To appear in Proceedings of the 24th Annual International Symposium on Computer Architecture

Trading Conflict and Capacity Aliasing in Conditional Branch Predictors

Pierre Michaud pmichaud@irisa.fr
André Seznec seznec@irisa.fr
Richard Uhlig * ruhlig@ichips.intel.com

IRISA/INRIA
Campus de Beaulieu
35042 Rennes, France
http://www.irisa.fr/caps/

Abstract

As modern microprocessors employ deeper pipelines and issue multiple instructions per cycle, they are becoming increasingly dependent on accurate branch prediction. Because hardware resources for branch-predictor tables are invariably limited, it is not possible to hold all relevant branch history for all active branches at the same time, especially for large workloads consisting of multiple processes and operating-system code. The problem that results, commonly referred to as aliasing *in the branch-predictor tables, is in many ways similar to the misses that occur in finite-sized hardware caches.*

In this paper we propose a new classification for branch aliasing based on the three-Cs model for caches, and show that conflict aliasing is a significant source of mispredictions. Unfortunately, the obvious method for removing conflicts – adding tags and associativity to the predictor tables – is not a cost-effective solution.

To address this problem, we propose the skewed branch predictor*, a multi-bank, tag-less branch predictor, designed specifically to reduce the impact of conflict aliasing. Through both analytical and simulation models, we show that the skewed branch predictor removes a substantial portion of conflict aliasing by introducing redundancy to the branch-predictor tables. Although this redundancy increases capacity aliasing compared to a standard one-bank structure of comparable size, our simulations show that the reduction in conflict aliasing overcomes this effect to yield a gain in prediction accuracy. Alternatively, we show that a skewed organization can achieve the same prediction accuracy as a standard one-bank organization but with half the storage requirements.*

Keywords

Branch prediction, aliasing, 3 C's classification, skewed branch predictor.

* Now with Intel Microcomputer Research Lab, Oregon

© 1997 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that new copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted.
To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request Permissions from Publications Dept, ACM Inc., Fax +1 (212) 869-0481, or permissions@acm.org.

1 Introduction and Related Work

In processors that speculatively fetch and issue multiple instructions per cycle to deep pipelines, dozens of instructions might be in flight before a branch is resolved. Under these conditions, a mispredicted branch can result in substantial amounts of wasted work and become a bottleneck to exploiting instruction-level parallelism. Accurate branch prediction has come to play an important role in removing this bottleneck.

Many dynamic branch prediction schemes have been investigated in the past few years, with each offering certain distinctive features. Most of them, however, share a common characteristic: they rely on a collection of 1- or 2-bit counters held in a *predictor table*. Each entry in the table records the recent outcomes of a given branch *substream* [21], and is used to predict the direction of future branches in that substream. A branch substream might be defined by some bits of the branch address, by a bit pattern representing previous branch directions (known as a *branch history*), by some combination of branch address and branch history, or by bits from target addresses of previous branches [14, 7, 18, 10, 8, 9].

Ideally, we would like to have a predictor table with infinite capacity so that every unique branch substream defined by an (address, history) pair will have a dedicated predictor. Chen et al. have shown that two-level predictors are close to being optimal, provided unlimited resources for implementing the predictors [3]. Real-world constraints, of course, do not permit this. Chip die-area budgets and access-time constraints limit predictor-table size, and most tables proposed in the literature are further constrained in that they are direct-mapped and without tags.

Fixed-sized predictor tables lead to a phenomenon known as *aliasing* or *interference* [21, 16], in which multiple (address, history) pairs share the same entry in the predictor table, causing the predictions for two or more branch substreams to intermingle. Aliasing has been classified as either *destructive* (i.e., a misprediction occurs due to sharing of a predictor-table entry), *harmless* (i.e., it has no effect on the prediction) or *constructive* (i.e., aliasing occasionally provides a good prediction, which would have been wrong otherwise) [21]. Young et al. have shown that constructive aliasing is much less likely than destructive aliasing [21].

Recent studies have shown that large or multi-process workloads with a strong OS component exhibit very high degrees of aliasing [11, 5], and require much larger predictor tables than previously thought necessary to achieve a level of accuracy close to an ideal, unaliased predictor table [11]. We therefore expect that new techniques for removing conflict aliasing could provide important gains towards increased branch-prediction accuracy.

Branch aliasing in fixed-size, direct-mapped predictor tables is in many ways analogous to instruction-cache or data-cache misses.

This suggests an alternative classification for branch aliasing based on the *three-Cs model* of cache performance first proposed by Hill [6]. As with cache misses, aliasing can be classified as *compulsory*, *capacity* or *conflict* aliasing. Similarly, as with caches, larger predictor tables reduce capacity aliasing, while associativity in a predictor table could remove conflict aliasing.

Unfortunately, a simple-minded adaptation of cache associativity would require the addition of costly tags, substantially increasing the cost of a predictor table. In this paper we examine an alternative approach, called skewed branch prediction, which borrows ideas from skewed-associative caches [12]. A *skewed branch predictor* is constructed from an odd number (typically 3 or 5) of *predictor-table banks*, each of which functions like a standard tag-less predictor table. When performing a prediction, each bank is accessed in parallel but with a different indexing function, and a majority vote between the resulting lookups is used to predict the direction of the branch.

In the next section we explain in greater detail our aliasing classification. In section 3, we quantify aliasing and assess the effect of conflict aliasing on overall branch-prediction accuracy. In section 4, we introduce the skewed branch predictor, a hardware structure designed specifically to reduce conflict aliasing. In section 5, we show how and why the skewed branch predictor removes conflict aliasing effects at the cost of some redundancy. Our analysis includes both simulation and analytical models of performance, and considers a range of possible skewed predictor configurations driven by traces from the instruction-benchmark suite (IBS) [17], which includes complete user and operating-system activity. Section 6 proposes the enhanced skewed branch predictor, a slight modification to the skewed branch predictor, which enables more attractive tradeoffs between capacity and conflict aliasing. Section 7 concludes this study and proposes some future research directions.

2 An Aliasing Classification

Throughout this paper, we will focus on global-history prediction schemes for the sake of conciseness. Global-history schemes use both the branch address and a pattern of global history bits, as described in [18, 19, 20, 10, 8]. Previously-proposed global-history predictors are all direct-mapped and tag-less. Given a history length, the distinguishing feature of these predictors is the hashing function that is used to map the set of all (address, history) pairs onto the predictor table.

The *gshare* and *gselect* schemes [8] have been the most studied global schemes (*gselect* corresponds to GAs in Yeh and Patt's terminology [18, 19, 20]). In *gshare*, the low-order address bits and global history bits are XORed together to form an index value[1], whereas in *gselect*, low-order address bits and global history bits are concatenated.

Aliasing occurs in direct-mapped tag-less predictors when two or more (address, history) pairs map to the same entry. To measure aliasing for a particular global scheme and table, we simulate a structure having the same number of entries and using the same indexing function as the predictor table considered. However, instead of storing 1-bit or 2-bit predictors in the structure, we store the identity of the last (address, history) pair that accessed the entry. Aliasing occurs when the indexing (address, history) pair is different from the stored pair. The *aliasing ratio* is the ratio between the number of aliasing occurrences and the number of dynamic conditional branches. When measured in this way, we can see the relationship between branch aliasing and cache misses. Our simulated tagged table is like a cache with a line size of one datum, and an aliasing occurrence corresponds to a cache miss.

[1] When the number of history bits is less than the number of index bits, the history bits are XORed with the *higher-order* end of the section of low-order address bits, as explained in [8]

benchmark	conditional branch count	
	dynamic	static
groff	11568181	5634
gs	14288742	10935
mpeg_play	8109029	4752
nroff	21368201	4480
real_gcc	13940672	16716
verilog	5692823	3918

Table 1: Conditional branch counts

A widely-accepted classification of cache misses is the three-Cs model, first introduced by Hill [6] and later refined by Sugumar and Abraham [15]. The three-Cs model divides cache misses into three groups, depending on their causes.

- **Compulsory misses** occur when an address is referenced for the first time. These unavoidable misses are required to fill an empty or "cold" cache.
- **Capacity misses** occur when the cache is not large enough to retain all the addresses that will be re-referenced in the future. Capacity misses can be reduced by increasing the total size of the cache.
- **Conflict misses** occur when two memory locations contend for the same cache line in a given window of time. Conflict misses can be reduced by increasing the associativity of a cache, or improving the replacement algorithm.

Aliasing in branch-predictor tables can be classified in a similar fashion:

- **Compulsory aliasing** occurs when a branch substream is encountered for the first time.
- **Capacity aliasing**, like capacity cache misses, is due to a program's working set being too large to fit in a predictor table, and can be reduced by increasing the size of the predictor table.
- **Conflict aliasing** occurs when two concurrently-active branch substreams map to the same predictor-table entry. Methods for reducing this component of aliasing have not yet, to our knowledge, appeared in the published literature.

3 Quantifying Aliasing

3.1 Experimental Setup

We conducted all of our trace-driven simulations using the IBS-Ultrix benchmarks [17]. These benchmarks were traced using a hardware monitor connected to a MIPS-based DECstation running Ultrix 3.1. The resulting traces include activity from all user-level processes as well as the operating-system kernel, and have been determined by other researchers to be a good test of branch-prediction performance [5, 11]. Conditional branch counts[2] derived from these traces are given in Table 1.

Although we simulated the *sdet* and *video_play* benchmarks, they exhibited no special behavior compared with the other benchmarks. We therefore omit *sdet* and *video_play* results from this paper in the interest of saving space.

[2] **beq r0,r0** is used as an unconditional relative jump by the MIPS compiler, therefore we did not consider it as conditional. This explains the discrepancy with the branch counts reported in [5, 11]

4-bit history				
benchmark	**substream ratio**	**compulsory aliasing**	**misprediction**	
			1-bit	**2-bit**
groff	1.82	0.09 %	5.47 %	3.77 %
gs	1.91	0.15 %	7.03 %	5.28 %
mpeg_play	1.83	0.11 %	9.08 %	7.24 %
nroff	1.79	0.04 %	4.99 %	3.72 %
real_gcc	2.36	0.28 %	9.38 %	7.16 %
verilog	1.96	0.13 %	6.48 %	4.57 %

12-bit history				
benchmark	**substream ratio**	**compulsory aliasing**	**misprediction**	
			1-bit	**2-bit**
groff	7.14	0.35 %	3.63 %	2.56 %
gs	7.95	0.61 %	3.71 %	2.77 %
mpeg_play	6.27	0.37 %	5.85 %	4.52 %
nroff	5.71	0.12 %	3.04 %	2.20 %
real_gcc	12.90	1.55 %	4.90 %	3.93 %
verilog	9.24	0.64 %	3.74 %	2.66 %

Table 2: Unaliased predictor

We first simulated an ideal unaliased scheme (i.e., a predictor table of infinite size). The misprediction ratios that we obtained are shown in Table 2 for history lengths of 4 and 12 bits, and for both 1-bit and 2-bit predictors (we include unconditional branches as part of the global-history bits). When an (address, history) pair is encountered for the first time, we do not count it as a misprediction, so compulsory miss contribution to mispredictions is not reported in the last two columns of Table 2.

The 2-bit saturating counter gives better prediction accuracy in an unaliased predictor table than the 1-bit predictor. Our intuition is that this difference is due mainly to loop branches. We also measured the *substream ratio*, which we define as the average number of different history values encountered for a given conditional branch address (see first column of Table 2).

The compulsory-aliasing percentage was computed from the number of different (address, history) pairs referenced throughout the trace divided by the total number of dynamic conditional branches. From Table 2, we observe that compulsory aliasing, with a 12-bit history length, generally constitutes less than 1% of the total of all dynamic conditional branches, except in the case of *real_gcc*, which exhibits a compulsory-aliasing rate of 1.55%.

3.2 Quantifying Conflict and Capacity Aliasing

To quantify conflict and capacity aliasing, we simulated tagged predictor tables holding (address, history) pairs. Figures 1 and 2 show the miss ratio in direct-mapped (DM) and fully-associative (FA) tables using 4 bits and 12 bits of global history, respectively. The two direct-mapped tables are indexed with a *gshare*- and a *gselect*-like function. The fully-associative table uses a least-recently-used (LRU) replacement policy.

The miss ratio for the fully-associative table gives the sum of compulsory and capacity aliasing. The difference between *gshare* or *gselect* and the fully-associative table gives the amount of conflict aliasing in the corresponding *gshare* and *gselect* predictors. It should be noted that LRU is not an optimal replacement policy [15]. However, because it bases its decisions solely on past information, the LRU policy gives a reasonable base value of the amount of conflict aliasing that can be removed by a hardware-only scheme.

It appears that for our benchmarks, *gselect* has a higher aliasing rate than *gshare*. This explains why, for a given table size and history length, *gshare* has a lower misprediction rate than *gselect*, as claimed in [8]. This difference is very pronounced with 12 bits of global history, because in this case, *gselect* uses only a very small number of address bits (e.g., only 4 address bits for a 64K-entry table).

Figure 1 shows that when the number of entries is larger than or equal to 4K, capacity aliasing nearly vanishes, **leaving conflicts as the overwhelming cause of aliasing**. The same condition holds in Figure 2 for table sizes greater than about 16K. This leads us to conclude that some amount of associativity in branch prediction tables is needed to limit the impact of aliasing.

3.3 Problems with Associative Predictor Tables

Associativity in caches introduces a degree of freedom for avoiding conflicts. In a direct-mapped cache, tag bits are used to determine whether a reference hits or misses. In an associative cache, the tag bits also determine the precise location of the requested data in the cache.

Because of its speculative nature, a direct-mapped branch prediction table can be tag-less. To implement associativity, however, we must introduce tags identifying (address, history) pairs. Unfortunately, the tag width is disproportionately large compared to the width of the individual predictors, which are usually 1 or 2 bits wide.

Another method for achieving the benefits of associativity, without having to pay the cost of tags is needed. The skewed branch predictor, described in the next section, is one such method.

4 The Skewed Branch Predictor

We have previously noted that the behaviors of *gselect* and *gshare* are different even though these two schemes are based on the same (address, history) information. This is illustrated on Figure 3 where we represent a *gshare* and a *gselect* table with 16 entries. In this example, there is a conflict both with *gshare* and *gselect*, but the (address, history) pairs that conflict are not the same. We can conclude that the precise occurrence of conflicts is strongly related to the mapping function. The skewed branch predictor is based on this observation.

The basic principle of the skewed branch predictor is to use several branch-predictor banks (3 banks in the example illustrated in Figure 4), but to index them by different and independent hashing functions computed from the same vector V of information (e.g., branch address and global history). A prediction is read from each of the banks and a majority vote is used to select a final branch direction.

The rationale for using different hashing functions for each bank is that two vectors, V and W, that are aliased with each other in one bank are unlikely to be aliased in the other banks. A destructive aliasing of V by W may occur in one bank, but the overall prediction on V is likely to be correct if V does not suffer from destructive aliasing in the other banks.

4.1 Execution Model

We consider two policies for updating the predictors across multiple banks:

- A **total update** policy: each of the three banks is updated as if it were a sole bank in a traditional prediction scheme.
- A **partial update** policy: when a bank gives a bad prediction, it is not updated when the overall prediction is good. This

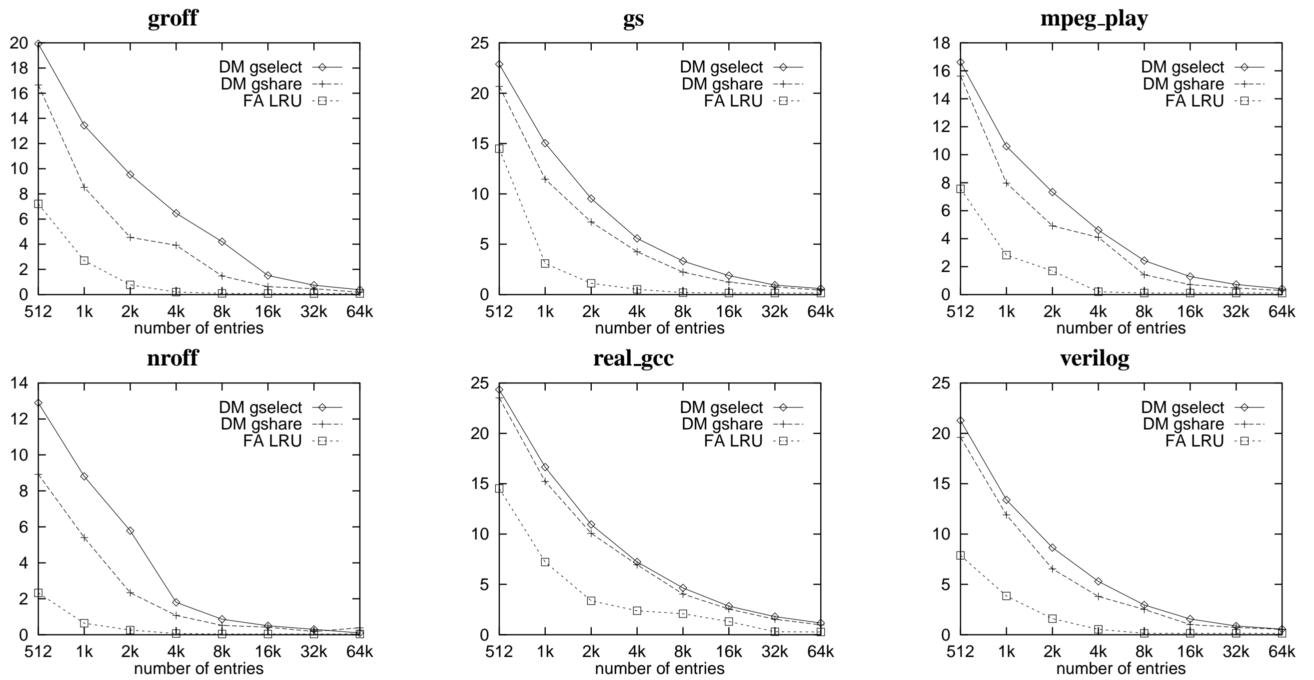

Figure 1: Miss percentages in tables tagged with (address, history) pairs (4-bit history)

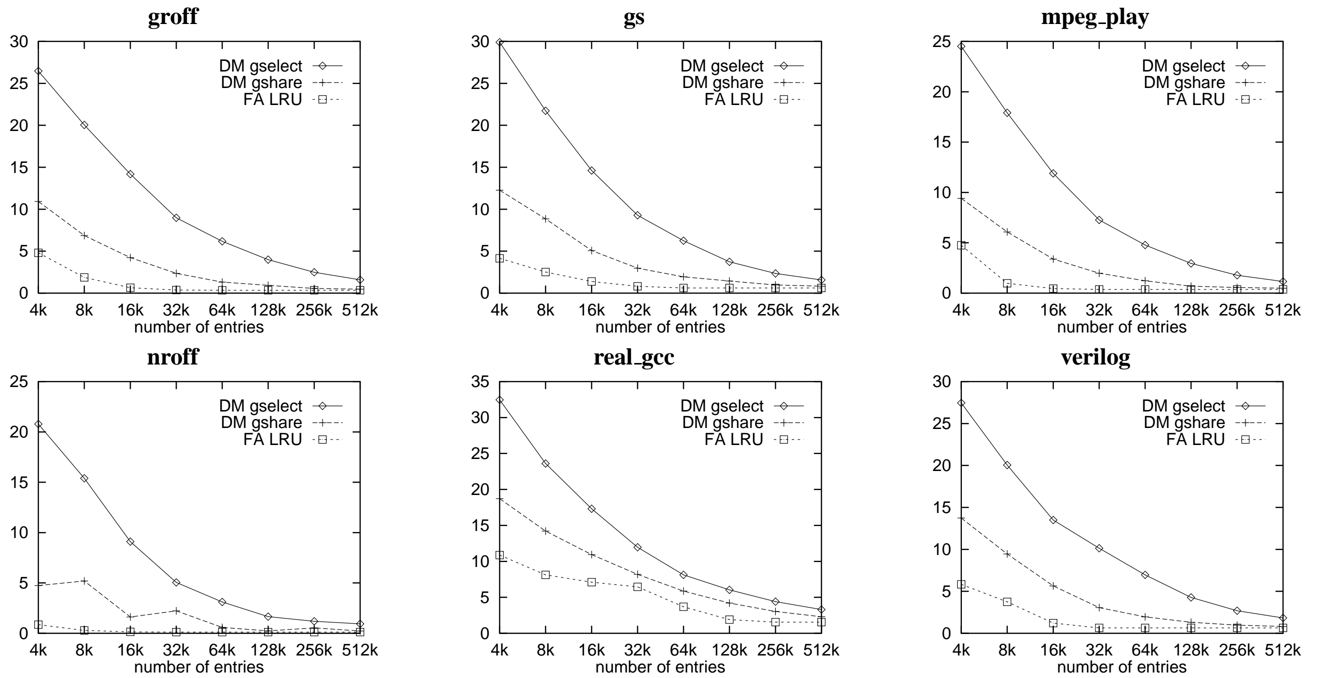

Figure 2: Miss percentages in tables tagged with (address, history) pairs (12-bit history)

wrong predictor is considered to be attached to another (address, history) pair. When the overall prediction is wrong, all banks are updated as dictated by the outcome of the branch.

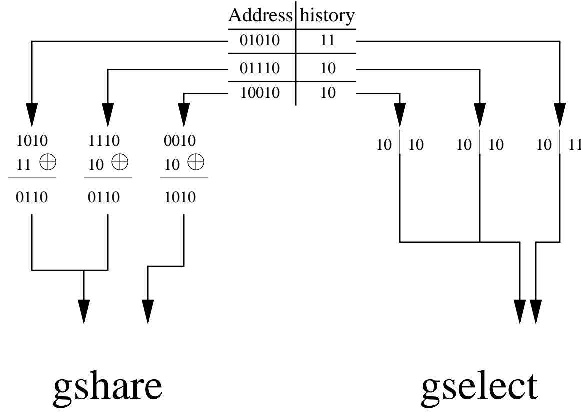

Figure 3: Conflicts depend on the mapping function

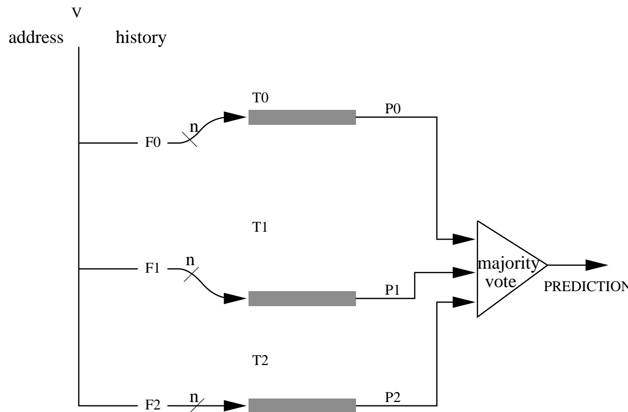

Figure 4: A Skewed Branch Predictor

4.2 Design Space

Chosing the information (branch address, history, etc.) that is used to divide branches into substreams is an open problem. The purpose of this section is not to discuss the relevance of using some combination of information or the other, but to show that most conflict aliasing effects can be removed by using a skewed predictor organization. For the remainder of this paper, the vector of information that will be used for recording branch-prediction information is the concatenation of the branch address and the k bits of global history: $V = (a_N, ...a_2, h_k, ...h_1)$. Let $\mathcal{V}$ be the set of all V's.

The functions f_0, f_1 and f_2 used for indexing the three 2^n-entry banks in the experiments are the same as those proposed for the skewed-associative cache in [13]. Consider the decomposition of the binary representation of vector V in bit substrings (V_3,V_2,V_1), such that V_1 and V_2 are two n-bit strings. Now consider the function H defined as follows:

$$\begin{array}{llll} H: & \{0,..,2^n-1\} & \longrightarrow & \{0,..,2^n-1\} \\ & (y_n, y_{n-1}, .., y_1) & \longmapsto & (y_n \oplus y_1, y_n, y_{n-1}, .., y_3, y_2) \end{array}$$

where $\oplus$ is the XOR (exclusive or) operation. We can now define three different mapping functions as follows:

$$\begin{array}{llll} f_0: & \mathcal{V} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_3, V_2, V_1) & \longmapsto & H(V_1) \oplus H^{-1}(V_2) \oplus V_2 \end{array}$$

$$\begin{array}{llll} f_1: & \mathcal{V} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_3, V_2, V_1) & \longmapsto & H(V_1) \oplus H^{-1}(V_2) \oplus V_1 \end{array}$$

$$\begin{array}{llll} f_2: & \mathcal{V} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_3, V_2, V_1) & \longmapsto & H^{-1}(V_1) \oplus H(V_2) \oplus V_2 \end{array}$$

Further information about these functions can be found in [13]. The most interesting property of these functions is that if two distinct vectors $(V3, V2, V1)$ and $(W3, W2, W1)$ map to the same entry in a bank, they will not conflict in the other banks if $(V2, V1) \neq (W2, W1)$. Any other function family exhibiting the same property might be used.

Having defined an implementation of the skewed branch predictor, we are now in a position to evaluate it and check its behavior against conventional global-history schemes.

For the purposes of comparison, we will use the *gshare* global scheme for referencing the standard single-bank organization. The skewed branch predictor described earlier will also be referred to as *gskewed* for the remainder of this paper.

5 Analysis

5.1 Simulation Results

The aim of this section is to evaluate the cost-effectiveness of the skewed branch predictor via simulation. In the skewed branch predictor, a prediction associated with a (branch, history) pair is recorded up to three times. It is intuitive that the impact of conflict aliasing is lower in a skewed branch predictor than in a direct-mapped *gshare* table. However, if the same total number of predictor storage bits is allocated to each scheme, it is not clear that *gskewed* will yield better results – the redundancy that makes *gskewed* work also has the effect of increasing the degree of capacity aliasing among a fixed set of predictor entries. Said differently, it may be better to simply build a one-bank predictor table 3 times as large, rather than a 3-bank skewed table.

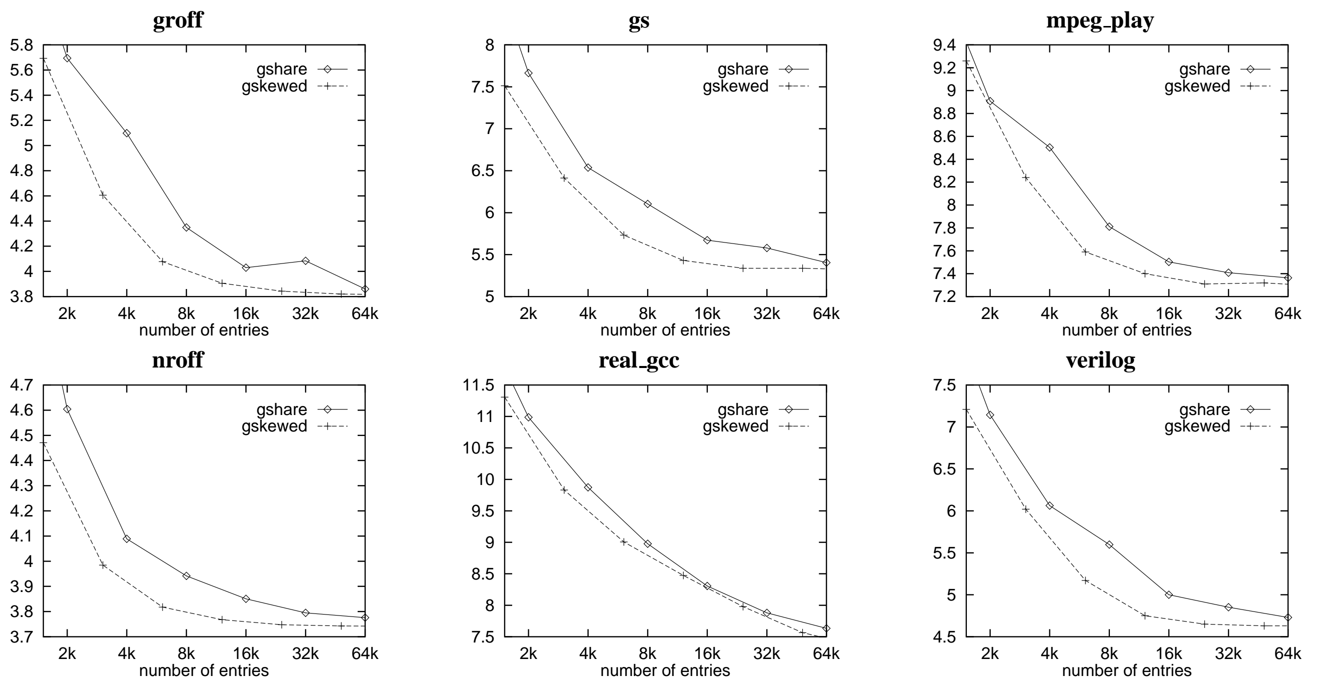

Figure 5: Misprediction percentage with 4-bit history

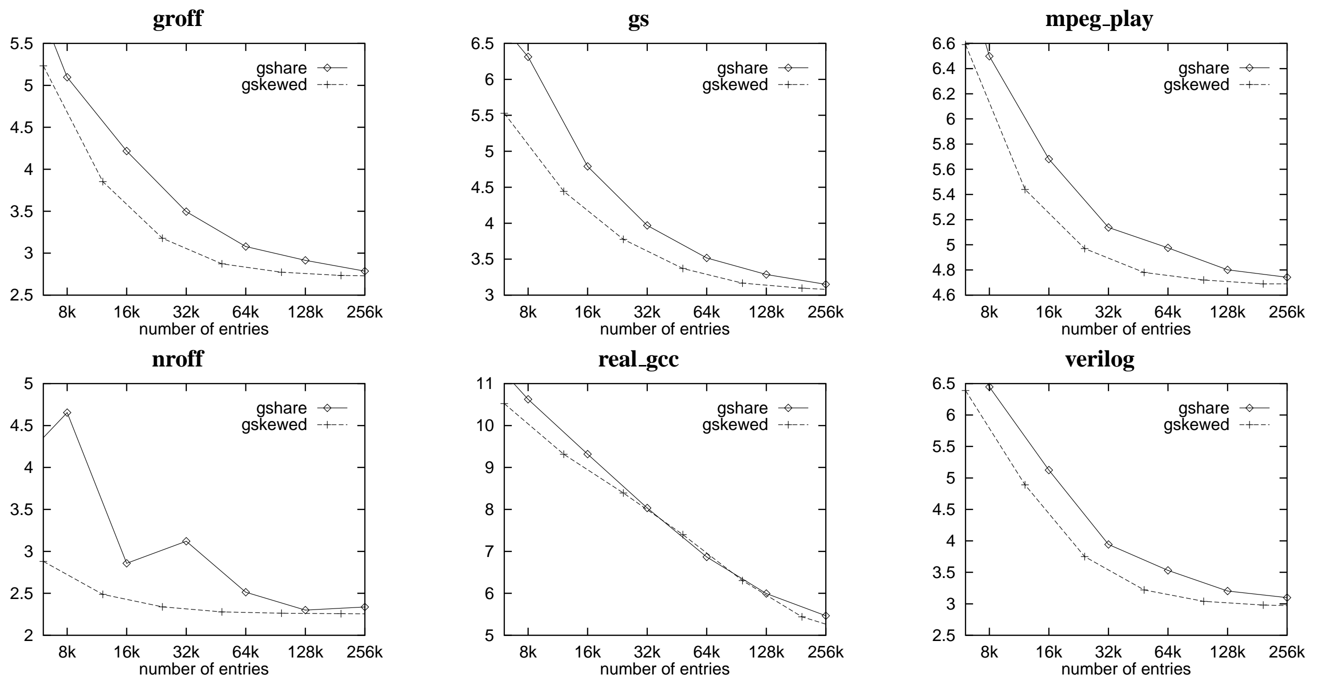

Figure 6: Misprediction percentage with 12-bit history

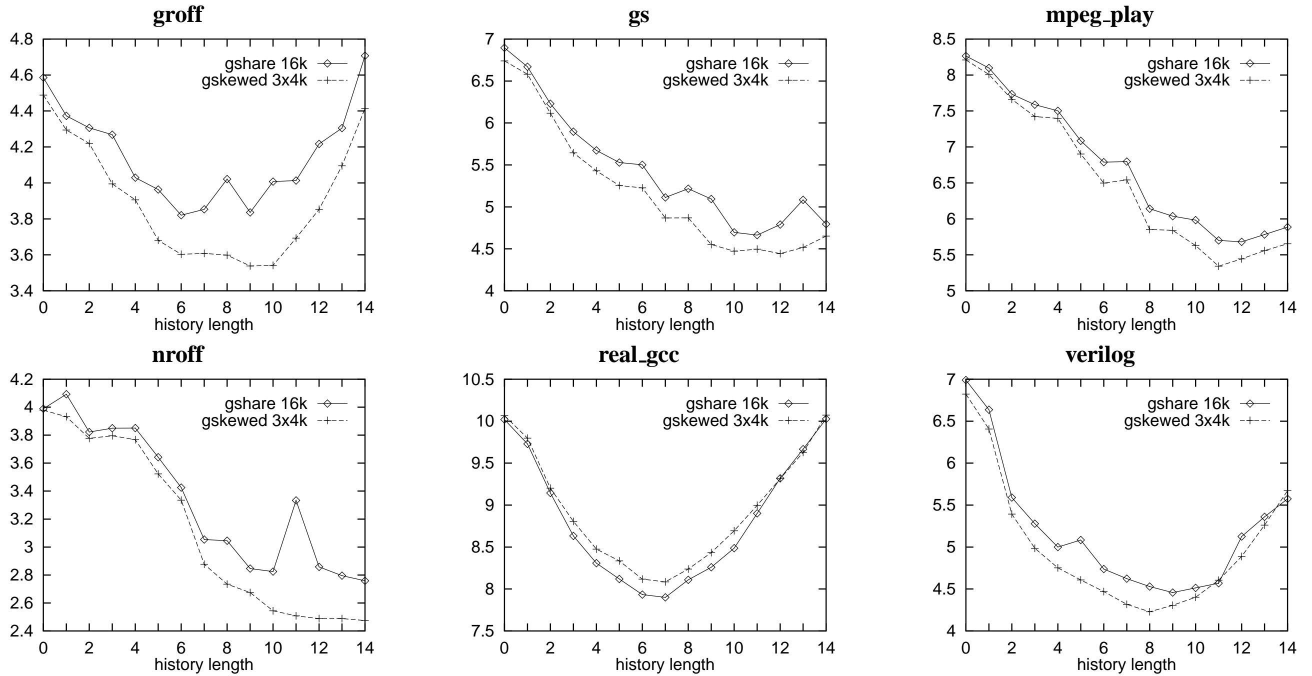

Figure 7: Misprediction percentage of 3x4k-gskewed vs. 16k-gshare

For the direct comparison between *gshare* and *gskewed*, we used 2-bit saturating counters and a *partial update* policy for *gskewed*.

Varying prediction table size The results for a history size of 4 bits and 12 bits are plotted in Figures 5 and 6, respectively, for a large spectrum of table sizes.

The interesting region of these graphs is where capacity aliasing for *gshare* has vanished. In this region,

> *a skewed branch predictor with a partial update policy achieves the same prediction accuracy as a 1-bank predictor, but requires approximately half the storage resources.*

For all benchmarks and for a wide spectrum of predictor sizes, the skewed branch predictor consistently gives better prediction accuracy than the 1-bank predictor. It should be noted that when using the skewed branch predictor and a history length of 4 (12), there is very little benefit in using more than 3x4k (3x16k) entries, while increasing the number of entries to 64k (256k) on *gshare* still improves the prediction accuracy.

Notice that the skewed branch predictor is more able to remove pathological cases. This appears clearly on Figure 6 for *nroff*.

Varying history length For any given prediction table size, some history length is better than others. Figure 7 illustrates the miss rates of a 3x4k-entry *gskewed* vs. a 16k-entry *gshare* when varying the history length. The plots show that despite using 25 % less storage resources, *gskewed* outperforms *gshare* on all benchmarks except *real_gcc*.

Varying number of predictor banks We also considered skewed configurations with five predictor banks. Our simulations results (not reported here) showed that there is very little benefit to increasing the number of banks to five; it appears that a 3-bank skewed branch predictor removes the most significant part of conflict aliasing, and a more cost-effective use of resources would be to increase the size of the banks rather than to increase their number.

Update policy To verify that *gskewed* is effective in removing conflict aliasing, we compare a 3$*$N-entry *gskewed* branch predictor with a fully-associative N-entry LRU table. Figure 8 illustrates this experiment for a global history length of 4 bits and 2-bit saturating counters. For (address, history) pairs missing in the fully-associative table, a static prediction *always taken* was assumed. For *gskewed*, both partial-update and total-update policies are shown.

It appears that a 3*N-entry *gskewed* table with partial update delivers slightly better behavior than the N-entry fully-associative table, but when it uses total-update policy, it exhibits slightly worse behavior. We conclude that a 3xN-entry *gskewed* predictor with partial update delivers approximately the same performance as an N-entry fully-associative LRU predictor.

The reason why *partial update* is better than *total update* is intuitive. For partial update, when 2 banks give a good prediction and the third bank gives a bad prediction, we do not update the third bank. By not updating the third bank, we enable it to contribute to the correct prediction of a different substream and effectively increase the capacity of the predictor table as a whole.

5.2 Analytical Model

Although our simulation results show that a skewed predictor table offers an attractive alternative to the standard one-bank predictor structure, they do not provide much explanation as to why a skewed organization works. In this section, we present an analytical model that helps to better understand why the technique is effective.

To make our analytical modeling tractable, we make some simplifying assumptions: we assume 1-bit automatons and the total update policy. We begin by defining the table *aliasing probability*. Consider a hashing function F which maps (address, history)

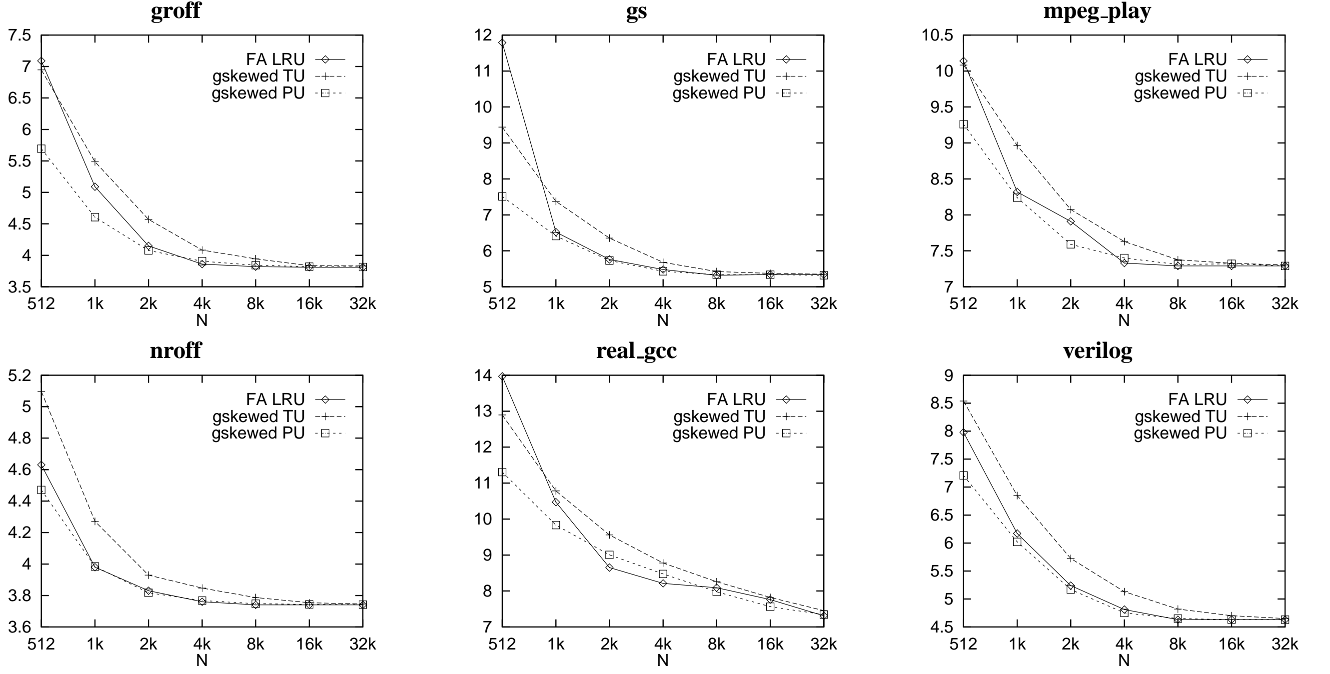

Figure 8: Misprediction percentage of 3N-entry gskewed vs. N-entry fully-associative LRU

pairs onto a N-entry table. The aliasing probability for a dynamic reference V = (address, history) is defined as follows:

Let D be the last-use distance of V, i.e. the number of distinct (address, history) pairs that have been encountered since the last occurrence of V. Assuming F distributes these D vectors equally in the table (i.e., assuming F is a good hashing function), the aliasing probability for dynamic reference V is

$$p_N = 1 - (1 - \frac{1}{N})^D \qquad (1)$$

When N is much greater than 1, we get a good approximation with

$$p_N = 1 - e^{-\frac{D}{N}} \qquad (2)$$

The aliasing probability is a function of the ratio between the last-use distance and the number of entries.

Let p represent the per-bank aliasing probability, and b be the probability that an (address, history) pair is biased *taken*. With 1-bit predictors, when an entry is aliased, the probability that the prediction given by that entry differs from the unaliased prediction is $2b(1-b)$. It should be noted that the aliasing is less likely to be destructive if b is close to 0 or 1 than if b is close to $1/2$.

Assuming a *total update policy*, and because we use different hashing functions for indexing the three banks, the events in a bank are not correlated with the events in an other bank. Now consider a particular dynamic reference V. Four cases can occur:

1. With probability $(1-p)^3$, V is not aliased in any of the three banks: the prediction will be the same as the unaliased prediction.

2. With probability $3p(1-p)^2$, V is aliased in one bank, but not in the other two banks: the resulting majority vote will be in the same direction as the unaliased prediction.

3. With probability $3p^2(1-p)$, V is aliased in two banks, but not in the remaining one.
With probability $b(1-b)^2 + (1-b)b^2$, predictions for both aliased banks are different from the unaliased prediction: the overall prediction is different from the unaliased prediction.

4. With probability p^3, V is aliased in all three banks.
With probability $b((1-b)^3+3b(1-b)^2)+(1-b)(b^3+3(1-b)b^2)$, the predictions are different from the unaliased prediction in at least two prediction banks: the skewed prediction is different from the unaliased prediction.

In summary, the probability that a prediction in our 3-bank skewed predictor differs from the unaliased prediction is :

$$\begin{aligned} P_{sk} = \quad & 3p^2(1-p)b(1-b) \\ & +p^3 b[3b(1-b)^2 + (1-b)^3] \\ & +p^3(1-b)[3(1-b)b^2 + b^3] \end{aligned} \qquad (3)$$

In contrast, the formula for a direct-mapped 1-bank predictor table is:

$$P_{dm} = [b(1-b) + (1-b)b]p = 2b(1-b)p \qquad (4)$$

P_{dm} and P_{sk} are plotted in Figure 9 for the worst case $b = 50$ %. We have

$$P_{sk} = \frac{3}{4}p^2(1-p) + \frac{1}{2}p^3$$

$$P_{dm} = \frac{1}{2}p$$

The main characteristic of the skewed branch predictor is that its mispredict probability is a polynomial function of the aliasing probability. The most relevant region of the curve is where the per-bank aliasing probability, p, is low (Figure 10 magnifies the curve for small aliasing probabilities).

At comparable storage resources, a 3-bank scheme has a greater per-bank aliasing probability than a 1-bank scheme, because each bank has a smaller number of entries. By taking into account formula (1), we find that for a 3x(N/3)-entry *gskewed*, P_{sk} is lower than P_{dm} in a N-entry direct-mapped table when the last-use distance D is less than approximately $\frac{N}{10}$, while for $D > \frac{N}{10}$, P_{sk} exceeds P_{dm}.

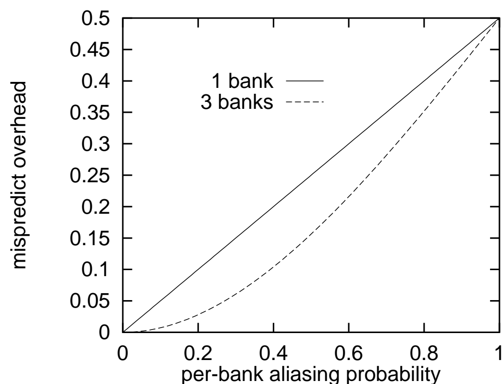

Figure 9: destructive aliasing

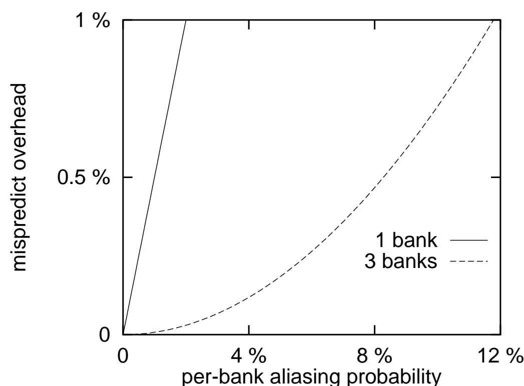

Figure 10: destructive aliasing

This highlights the tradeoff that takes place in the skewed branch predictor: a gain on short last-use distance references is traded for a loss on long last-use distance references. Now consider a N-entry fully-associative LRU table. When the last-use distance D is less than N, there is a hit, otherwise there is a miss. Hence, in a predictor table, aliasing for short last-use distance references is conflict aliasing, and aliasing for long last-use distance references is capacity aliasing.

In other words, **the skewed branch predictor trades conflict aliasing for capacity aliasing**.

To verify if our mathematical model is meaningful, we extrapolated the misprediction rate for *gskewed* by measuring D for each dynamic (address, history) pair and applied formulas (1) and (3). When an (address, history) pair was encountered for the first time, we applied formula (3) with $p = 1$. The bias probability b was evaluated for the entire trace by measuring the density of static (address, history) pairs with bias *taken*, and the value found was then fed back to the simulator when applying formula (3) on the same trace. Finally, we added the unaliased misprediction rate of table 2 (the contribution of compulsory aliasing to mispredictions appears only in the mispredict overhead).

The results are shown in Figure 11 for a history length of 4. It should be noted that our model always slightly overestimates the misprediction rate. This can be explained by the *constructive* aliasing phenomenon that is reported in [21].

As noted above, we made some simplifying assumptions when we devised our analytical model. The difficulty with extending the model to a partial-update policy is that occurrences of aliasing in a bank depend on what happens in the other banks. Modeling the effect of using 2-bit automatons is also difficult because a 2-bit automaton by itself removes some part of aliasing effects on prediction.

Despite the limitations of the model, it effectively explains why skewed branch prediction works: in a standard one-bank table, the mispredict overhead increases linearly with the aliasing probability, but in an M-bank skewed organization, it increases as an M-th degree polynomial. Because we deal with per-bank aliasing probabilities, which range from 0 to 1, a polynomial growth rate is preferable to a linear one.

6 An Enhanced Skewed Branch Predictor

Using a short history vector limits the number of (address, history) pairs (see the *substream ratio* column of Table 2) and therefore the amount of capacity aliasing. On the other hand, using a long history length leads to better intrinsic prediction accuracy on unaliased predictors, but results in a large number of (address, history) pairs. Ideally, given a fixed transistor budget, one would like to benefit from the better intrinsic prediction accuracy associated with a long history, and from the lower aliasing rate associated with a short history. Selecting a good history length is essentially a tradeoff between the accuracy of the unaliased predictor and the aliasing probability.

While the effect of conflict aliasing on the skewed branch predictor has been shown to be negligible, capacity aliasing remains a major issue. In this section we propose an enhancement to the skewed branch predictor that removes a portion of the capacity-aliasing effects without suffering from increased conflict aliasing.

In the *enhanced skewed branch predictor*, the complete information vector (i.e., branch history and address) is used with the hashing functions f_1 and f_2 for indexing bank 1 and bank 2, as in the previous *gskewed* scheme. But for function f_0, which indexes bank 0, we use the usual bit truncation of the branch address ($address \bmod 2^n$).

The rationale for this modification is as follows:
Consider an *enhanced gskewed* and *gskewed* using the same history length L, and (address, history) pair (A, H).

(A, H) has the same last-use distance D_L on the three banks of *gskewed* and on banks 1 and 2 of *enhanced gskewed*. But for *enhanced gskewed*, only the address is used for indexing bank 0, so the last-use distance D_S of the address A on bank 0 is shorter than D_L.

Two situations can occur:

1. When D_L is small compared with the bank size, the aliasing probability on a bank in either *gskewed* or *enhanced gskewed* is small, and both *gskewed* and *enhanced gskewed* are likely to deliver the same prediction as the unaliased predictor for history length L, because these predictions will be present in at least two banks.

2. When D_L becomes large compared with the bank size, the aliasing probability p_L on a any bank of *gskewed* or banks 1 and 2 of *enhanced gskewed* becomes close to 1 (formula (2) in the previous section).

   For both designs, when predictions on banks 1 and 2 differ, the overall prediction is equal to the prediction on bank 0. Now, since $D_S < D_L$, the aliasing probability p_S on bank 0 of *enhanced gskewed* is lower than the aliasing probability p_L on bank 0 of *gskewed*. When D_L is too high, the better intrinsic prediction accuracy associated with the long history on bank 0 in *gskewed* cannot compensate for the increased aliasing probability in bank 0.

Our intuition is that when the history length is short, the first situation will dominate and both predictors will deliver equivalent

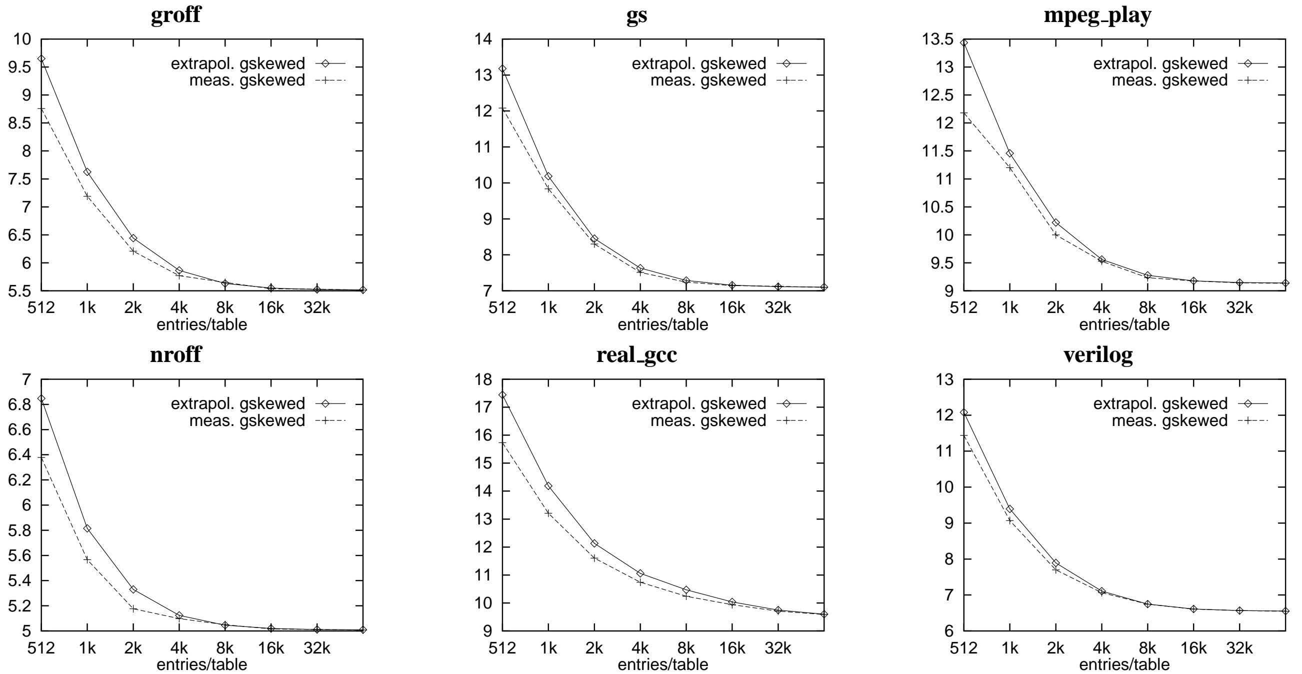

Figure 11: Extrapolated vs. measured misprediction percentage

prediction, but for a longer history length, the second situation will occur more often and *enhanced gskewed* will deliver better overall prediction than *gskewed*.

Simulation results: Figure 12 plots the results of simulations that vary the history length for a 3x4K-entry *enhanced gskewed*, a 3x4K-entry *gskewed* and a 32K-entry *gshare*. A partial-update policy was used in these experiments.

The curves for *gskewed* and *enhanced gskewed* are nearly indistinguishable up to a certain history length. After this point, which is different for each benchmark, the curves begin to diverge, with *enhanced gskewed* exhibiting lower mispredication rates at longer history lengths.

Based on our simulation results, 8 to 10 seems to be a reasonable choice for history length for a 3x4K-entry *gskewed* table, while for *enhanced gskewed*, 11 or 12 would be a better choice.

Notice that the 3x4K-entry *enhanced gskewed* performs as well as the 32K-entry *gshare* on all our benchmarks and for all history lengths, but with less than half of the storage requirements.

7 Conclusions and Future Work

Aliasing effects in branch-predictor tables have been recently identified as a significant contributor to branch-misprediction rates. To better understand and minimize this source of prediction error, we have proposed a new branch-aliasing classification, inspired by the three-Cs model of cache performance.

Although previous branch-prediction research has shown how to reduce compulsory and capacity aliasing, little has been done to reduce conflict aliasing. To that end, we have proposed *skewed branch prediction*, a technique that distributes branch predictors across multiple banks using distinct and independent hashing functions; multiple predictors are read in parallel and a majority vote is used to arrive at an overall prediction.

Our analytical model explains why skewed branch prediction works: in a standard one-bank table, the misprediction overhead increases linearly with the aliasing probability, but in an M-bank skewed organization, it increases as an M-th degree polynomial. Because we deal with per-bank aliasing probabilities, which range from 0 to 1, a polynomial growth rate is preferable to a linear one.

The redundancy in a skewed organization increases the amount of capacity aliasing, but our simulation results show that this negative effect is more than compensated for by the reduction in conflict aliasing when using a partial-update policy.

For tables of 2-bit predictors and equal lengths of global history, a 3-bank skewed organization consistently outperforms a standard 1-bank organization for all configurations with comparable total storage requirements. We found the update policy to be an important factor, with partial update consistently outperforming total update. Although 5-bank (or greater) configurations are possible, our simulations showed that the improvement over a 3-bank configuration is negligible. We also found skewed branch prediction to be less sensitive to pathological cases (e.g., *nroff* in Figure 6).

To reduce capacity aliasing further, we proposed the enhanced skewed branch predictor, which was shown to consistently reach the performance level of a conventional *gshare* predictor of more than twice the same size.

In addition to these performance advantages, skewed organizations offer a chip designer an additional degree of flexibility when allocating die area. Die-area constraints, for example, may not permit increasing a 1-bank predictor table from 16K to 32K, but a skewed organization offers a middle point: 3 banks of 8K entries apiece for a total of 24K entries.

In this paper, we have only addressed aliasing on prediction schemes using a global history vector. The same technique could be applied to remove aliasing in other prediction methods, including per-address history schemes [18, 19, 20], or hybrid schemes [8, 2, 1, 4].

Skewed branch prediction raises some new questions:

- **Update Policies:** Are there policies other than partial-update and total-update that offer better performance in skewed or enhanced skewed branch predictors?

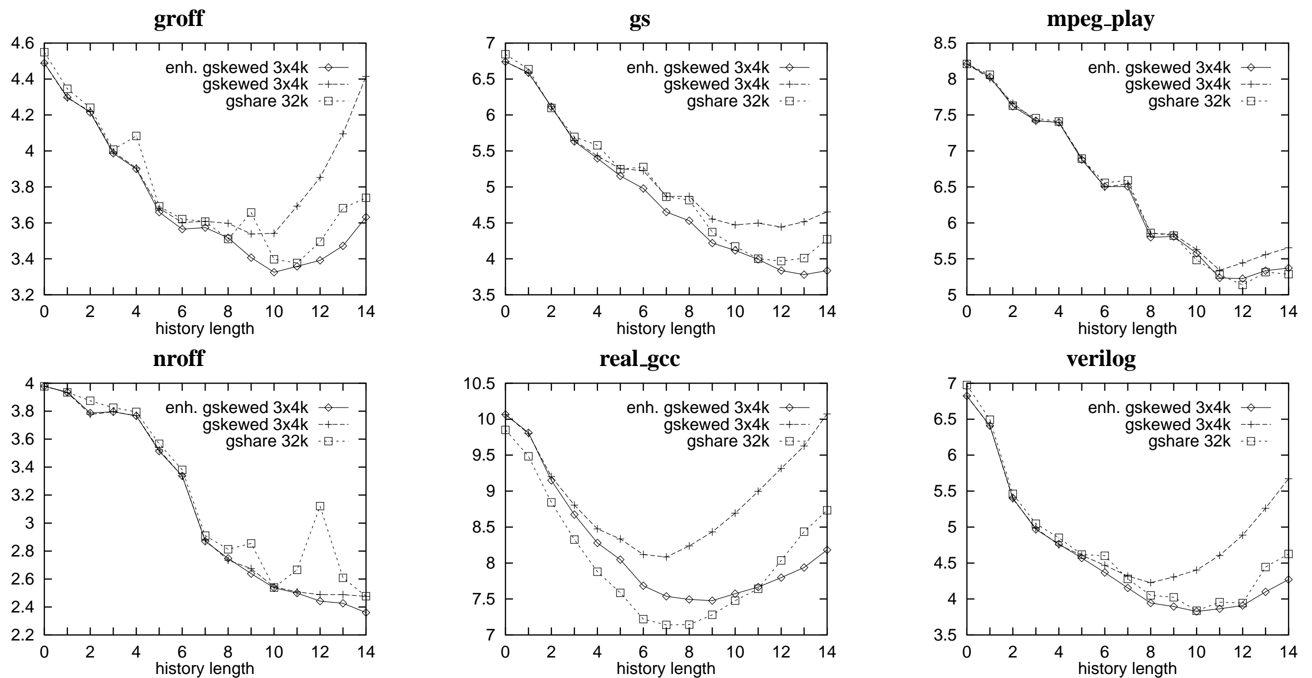

Figure 12: Misprediction percentage of *enhanced gskewed*

- **Distributed Predictor Encodings:** In our simulations we adopted the standard 2-bit predictor encodings and simply replicated them across 3 banks. Do there exist alternative "distributed" predictor encodings that are more space efficient, and more robust against aliasing?
- **Minimizing Capacity Aliasing**: Skewed branch predictors are very effective in reducing conflict-aliasing effects, but they do so at the expense of increased capacity aliasing. Do there exist other techniques, like those used in the enhanced skewed predictor, that could minimize these effects?

References

[1] P.-Y. Chang, E. Hao, and Y.N. Patt. Alternative implementations of hybrid branch predictors. In *Proceedings of the 28th International Symposium on Microarchitecture*, 1995.

[2] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y.N. Patt. Branch classification: a new mechanism for improving branch predictor performance. In *Proceedings of the 27th International Symposium on Microarchitecture*, 1994.

[3] I.-C.K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[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 *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[5] N. Gloy, C. Young, B. Chen, and M.D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[6] M.D. Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, University of California, Berkeley, 1987.

[7] J.K.F. Lee and A.J. Smith. Branch prediction strategies and branch target buffer design. *IEEE Computer*, pages 6–22, January 1984.

[8] Scott McFarling. Combining branch predictors. Technical report, DEC, 1993.

[9] R. Nair. Dynamic path-based branch correlation. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, 1995.

[10] S.T. Pan, K. So, and J.T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[11] 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*, May 1996.

[12] A. Seznec. A case for two-way skewed-associative caches. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993.

[13] A. Seznec and F. Bodin. Skewed associative caches. In *Proceedings of PARLE' 93*, May 1993.

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

[15] R.A. Sugumar and S.G. Abraham. Efficient simulation of caches under optimal replacement with applications to miss

characterization. In *Proceedings of the ACM SIGMETRIC Conference*, 1993.

[16] 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 3rd Annual International Conference on Parallel Architectures and Compilation Techniques*, 1995.

[17] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.

[18] T.-Y. Yeh and Y.N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th International Symposium on Microarchitecture*, Nov. 1991.

[19] T.-Y. Yeh and Y.N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

[20] 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*, May 1993.

[21] 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*, June 1995.