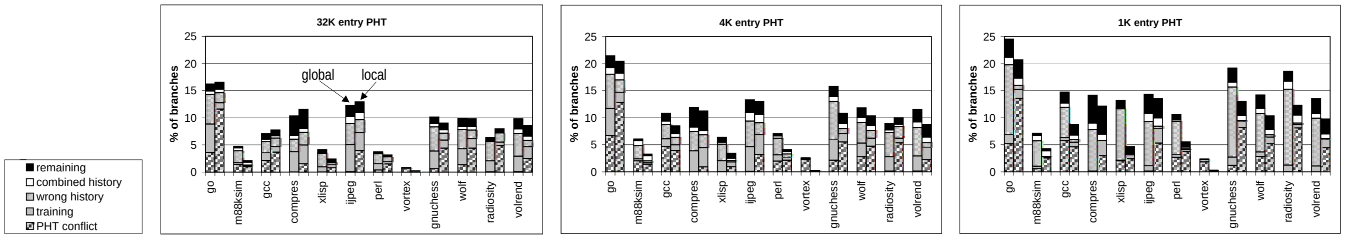

Figure 6. Breakdown of misprediction types for GAs and PAs with 32K-entry, 8K-entry, and 4K-entry PHTs and an interference-free BHT. KEY: For each benchmark, the left-hand bar represents GAs, and the right-hand bar PAs. Shorter bars mean fewer mispredictions.

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.

A possible drawback of our approach 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 some more idealized organization. At the limit, one might consider infinite history or prediction by partial matching, but this would not measure wrong-history, but rather the intrinsic predictability of the branch. Our approach characterizes the degree to which a particular history length produces wrong-history mispredictions and a different history type of the same length could remove those mispredictions.

Having seen how important wrong-history mispredictions are, we were motivated to explore ways to make both types of history available, and this led us to develop alloyed prediction.

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 alloyed predictor like the one described in the next section. The best method for measuring these mispredictions that we have been able to devise is to use an alloyed predictor: in particular, an MAp predictor using an alias-free PHT, just as with GAp and PAp. This maintains our "cascade" of tests and continues excluding conflict mispredictions. Mispredictions which do not fall into the preceding categories and which an MAp organization eliminates are combined-history mispredictions. 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* mispredictions are either inherently difficult to remove, or fall into a category not yet included in the taxonomy.

4.2. Taxonomy Results

Figure 6 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. 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 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 configuration that performs best overall for the entire benchmark suite is the one used for the experiments. That configuration is reported in Table 7. MAp is included to determine the impact of combined history.

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 6, 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 mispredic-

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

tions 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.

5. Static vs. Dynamic Choice

This section extends the information from the taxonomy and shows that dynamic references by the same branch instruction can also suffer wrong-history mispredictions—that is, some *individual* branches dynamically alternate between needing global history and needing local history.

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.* [6] 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 a static selector permanently assigns each branch to one or the other, and so penalizes any branches that alternate between history types, while dynamic selection can accommodate such alternation. The taxonomy's wrong-history data in Figure 6 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 interference-free BHTs, an 8K-entry PHT for PAs, and a 16K-entry PHT for GAs. We find that a significant number of individual branches do indeed switch between using global and local history.

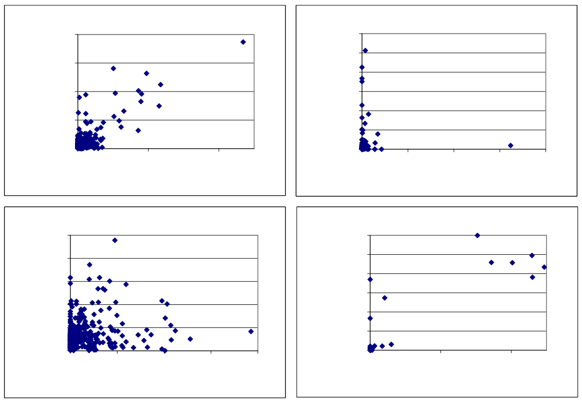

Figure 7. X-Y scatter plots showing, for each branch site, the number of times that a static branch site needs a global predictor or a local predictor.

Figure 7 presents X-Y scatter graphs for four benchmarks, *go*, *m88ksim*, *gcc* and *compress*. Each point represents one branch site; only branches executed 100,000 times or more are plotted. The graph's x-coordinate gives the number of times that a particular branch can only be predicted with global history; its y-coordinate gives the number of times that a branch can only be predicted with local history. Branches that consistently use one component lie on one or the other axis. Frequency counts were plotted rather than percentages, because this format reflects how often a branch executes (less-frequently executed branches lie closer to the origin no matter what their behavior).

For all four benchmarks, 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. *Go* and *gcc* have many such branches; *compress* has fewer, but they execute a huge number of times. *M88ksim*'s branches, on the other hand, need only one or the other type of history. Further data can be found in [8].

These data show that for the most part, individual branches do alternate between using global and local history. While some branches might conceivably change between history types just once, preliminary measurements suggest that the frequency of this alternation is rapid for most branches. We are unaware of any prior recognition or characterization of this "individual-branch" wrong-history effect. This data indicates that, 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. The configurations eventually chosen are the ones used in Section 3.6 (see Tables 3 and 4). Note that the equal-area comparison includes the presence/absence of the dynamic selector, so that dynamic selection is appropriately penalized for the area required by its selection table. Indeed, to further benefit static selection, we did not cross-train, using the same input data for measurement as for assigning the static selection.

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. Yet despite our attempts to favor static selection, 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.* [6]. 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.

6. Related Work

A great deal of literature focuses on characterizing why mispredictions happen in two-level predictors. Most, however, focus on PHT interference. For example, Young *et al.* [23] characterized PHT interference, and showed that while both significant amounts of both constructive and destructive interference occur, the destructive interference consistently dominates. Lee *et al.* [9] observed that conflicting substreams may be strongly biased, just in opposite directions, and this led to the development of the bimode predictor. Most recently, Evers *et al.* [5] moved beyond the question of conflicts to focus on the correlation characteristics of branches, and found that many branches do benefit from global history. Yet for a given prediction, most global history bits go unused—adding to interference—while often the most useful branch outcomes have already been forced out of the history.

Hybrid prediction was originally proposed by McFarling [10]. Chang *et al.* [2] extended his work, finding—as we did—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. Most recently, Grunwald *et al.* [6] 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 [4] that contain more than two components, and in this case static selection might be beneficial.

Other than the bi-mode work [9], we are unaware of any published work describing alloying. Other researchers have described a variety of aggressive techniques for reducing conflict mispredictions. The YAGS predictor [3] is the most recent. It extends the bi-mode predictor by identifying when branches disagree with the predicted bi-mode bias. It is important to note that MAs can easily be extended to incorporate such anti-aliasing schemes.

7. 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 work has mainly focused on reducing conflict mispredictions due to aliasing in the pattern history table.

This paper has presented a new 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. We also showed that dynamic references by the same branch instruction can also suffer wrong-history mispredictions—that is, some *individual* branches dynamically alternate between needing global history and needing local history. In conventional hybrid predictors [2, 10], this favors dynamic selection over static selection [6].

An alloyed predictor merges local and global history bits together in a single PHT index. This is a generalization of the bi-mode branch predictor proposed by Lee, Chen, and Mudge [9]. 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, achieves substantially better prediction accuracies than other solo, two-level predictors, and also performs well against hybrid predictors, especially at smaller sizes where the components in a hybrid organization become too small. Conventional hybrid predictors perform poorly at small sizes, and conventional two-level predictors reach a domain of diminishing returns too early and therefore perform poorly at large sizes.

There are several promising avenues for further research. The most important additional work is on understanding branch-prediction behavior, extending our characterization of misprediction sources and individual-branch wrong-history effects. A second area for future work lies in further exploring alloying. For example, alloying might provide further benefits as part of a hybrid predictor, and in particular, might work well with static selection, where it could expose both types of history in one component and possibly obviate the need for dynamic selection. New hash functions might also be considered, and comparisons of alloyed prediction against newly proposed predictors like YAGS [3] would be informative. Finally, isolating exactly which branches benefit most from alloying and characterizing the reasons for this would also be informative.

Acknowledgments

We would like to thank Mikko Lipasti, Tom Conte, and the anonymous reviewers for their helpful comments. This work was supported in part by NSF grant CCR-94-23123, NSF Career Award CCR-95-02516 (Martonosi), and an NDSEG Graduate Fellowship (Skadron). We would also like to thank Adrian V. Lanning, whose senior thesis at the University of Virginia extended our understanding of how branches alternate among using different history types, and the SHRIMP research group at Princeton for the extensive use of their computing resources.

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, E. Hao, and Y. N. Patt. Alternative implementations of hybrid branch predictors. In *Proc. Micro-28*, pp. 252–57, Dec. 1995.

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

[4] 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*, pp. 3–11, May 1996.

[5] 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*, pp. 52–61, June 1998.

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

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

[8] A. V. Lanning. Pipelined branch prediction: Characterizing wrong-history misprediction. Senior thesis, Univ. of Virginia School of Engineering and Applied Science, Apr. 2000.

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

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

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

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

[13] 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*, 48(11):1260–81, Nov. 1999.

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

[15] K. Skadron, M. Martonosi, and D. W. Clark. Alloying global and local branch history: A robust solution to wrong-history mispredictions. Tech. Report TR-606-99, Princeton Dept. of Computer Science, Oct. 1999.

[16] K. Skadron, M. Martonosi, and D. W. Clark. Alloying global and local branch history: Taxonomy, performance, and analysis. Tech. Report TR-594-99, Princeton Dept. of Computer Science, Jan. 1999.

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

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

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

[20] R. Uhlig et al. Instruction fetching: Coping with code bloat. In *Proc. ISCA-22*, pp. 345–56, June 1995.

[21] S. C. Woo et al. The SPLASH-2 programs: Characterization and methodological considerations. In *Proc. ISCA-22*, pp. 24–36, June 1995.

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

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