Accurate Indirect Branch Prediction

Karel Driesen and Urs Hölzle
Department of Computer Science
University of California
Santa Barbara, CA 93106

Technical Report TRCS97-19
Revised March 15, 1998

Abstract. Indirect branch prediction is likely to become increasingly important in the future because indirect branches occur more frequently in object-oriented programs. With misprediction rates of around 25% on current processors, indirect branches can incur a significant fraction of branch misprediction overhead even though they remain less frequent than the more predictable conditional branches. We investigate a wide range of two-level predictors dedicated exclusively to indirect branches. Starting with predictors that use full-precision addresses and unlimited tables, we progressively introduce hardware constraints and minimize the loss of predictor performance at each step. For programs from the SPECint95 suite as well as a suite of large C++ applications, a two-level predictor achieves a misprediction rate of 9.8% with a 1K-entry table and 7.3% with an 8K-entry table, representing more than a threefold improvement over an ideal BTB. A hybrid predictor further reduces the misprediction rates to 8.98% and 5.95%, respectively.

1. Introduction

Current high-performance superscalar processors use branch prediction to speculatively execute instructions beyond an unresolved branch. If the branch is mispredicted, this work is lost, and execution must restart right after the branch instruction. Newer designs increase instructions issue width and pipeline depth, increasing the relative overhead of mispredicted branches and making accurate branch prediction even more critical to performance.

Conditional direct branches, whose target is encoded in the instruction itself, can already be predicted with reported hit rates of up to 97% ([YP93]). In contrast, indirect branches, which transfer control to an address stored in a register, are harder to predict accurately. Unlike conditional branches, they can have more than two targets so that prediction requires a full 32-bit or 64-bit address rather than just a "taken" or "not taken" bit. Current processors predict indirect branches with a branch target buffer (BTB) which caches the most recent target address of a branch. Unfortunately, BTBs typically have much lower prediction rates than the best predictors for conditional branches. For example, an ideal (unconstrained) BTB achieves an average prediction hit ratio of only 64% on the SPECint95 benchmarks.

Though not as common as conditional branches, indirect branches occur frequently enough to cause substantial overhead. Chang et al. [CHP97] predict a reduction in execution time of 14% and 5% for the *perl* and *gcc* benchmarks on a wide-issue superscalar processor with an improved prediction mechanism for indirect branches (Target Cache).

In C++ and Java programs, indirect branches occur with even higher frequency (see Table 1). These languages promote a polymorphic programming style in which late binding of subroutine invocations is the main instrument for modular code design. Virtual function tables, the implementation of choice for most C++ and Java compilers, execute an indirect branch for every polymorphic call. The C++ programs studied here execute an indirect branch as frequently as once every 50 instructions; other studies [CGZ94] have shown similar results. Some of the C++ programs in Table 1 execute only 6 conditional branches for every indirect branch.

Predictated instructions [M+94] further increase the importance of indirect branch prediction since they remove conditional branches and thus conditional branch misses. For example, Intel expects predication to reduce the number of conditional branches by half for the IA-64 architecture ([Intel97]). With indirect branches becoming more frequent relative to conditional branches, and with indirect branches being mispredicted much more frequently, indirect branch prediction misses can start to dominate the overall branch misprediction cost. For example, if indirect branches are mispredicted 12 times more frequently (36% vs. 3% miss ratio), indirect branch misses will dominate conditional branch misses as long as indirect branches occur more frequently than every 12 conditional branches.

As the relevance of indirect branches grows, so does the opportunity for more sophisticated prediction mechanisms. In the next decade, uniprocessors may reach one billion transistors, with 48 million transistors dedicated to branch prediction ([P+97]).

In this study, we explore the design space of prediction mechanisms that are exclusively dedicated to indirect branches. Since the link between misprediction rate and execution overhead has been demonstrated in [CHP97], we focus on the minimization of branch misprediction rate. Initially, we assume unlimited hardware resources so that results are not obscured by implementation artifacts such as interference in tagless tables. We then progressively introduce hardware constraints, each of which causes a new type of interference and corresponding performance loss. We repeat this process until we obtain implementable predictors. Finally, the practical predictors are pairwise combined into a hybrid predictor, further improving prediction accuracy.

2. Benchmarks

Our main benchmark suite consists of large object-oriented C++ applications that range from 8,000 to over 75,000 non-blank lines of C++ code each (see Table 1), and *beta*, a compiler for the Beta programming language ([MMN93]), written in Beta. We also measured the SPECint95 benchmark suite (see Table 2) with the exception of *compress* which executes only 590 branches during a complete run. Together, the benchmarks represent over 500,000 non-comment source lines.

All C and C++ programs except *self*[1] were compiled with GNU gcc 2.7.2 (options -O2 -multrasparc plus static linking) and run under the *shade* instruction-level simulator [CK93] to obtain traces of all indirect branches. Procedure returns were excluded because they can be predicted accurately with a return address stack [KE91]. All programs were run to completion or until six million indirect branches were executed.[2] In *jhm* and *self* we excluded the initialization phases by skipping the first 5 and 6 million indirect branches, respectively.

[1] *self* does not execute correctly when compiled with -O2 and was thus compiled with “-O” optimization. Also, *self* was not fully statically linked; our experiments exclude instructions executed in dynamically-linked libraries.

Name	Description	lines of code	# of indirect branches	instr. / indirect	cond. / indirect	virt. func.	active branch sites			
							90%	95%	99%	100%
idl	SunSoft's IDL compiler (version 1.3)	13,900	1,883,641	47	6	93%	6	15	70	543
jhm	Java High-level Class Modifier: 6-12M	15,000	6,000,000	47	5	94%	11	16	34	155
self	Self-93 VM: 5-6M	76,900	1,000,000	56	7	76%	309	462	848	1855
troff	GNU groff version 1.09	19,200	1,1105,92	90	13	74%	19	32	61	161
lcom	compiler for hardware description language	14,100	1,7377,51	97	10	60%	8	17	87	328
porky	SUIF 1.0 scalar optimizer	22,900	5,392,890	138	19	71%	35	51	89	285
ixx	IDL parser, part of the Fresco X11R6 library	11,600	212,035	139	18	47%	31	46	91	203
eqn	typesetting program for equations	8,300	296,425	159	25	34%	17	23	58	114
beta	BETA compiler	72,500	1,005,995	188	23	N/A	37	54	135	376

Table 1. OO Benchmarks

Name	Description	lines of code	# of indirect branches	instr. / indirect	cond./ indirect	active branch sites			
						90%	95%	99%	100%
xlisp	SPEC95	4,700	6,000,000	69	11	3	3	4	13
perl	SPEC95	21,400	300,000	113	17	6	6	7	24
edg	EDG C++ front end	114,300	548,893	149	23	91	125	186	350
gcc	SPEC95	130,800	864,838	176	31	38	56	95	166
m88ksim	SPEC95	12,200	300,000	1827	233	3	4	5	17
vortex	SPEC95	45,200	3,000,000	3480	525	5	6	10	37
ijpeg	SPEC95	16,800	32,975	5770	441	3	5	7	60
go	SPEC95	29,200	549,656	56355	7123	2	2	5	14

Table 2. C Benchmarks

For each benchmark, the tables list the number of indirect branches executed, the number of instructions executed per indirect branch, the number of conditional branches executed per indirect branch, and the percentage of indirect branches in C++ programs that correspond to virtual function calls. For example, only 34% of the indirect branches in *eqn* are due to virtual function calls; the rest represent indirect calls through function pointers, indirect branches of switch statements, etc. In addition, the tables list the number of indirect branch sites responsible for 90%, 95%, 99%, and 100% of the indirect branches. For example, only 2 different branch sites are responsible for 95% of the dynamic indirect branches in *go*.

99% of the indirect branches in the OO and SPEC programs are executed from less than 200 indirect branch sites, except for *self* which contains a much larger number of active indirect branches (848). The SPECint95 programs are even more dominated by very few indirect branches, with less than ten interesting branches for all programs except *gcc*. Because there are so few distinct indirect branches in these programs, they are much more sensitive to variations in indirect branch prediction

[2] We reduced the traces of three of the SPEC benchmarks in order to reduce simulation time. In all of these cases, the BTB misprediction rate differs by less than 1% (relative) between the full and truncated traces, and thus we believe that the results obtained with the truncated traces are accurate.

schemes since a change in the prediction accuracy of a single indirect branch may significantly affect the overall prediction rate.

The relevance of indirect branch prediction is indicated by the number of instructions per indirect branch, and by the number of conditional branches per indirect branch. Three groups emerge: five of the OO benchmarks and one C benchmark execute fewer than 100 instructions per indirect branch; four OO benchmarks and three C benchmarks execute between 100 and 200 instructions for each indirect branch; and four of the SPEC benchmarks execute more than 1,000 instructions per indirect branch. Since the impact of branch prediction will be very low for the latter four benchmarks, we exclude them from all averages. Table 3 shows the groups for which we will commonly show average misprediction rates.

Name	Number of programs in group	Criteria
AVG-OO	9	OO benchmarks (Table 1)
AVG-C	4	C-benchmarks (excluding AVG-infreq, see Table 2)
AVG	13	AVG-100 plus AVG-200
AVG-100	6	all benchmarks with fewer than 100 instructions per indirect branch
AVG-200	7	all benchmarks with between 100 and 200 instructions per indirect branch
AVG-infreq	4	all benchmarks with more than 1,000 instructions per indirect branch

Table 3. Benchmark groups

We have included the SPECint95 programs mostly for comparison purposes; we do not believe that they are the best choice for evaluating indirect branch predictors (except for *gcc*). In effect, most SPEC benchmarks are microbenchmarks as far as indirect branch prediction is concerned, since very few branches dominate their behavior, and they are executed sparingly. In our evaluation of indirect branch prediction schemes we will therefore focus on the behavior of the larger OO programs (by minimizing AVG misprediction rates).

3. Unconstrained Indirect Branch Predictors

We first study the intrinsic predictability of indirect branches by ignoring any hardware constraints on predictor size or organization. Thus, we assume unconstrained, fully associative tables and full 32-bit addresses (unless indicated otherwise).

3.1 Branch Target Buffers

Current processors use a branch target buffer (BTB) to predict indirect branches. The predictor uses the branch address as a key into a table (the BTB) which stores the last target address of the branch (Figure 1).

We simulated two variants: “BTB” is a standard BTB which updates its target address after each branch execution. “BTB-2bc” is a BTB with two-bit counters which updates its target only after two consecutive mispredictions[1]. BTB-2bc predictors perform better in virtually all cases, with an

[1] In conditional branch predictors, the latter strategy is implemented with a two-bit saturating counter (2bc), hence the name. For an indirect branch, one bit suffices to indicate whether the entry had a miss the last time it was consulted.

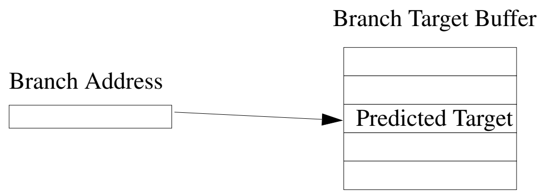


Figure 1. Branch target Buffer

average of 24.9% misprediction rate, compared to 28.1% for a standard BTB. Polymorphic branches occasionally switch their target but are often dominated by one most frequent target, a situation observed in object-oriented programs [AH96], [D+96]. But even with two-bit counters BTB accuracy is quite poor, ranging from average misprediction ratios of 20% in OO programs to 37% for C programs. Infrequent indirect branches (AVG-200) are less predictable, with a misprediction average of 38% vs. 10% for the programs in AVG-100.

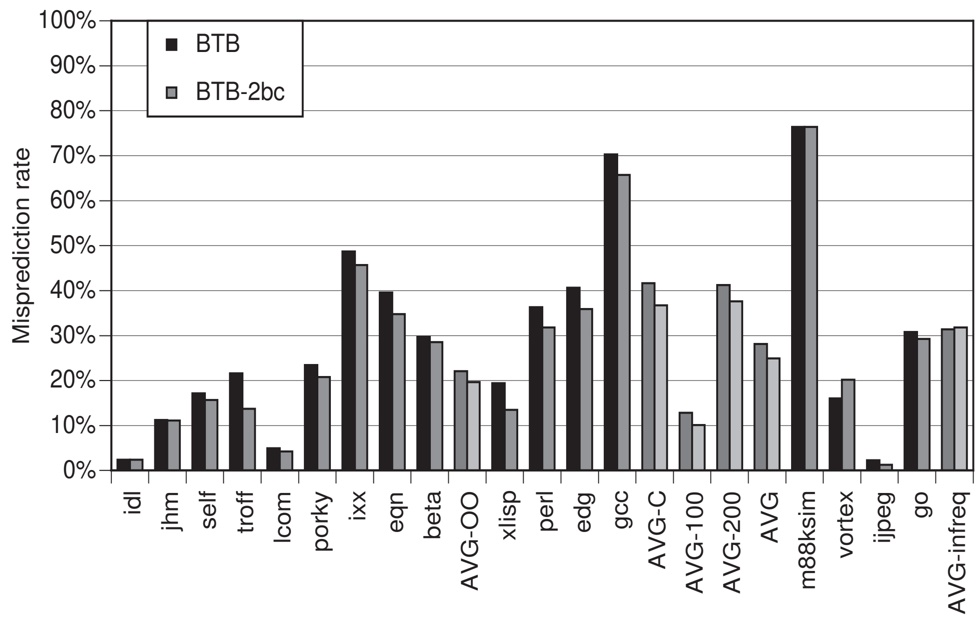


Figure 2. Indirect branch misprediction rates for an unconstrained BTB

3.2 Two-Level Prediction for Indirect Branch Paths

Two-level predictors improve prediction accuracy by keeping information from previous branch executions in a history buffer. Combined with the branch address, this history pattern is used as a key into the History Table which contains the predicted target addresses. As in BTBs, the entries can be updated on every miss or after two consecutive misses (2-bit counters). We tested every predictor in this section with both variants, and always saw a slight improvement with 2-bit counters. I.e., ignoring a stand-alone miss when updating seems to be a good strategy in general. Thus, we will only show 2-bit counter results in the rest of the paper.

For conditional branches, a branch history of length *p* consists of the taken/not-taken bits of the *p* most recently executed branches [YP93]. In contrast, most indirect branches are unconditional, and thus keeping a history of taken/not-taken bits would be ineffective. Instead, the history must consist of previous target addresses or bits thereof. Such a path-based history could also be used to predict

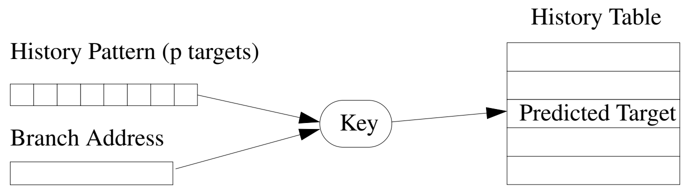


Figure 3. Two-level branch prediction

conditional branches, but since taken/not-taken bits succinctly summarize the target addresses of a conditional branch, conditional branch predictors usually do not employ target address histories (but see [Nair95]).

3.2.1 First Level: History Pattern

Branch predictors can use one or more history buffers. A *global history* uses a single history buffer (correlation branch prediction), and all branches are predicted using the outcomes of the *p* most recently executed branches. In contrast, a *per-address history* keeps a separate history for each branch, so that branches do not influence each other's prediction. Finally, *per-set history* prediction forms a compromise by using a separate history for each set of branches, where a set may be determined by the branch opcode, a compiler-assigned branch class, or a particular address range ([YP93]).

To investigate the impact of global vs. local histories, we simulated per-set histories where a set contains all branches in a memory region of size 2^s bytes, i.e., all branches with the same values in bits *s*..31 fall into the same set (Figure 4). With this parametrization, a global history corresponds

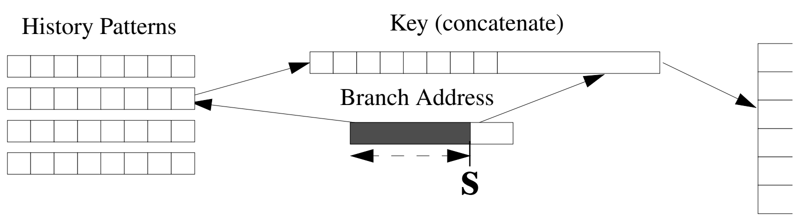


Figure 4. History pattern sharing

to *s*=31, and per-branch histories correspond to *s*=2. Using the results of the exhaustive run on a limited benchmark suite, we obtained good initial values for the other parameters (path length p=8, per-branch pattern entries in the history table). The results are shown in Figure 5.

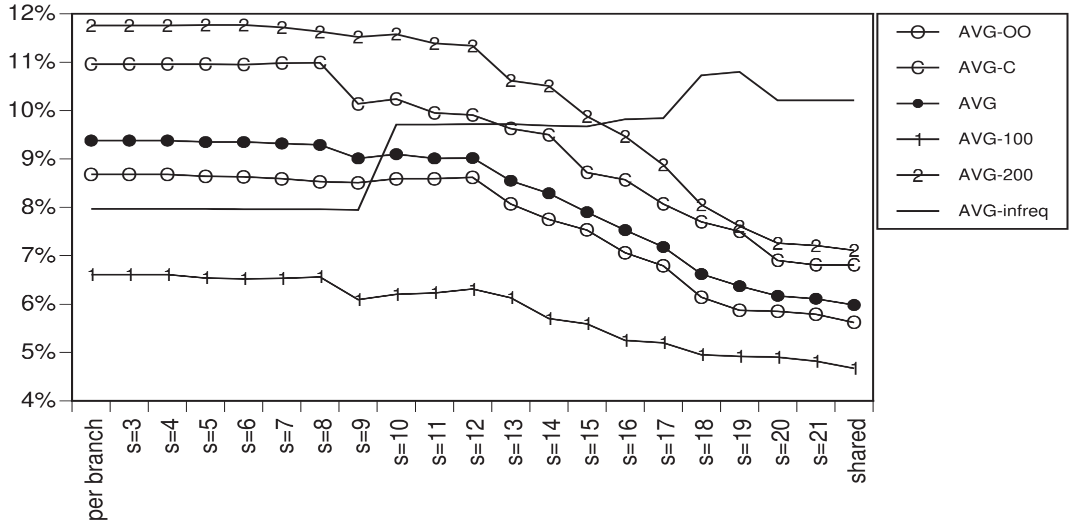


Figure 5. Influence of history sharing for path length p=8, per-branch entries

In general, a global history outperforms local histories for all benchmark groups except *AVG-infreq*. *AVG* declines from 9.4% with per-address paths down to 6.0% with a global path. The OO programs benefit most from sharing paths, with misprediction rates falling from 8.7% to 5.6%. This result indicates a substantial correlation between different branches (i.e., inter-branch correlation) in our benchmark suite, a correlation not limited by code distance. This result is analogous to the results for conditional branches in [YP93], where a global predictor generally performs better than a per-address scheme.

The C benchmarks show a pronounced dip for s=9 (i.e., if branches within 512-byte code regions share paths). On closer observation, the dip is caused by *xlisp* where only three indirect branches are responsible for 95% of the dynamic indirect branch executions. For *xlisp*, moving from s=8 to s=9 reduces mispredictions by a factor of three. Similarly, at s=10 *go*'s misprediction ratio jumps from 26% to 33% (*go* is dominated by two indirect branches), which causes *AVG-infreq* to jump at s=10.

The programs in *AVG-infreq* (which execute indirect branches very infrequently) are the only ones benefiting from per-address histories (*AVG-infreq*). Apparently, the targets of different branches do not correlate well with each other since they occur very far apart. Since these programs use indirect branches only sparingly, we can safely ignore their different behavior when designing branch predictors.

3.2.2 Second Level: History Table Sharing

A two-level predictor uses the history pattern to index into a history table that stores predicted targets. Again, we have a choice of per-branch, per-set, or global prediction. We simulated per-set tables that grouped all branches with identical address bits *h*..31 into the same set (see Figure 6). Thus, *h*=2 implies per-branch history tables (each branch has its own history table) and *h*=31 implies a single shared history table (i.e., all branches with the same history share the same predic-

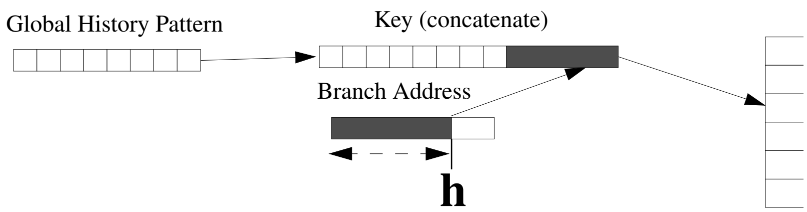


Figure 6. History Table sharing

tion). Figure 7 shows that the branch address matters: The misprediction average for all bench-

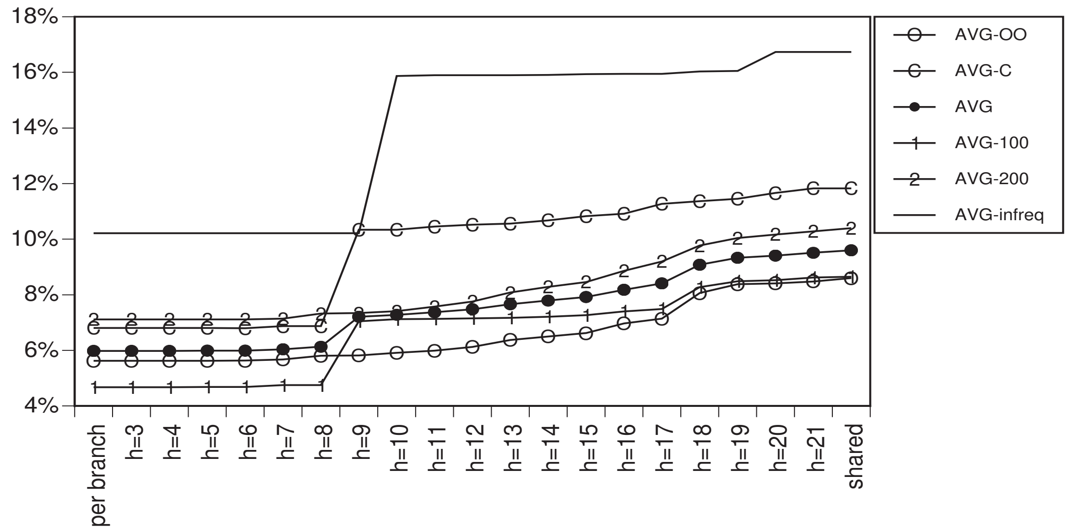


Figure 7. Influence of history table sharing for path length 8 with a global history pattern.

marks increases from 6.0% for per-address history tables to 9.6% for a globally shared history table, the rate of the OO programs increases from 5.6% to 8.6%, and that of the C benchmarks rises from 6.8% to 11.8%. (Again, *xlisp* changes dramatically at *h*=9, causing a sharp increase for some averages.) Therefore, we will only consider per-address tables (*h*=2) in subsequent experiments.

3.2.3 Path length

The history pattern consists of target addresses of recently executed branches. The history buffer is shared (global), so all indirect branches influence each other's history. Concatenation with the branch address results in the key used to access the history table. The path length *p* determines the number of branch targets in the history pattern. In theory, longer paths are better since a predictor

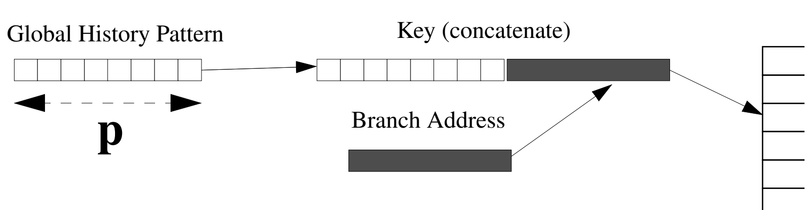


Figure 8. Two level branch prediction

cannot capture regularities in branch behavior with a period longer than *p*. Shorter paths have the advantage that they adapt more quickly to new phases in the branch behavior. A long path captures more regularities, but the number of different patterns mapping to a given target is larger, so it takes longer to fill in the table. This long "warm-up"-time for long patterns can prevent the predictor from taking advantage of longer term correlations before the program behavior changes again. We studied path lengths up to 18 target addresses in order to investigate both trends and see where they combine for the best prediction rate.

Figure 9 shows the impact of the history path length on the misprediction rate for all path lengths from 0 to 18. (A path length of 0 reduces the two-level predictor to a BTB predictor since the key pattern consists of the branch address only.) The average misprediction rate drops quickly from

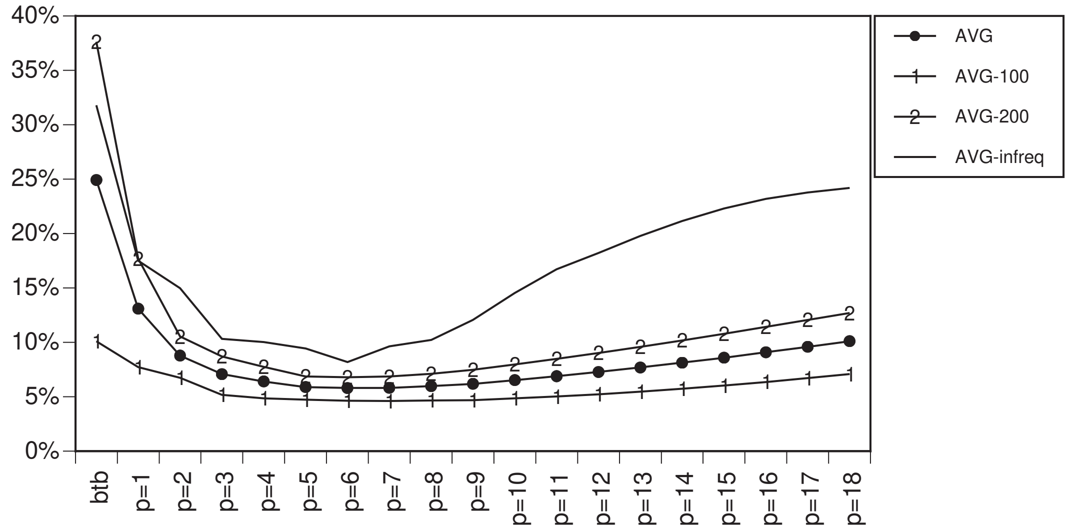


Figure 9. Misprediction rates as a function of path length (global history, per-address table entries)

24.9% for a BTB to 7.8% for p=3 and then slowly reaches a minimum of 5.8% at path length 6. Then the misprediction rate starts to rise again and keeps rising for larger path lengths up to the limit of our testing range at p=18. All benchmark suites follow this pattern, although programs with infrequent branches show uniformly higher misprediction rates.

This result indicates that most regularities in the indirect branch traces have a relatively short period. In other words, a predictable indirect branch execution is usually correlated with the execution of less than three branches before it. Increasing the path length captures some longer term correlations, but at path length six cold-start misses begin to negate the advantage of a longer history. At this point, adding an extra branch target to the path may still allow longer-term correlations to be exploited, but on the other hand it will take the branch predictor longer to learn a full new pattern association for every branch that changes its behavior due to a phase transition in the program. A hybrid branch predictor combining both short and long path components should be able to adapt quickly to phase changes while still exploiting longer-term correlations; we experiment with such hybrid predictors in section 6.

3.3 Variations

We explored a few other choices for the history pattern elements. In the first variant we used both branch address and target, and in the second we included targets of conditional branches in the history. Both resulted in inferior prediction capacity for any pattern length p (see [DH97]).

Although we only showed one-dimensional slices of the design space for two level predictors, we ran an exhaustive simulation for the whole design space as given in Table 4 for a representative subset. We simulated all combinations of *s*, *h*, and *p* for the OO suite (excluding *self* and *beta*, which were not available at the time). We did omit some parameter values that are provably iden-

parameter	meaning	range simulated	discussed in ...
s	history sharing	2..22	section 3.2.1
h	history table sharing	0..22	section 3.2.2
p	history (path) length	0..18	section 3.2.3

Table 4. Summary of two-level indirect branch prediction parameters

tical or nearly identical to others. In particular, s=0 and s=1 are meaningless since instructions are word-aligned, and s=22/h=22 were virtually identical to s=31/h=31 since only few benchmark executables exceeded 2^{21} bytes in size. The exhaustive search gave us an overview of the interesting regions in the parameter space of two-level predictors, establishing limits beyond which the misprediction rate stops improving and allowing use to choose preliminary values for path length, history pattern sharing, and history table sharing (*p*=8, *s*=31, *h*=0). The misprediction rate varies smoothly in the three dimensions of the parameter space in the vicinity of the global minimum, allowing us to minimize one parameter at a time for the full benchmark suite while still staying close to the global minimum. We omit graphs of the exhaustive results for space reasons.

4. Limited-Precision Branch Predictors

The global history pattern is a very long bit pattern. For p=8, it consists of 8 * 32 = 256 bits, and concatenation with the branch address results in a total of 288 bits. The information content of this bit pattern is quite low: the number of different patterns that occur during program execution is much smaller than 2^{288}. Since a tag in an associative table includes most of the pattern, long patterns inflate the size of the predictor table. We need to compress the pattern for each path length into a short bit pattern, ideally without compromising prediction accuracy. As a first step towards smaller history patterns, we will only consider path lengths up to size *p*=12, since longer path lengths result in higher misprediction rates (as seen in Figure 9)

4.1 History Pattern Compression

A straightforward approach for history pattern compression is to select a limited number of bits from each target and concatenate these partial addresses into the history pattern. We explored a number of choices by using a range [*a*..*A*] of the address bits. We varied *a* from 2 to 10, and *A* from *a* to *a*+(*b*-1), where *b* is the largest number of bits that still allows the history pattern to fit within a total of 24 bits (i.e. $b * p <= 24$). Starting with bit *a*=2 worked best on average, and thus we will not show data for other values of a.

Figure 10 shows the misprediction ratios resulting from the selection of bits [2..2+(*b*-1)], for *b* values of 1,2,3,4 and 8, as well as the misprediction rate for full-precision addresses. The curve for *b*=8 almost completely overlaps with the full-address curve, indicating that 8 bits are enough even for short path lengths. For decreasing address precision, shorter path lengths suffer most. For example, for path length p=10, 2 bits achieve a misprediction rate of 6.77% vs. 6.53% for full addresses, while for path length p=3, the miss ratio decreases from 10.6% (2 bits) to 7.1% (full addresses). A total bit length of 24 bits suffices for the history pattern to approach the full-address

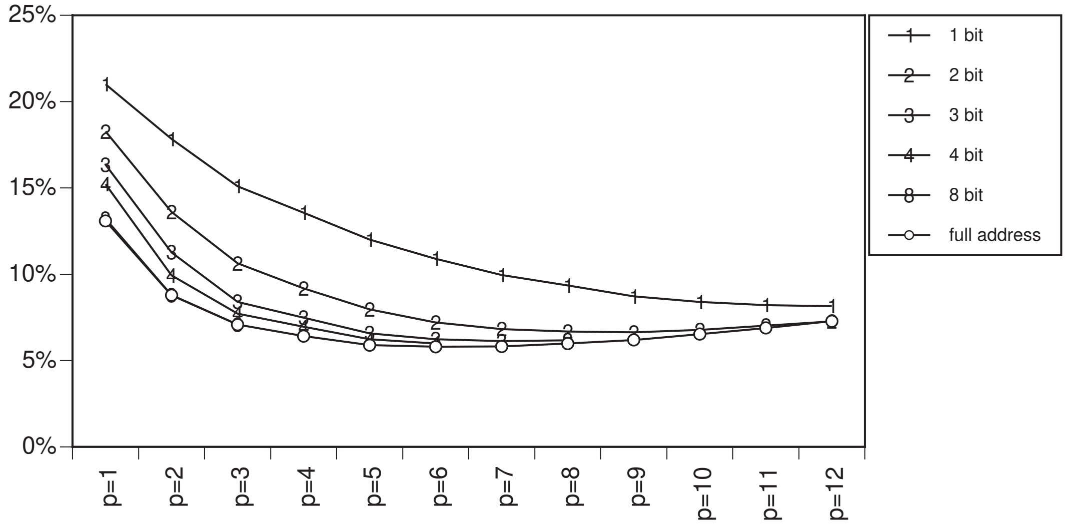


Figure 10. Limited Precision misprediction rates.
AVG for 1, 2, 3, 4, 8 bit and full target addresses.
(low order bits starting from bit 2)

performance for all path lengths. Thus, in the rest of the paper we always choose the largest number *b* of bits from each address that keeps $b * p <= 24$. For example, for path length 2 we choose 12 bits for each history entry, and for path length 6 we choose 4.

We also tried two other schemes for target address compression:

- Fold the new target address into the desired number of *b* bits by dividing it into chunks of *b* bits and xor-ing them all together.
- Shift the history pattern *b* bits to the left and xor with the complete new target address.

These variants were intended to use more information of the target address but did not reliably result in better prediction rates and were sometimes even worse. Since they require more logic than the bit selection discussed above, we decided to drop them from further tests.

4.2 Folding the Branch Address

As mentioned in section 3.3, omitting the branch address reduces the performance of a two-level predictor (for p=8, the misprediction rate increased from 6.0% to 9.6%). However, concatenating the branch address with the history pattern results in a key of 24 + 30 = 54 bits. In analogy with the *Gshare* predictor used in conditional branch prediction [CHP95], we can reduce the number of bits in the key pattern to 30 by xor-ing the branch address with the history pattern. Table 5 shows the

Operation	p=0	p=1	p=2	p=3	p=4	p=5	p=6	p=7	p=8	p=9	p=10	p=11	p=12
Xor	24.91%	13.58%	8.84%	7.09%	6.49%	6.27%	6.01%	6.18%	6.19%	7.44%	7.34%	7.49%	7.67%
Concat	24.91%	13.08%	8.78%	7.08%	6.48%	6.22%	5.99%	6.13%	6.16%	6.62%	6.77%	7.02%	7.27%
Xor-Concat	0.00%	0.50%	0.06%	0.01%	0.01%	0.05%	0.02%	0.05%	0.03%	0.82%	0.57%	0.47%	0.40%

Table 5. Concatenation versus Xor of history pattern with branch address (AVG)

misprediction rate averages for both alternatives. Compared to the increase in misprediction rate due to limited table size and associativity in the next section, the reduction of the key pattern from

54 to 30 bits by xor causes a very small increase in misprediction rate. Since this operation reduces the table space used for tag bits by more than half, we use this scheme in the remainder of the paper.

5. Resource-Constrained Branch Predictors

In this section we introduce limited table sizes and limited associativity in order to obtain practical indirect branch predictors.

5.1 Limited-Size Fully-Associative Tables

Limited tables introduce a new source of branch misses: capacity misses. When the table is too small to store the history patterns of all branches in its working set, some patterns will be evicted from the table, resulting in capacity misses.

Longer path lengths generate more patterns for a given set of branches. For example, *ixx* generates 203 different patterns for path length p=0, 402 for p=1, 865 for p=2, 1469 for p=3, and ends up with 9403 patterns for p=12. Though not all patterns are used more than once (some only occur once in the warm-up phase), for longer path lengths capacity misses will occur fairly soon. A predictor with a longer path length may be more accurate than a predictor of shorter path length for an unlimited table, but the capacity misses caused by a small table size can affect the longer path length predictor enough to negate this advantage.

To estimate the effect of capacity misses we simulate fully-associative tables with LRU replacement policy. Figure 11 shows the average misprediction rate for various fully-associative tables for

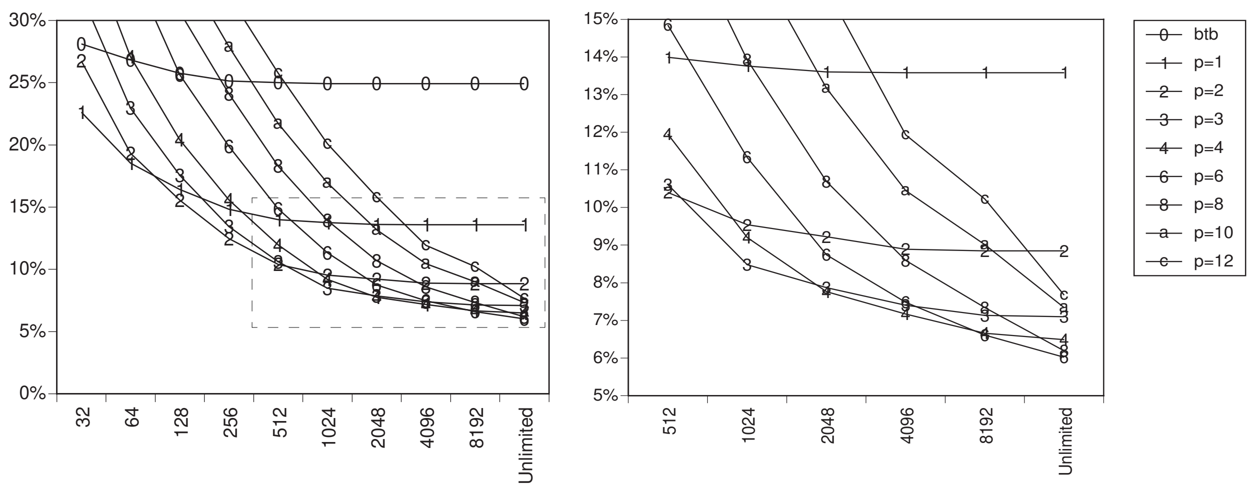


Figure 11. AVG of limited table size fully-associative misprediction ratios (w. LRU replacement policy) (cut-out section enlarged in right graph)

predictors with path length p=0-4,6,8,10 and 12. The misprediction rate of some path lengths reaches its minimum in the explored range. For p=0 (BTB), the miss rate decreases with increasing table size and reaches its minimum at 256 entries. Since there are no capacity misses left, increasing the table size beyond this point will not lower the miss rate for p=0. Increasing the path lengths pushes this point out to 1024 entries (p=1), 2048 entries (p=2), and 8192 entries (p=3 and p=4). Longer path lengths never completely recover from capacity misses in the explored range. A longer path's ability to detect longer-term regularities can pay off, although the best predictor for each table size is still affected by capacity misses. For instance, p=2 wins at table size 256 with a misprediction rate of 12.5%, 3.6% of which is due to capacity misses. For size 1024, p=3 takes over with a misprediction rate of 8.5%, with 1.4% due to capacity misses. For a 8192-entry table, p=6 (which achieved the lowest misprediction rate for an unlimited table) has a misprediction rate of 6.6%, with 0.6% due to capacity misses.

5.2 Limited-Size Limited-Associative Tables

In practice, a fully-associative LRU table of sufficient size requires too much logic to implement in hardware, and thus we will explore limited-associative tables in this section.

Limited associativity means that part of the key pattern is used as an index into a table to access a limited set of entries. Each entry in the set has a tag that is checked for equality to the rest of the key pattern. The index part of the key determines how a working set of branch patterns is spread out over the sets, and how many patterns share the same set. For instance, if one only used the high-order 8 bits of the branch address as index in a BTB of 256 sets, most of the patterns would have to share the same set. This can cause conflict misses; these are similar to capacity misses, but it is the capacity of the set instead of the table that is the limiting factor. Conflict misses can be reduced without changing the total size of the table by increasing the associativity or by choosing a different index scheme, so that different patterns share the same sets.We start out choosing the lower order bits of the key pattern as index. In a two-level predictor, this part contains the lower order branch address bits, xor-ed with the target address bits of the recent targets in the history pattern (see section 4.2).

We test 1, 2 and 4-way associativity, and tagless tables, which is like 1-way associativity but without tags. Where a one-way associative table will register a miss if the search pattern is not in the table, a tagless table will simply return the target corresponding to the index part of the pattern. We compare misprediction rates for equal table sizes, i.e. a table with 256 sets of one entry each (1-way associative) is compared to a table with 64 sets of four entries each (4-way associative).

We tested all table sizes of the previous section, but will show only selected examples for this analysis to reduce the amount of cluttering in the graphs. Figure 12 (a) shows the misprediction rate of different associativities for a 4096-entry table, for all path lengths.

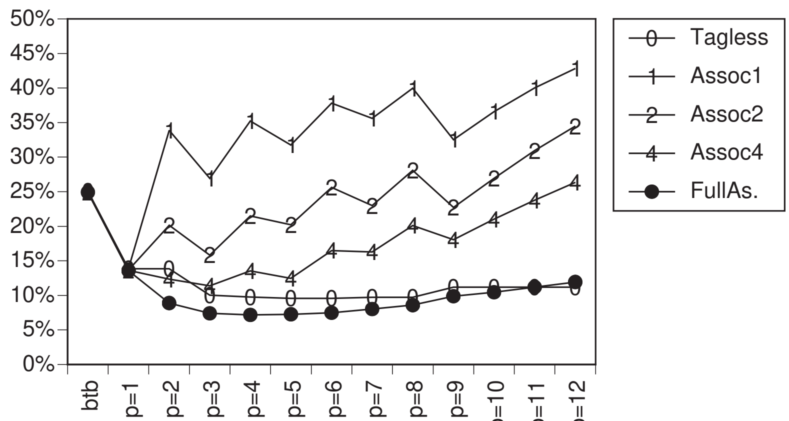


Figure 12. Misprediction rates for a 4096-entry table

5.2.1 Interleaving

The saw-tooth curve for associativities 1, 2 and 4 indicates that there is something wrong with the way the history pattern is assembled from the target address bits. In particular, for associativity one, the misprediction rate of a p=2 predictor is much higher than a p=1 predictor. Figure 13 shows an

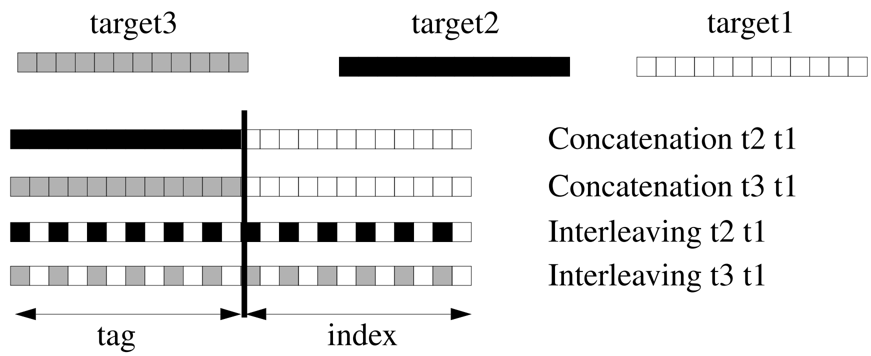


Figure 13. Concatenation and interleaving of target address bits for path length 2 for a 4096-entry table

example for p=2. Since the index part of the pattern is identical for target sequence t2t1 and t3t1, both paths will occupy the same set in the table. The predictor assigns sets in the same way as a predictor of path length one. If the two patterns alternate often, the path length two predictor will incur frequent conflict misses with a one-way associative table and not return a prediction, while the path length one predictor will return the predicted target address. To a lesser degree, the same effect applies to larger path lengths and higher associativities[1], explaining the saw-toothed lines for concatenation in Figure 12. Interleaving remedies this problem by ensuring that the index part of a pattern contains the lower order bits of all target addresses, rather than all bits of a subset of the target addresses. When the target bits are interleaved, target sequences t2t1 and t3t1 will likely differ in the index part of the pattern and will therefore not interfere with each other.

Interleaving of target bits is effective because it spreads patterns over more different sets than concatenation. For example, interleaving increases table utilization for *ixx* from 50% to 79% for a

[1] Also note that since concatenation places the oldest targets completely in the tag, they are invisible to a tagless table. A path length 12 pattern, with two bits per target in a predictor with a tagless, 4096-entry table will use only the 6 most recent targets, so its effective path length is only 6.

1024 entry, one-way associative table for path length four. Figure 14 shows that interleaving dramatically improves predictor performance compared to concatenation.

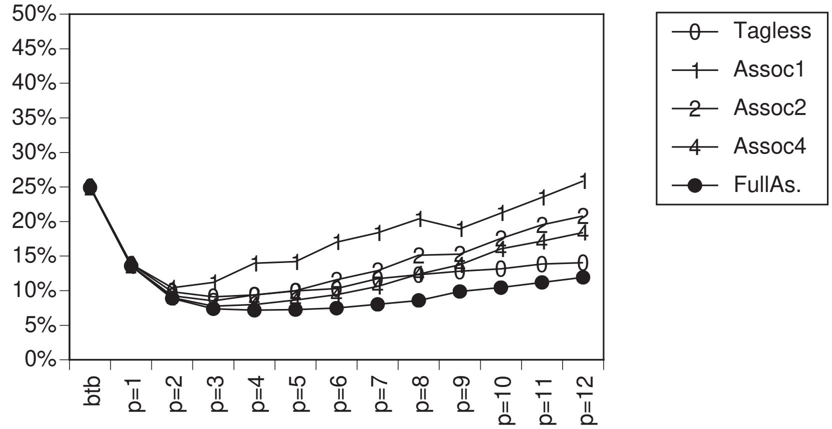


Figure 14. misprediction rates for a 4096-entry table with reverse interleaving

We experimented with three variants of interleaving schemes. Figure 15 shows the interleaving schemes for path length 4 and index length 10. The index part of the pattern contains low order bits

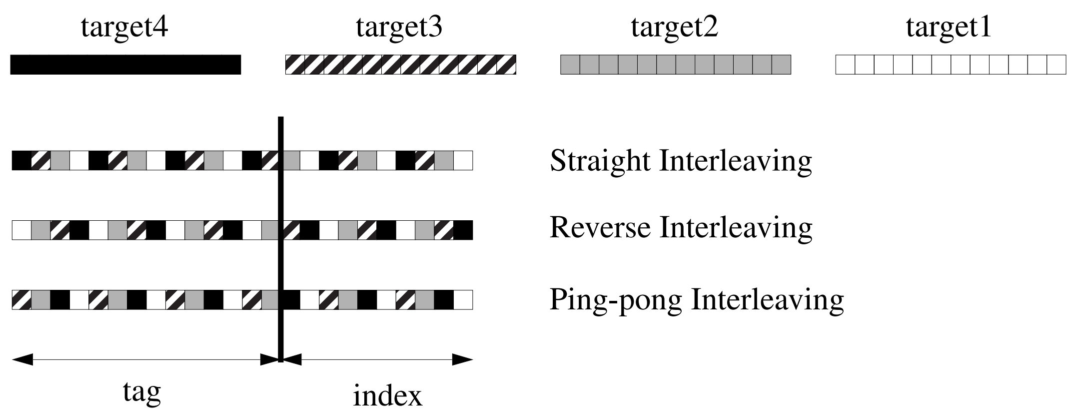


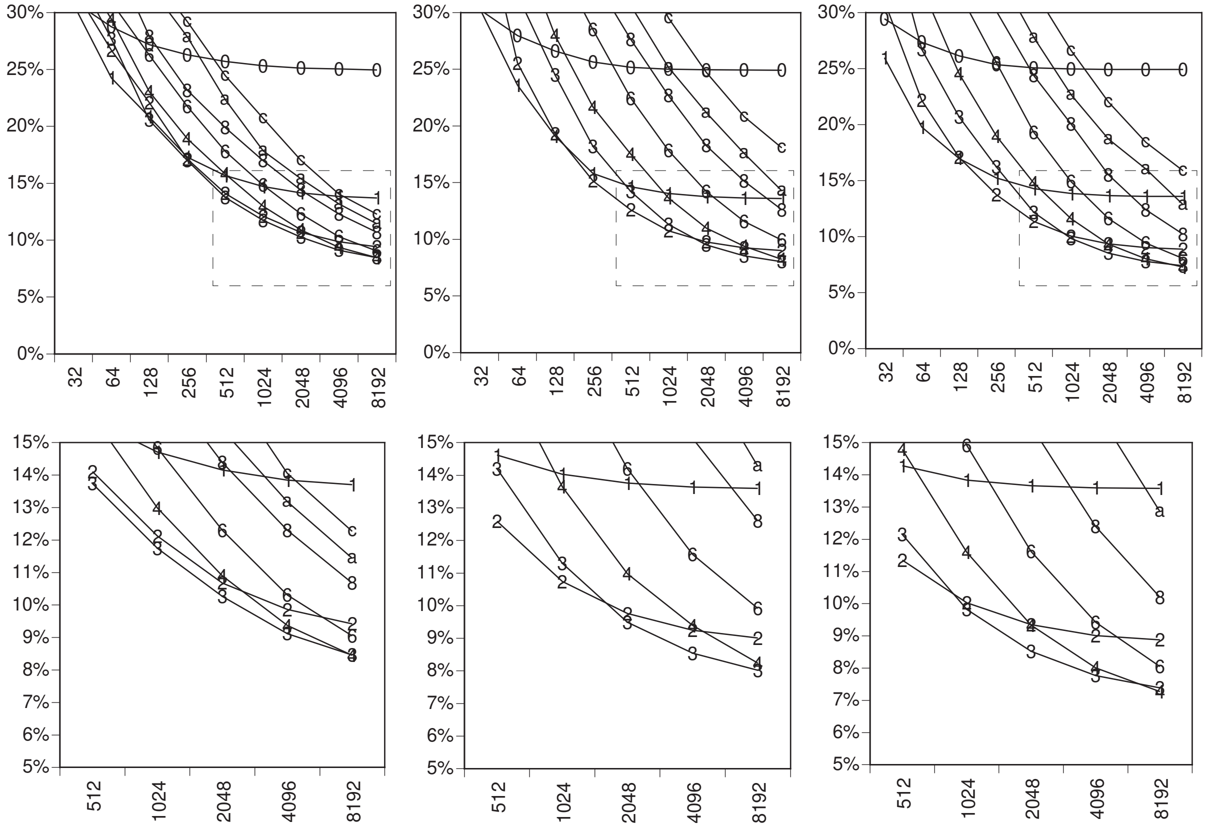
Figure 15. Interleaving schemes for path length 4 with a table of 1024 sets (10-bit index)

from all targets, but two targets are more precisely represented with three bits, and two contribute only their two lower order bits. Straight interleaving represents the most recent targets with higher precision (target 1 and 2), while reverse interleaving represents the older targets most precise (target 3 and 4). Ping-pong interleaving represents both the oldest and youngest target more precise (1 and 4). Suppose the current branch depends only on the address of target4, and some of the possibilities are equal in their two lower order bits. With straight interleaving, the two patterns will conflict. With reverse interleaving, they will use entries in different sets.

We found that reverse interleaving performs slightly better on average than the two other schemes. For shorter path lengths, the order does not make much difference since the index part of the pattern contains many bits from every target. For longer path lengths the difference in precision becomes more important. Reverse interleaving gives longer path length predictors the opportunity to use more exact information from older targets, which is their main advantage compared to shorter path lengths. In the remainder of the paper we use reverse interleaving.

5.2.2 Associativity

Figure 14 shows that for any given table size and path length, higher associativity results in lower misprediction rates. The only exception is the tagless table, which obtains a lower misprediction rate than a four-way associative table for path length 7 to 12. This effect is caused by positive interference. Since these longer path lengths generate a larger set of distinct patterns, conflict misses occur frequently even in four-way associative tables. The tagless table returns its stored target as a prediction even though it may belong to a different pattern, while the associative table registers a miss. Since many patterns map to a small number of targets, the prediction is better than random so that a tagless table can outperform the associative table. Even where tagless tables do worse than



Tagless | Associativity 2 | Associativity 4

Figure 16. AVG misprediction rates for various table sizes, for tagless, 2-way and 4-way associativity. (numbers indicate path length, a = 10, c = 12, bottom graphs enlarge cut-out section)

two- or four-way associative tables, the difference in miss rate remains relatively small. Since associative tables require tags and tag checking logic, the hardware implementation of a tagless table is smaller and faster than its associative counterpart, so that it may be the preferable choice under many circumstances.

Figure 16 shows the AVG misprediction rates for practical associativities. The best predictor for a given table size changes depending on associativity. For tagless tables, p=3 is best for table sizes 128 to 8192. For 2-way associative tables, p=1 wins for size 128, then p=2 is best for sizes 256 to 1024, after which p=3 performs better. For 4-way associativity, the best predictor for every size up to 1024 is the same as for a fully-associative table (see Figure 11). Then p=3 remains the best choice up to table size 4096. At size 8192, p=4 has a slight edge. P=6 retains too many conflict misses even for large table sizes and therefore loses its status as best practical predictor. Limited table size and associativity prevent the predictor from taking full advantage of the longer-term regularity detection capability of longer path length predictors (however, see the next section). Table A-1 in the appendix shows the exact misprediction rates for the best predictors for all table sizes, and Table A-2 contains their path lengths.

6. Hybrid Branch Predictors

As discussed in section 3.2, predictors with short path lengths adapt more quickly when the program goes through a phase change because it doesn't take much time for a short history pattern to fill up. Longer path length predictors are capable of detecting longer-term correlations but take longer to adapt and suffer more from table size limitations because a larger patterns set is mapped to the same number of targets. In this section we combine the two kinds of predictor in a hybrid predictor in order to obtain the advantages of both.

6.1 Metaprediction

A hybrid branch predictor combines two or more component predictors that each predict a target for the current branch. The hybrid predictor employs a selection mechanism (metapredictor) to predict which of the predictors is likely to be correct. A branch predictor selection table (BPST) [McFar93] associates a two-bit counter with each branch to keep track of which of two component predictors is more accurate. After resolving a branch, the counter is updated to reflect the relative accuracy of the two components. Alternatively, branches can be partitioned into different classes based on run-time or compile-time information, and each class is associated with the component predictor best suited to handle it [CHP94].

We attach a "confidence" counter to each table entry to keep track of the number of times the table entry predicted the correct target. The counter is a n-bit saturating counter which tracks the success rate over the last 2^{n-1} times the entry was consulted. (Replacing an entry resets the counter to zero). The hybrid predictor selects the target with the highest confidence value; ties are resolved using a fixed ordering (we test different orders in the next section). This metaprediction scheme is usually more fine-grained than a BPST since it keeps track of the prediction accuracy of a particular pattern rather than a particular branch. We tested 1,2,3 and 4-bit counters for all configurations in the next section. Although the performance difference between 2,3 and 4 bit counters was small, 2-bit counters usually performed best and are used for all results shown.

6.2 Component Predictors

We simulate hybrid predictors with two component predictors of equal table size and associativity but different path lengths. The component table sizes vary from 32 entries to 16K entries, and we simulate all combinations of path lengths in the range 0..12.

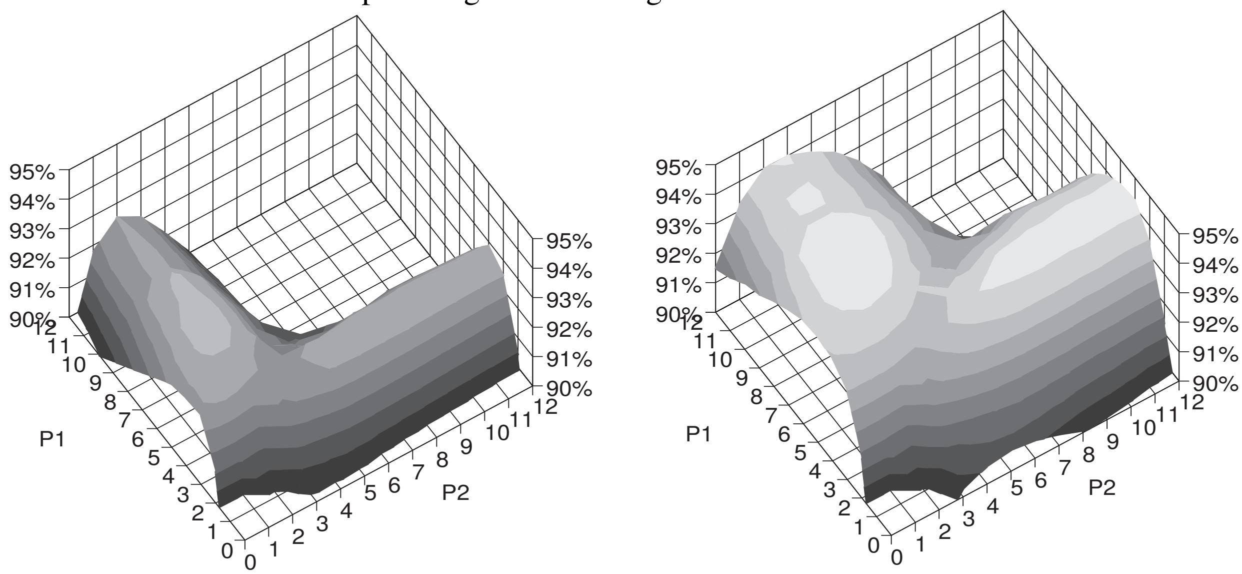


Figure 17. AVG prediction hit rates for hybrid predictors, for all path length combinations, for 4-way associative tables with 2-bit confidence counters and component table size of 2048(left) and 8192(right) entries. P1 is the path length of the first component predictor, P2 of the second. When P1 = P2 (the diagonal), the hit rate for a non-hybrid predictor of twice the component size is shown.

Figure 17 shows the AVG hit ratios for two component table sizes with representative behavior (2048 and 8192); more details are given in Table 6. The best hit rates are obtained by the combination of a short path length predictor (p=1..3) with a longer path length predictor (p=5..12). Since the curve is fairly symmetrical with respect to the diagonal, it appears that the order of the predictors (which is used to break ties in component predictor selection) does not matter much. For smaller tables, the curve is sharper and peaks at shorter path lengths, i.e., it the choice of the short path length component is more important, and very short path lengths do much better.

Figure 18 shows the misprediction rates of the best non-hybrid and hybrid predictors for each table size and associativity. We compare predictors based on total table size, i.e., we treat a hybrid predictor with two component predictors of size N as a predictor of size 2N and compare it against the non-hybrid predictor of that size. In all but one case (64 entry, associativity 4), hybrid predictors obtain lower misprediction rates than equal-sized non-hybrid predictors, even though each component separately suffers more from capacity and conflict misses than the non-hybrid predictor. For smaller table sizes (between 64 and 512 entries), the effect of increased associativity remains stronger than that of hybridization. For example, a non-hybrid 4-way associative table of size 256 achieves a lower misprediction rate than a hybrid predictor with two 2-way associative components of size 128 each. For larger table sizes (between 1K and 32K entries), a hybrid predictor with 2-way associative components performs better than a non-hybrid 4-way associative predictor of the same size. For 2- and 4-way associative non-hybrid predictors with tables larger than 2K entries, the prediction rate improves more by changing to a hybrid predictor than by doubling the total table size. For tables larger than 4K entries, a 4-way associative hybrid predictor outperforms even a fully-associative table of the same size.

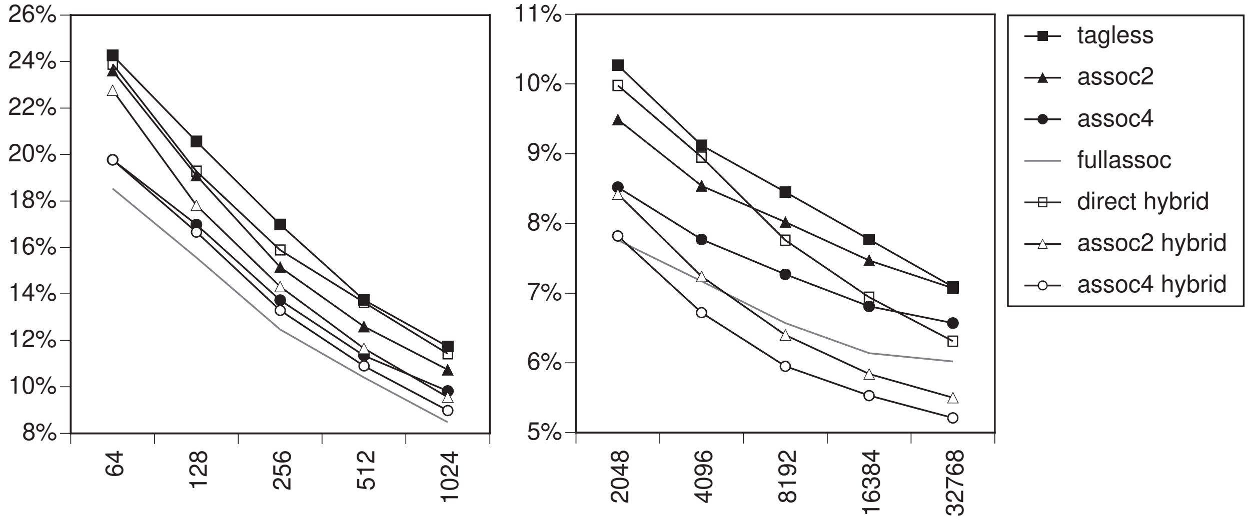


Figure 18. Mispredictions rates for best predictor (choice of path length) for each table size, for tagless, two-way associative and four-way associative tables, hybrid and non-hybrid versions, compared to a fully-associative predictor (table size given in total number of entries)

Table 6 shows the misprediction rate of the best predictor for each table size as well as the component path lengths for which the misprediction rate was achieved. The trend towards longer path lengths with increasing table size is very pronounced; clearly, long paths are ineffective for small predictor tables.

size	tagless		assoc2		assoc4	
	miss%	p1.p2	miss%	p1.p2	miss%	p1.p2
64	23.89%	0.2	22.76%	1.0	19.77%	1[a]
128	19.28%	1.4	17.81%	1.4	16.66%	2.0
256	15.89%	1.3	14.31%	2.1	13.29%	2.0
512	13.64%	3.1	11.65%	3.1	10.90%	3.1
1024	11.42%	3.1	9.56%	3.1	8.98%	3.1
2048	9.98%	3.1	8.42%	4.1	7.82%	5.1
4096	8.95%	3.7	7.24%	5.2	6.72%	6.2
8192	7.76%	3.7	6.40%	6.2	5.95%	6.2
16384	6.94%	3.9	5.84%	7.2	5.53%	7.2
32768	6.31%	3.9	5.50%	7.2	5.21%	8.2

Table 6. AVG misprediction rates and path length combinations for best hybrid predictors

[a] A non-hybrid predictor outperforms all hybrid predictors in this case.

7. Related work

Lee and Smith [LS84] describe several forms of BTBs. Jacobson et al. [J+96] study efficient ways to implement path-based history schemes and observe that BTB hit rates increase substantially when using a global path history. Their Correlated Task Target Buffer (CTTB), unconstrained and fully associative, reached misprediction rates of 18% and 15% for *gcc* and *xlisp* with path length 7; our study found misprediction rates of 12% and 1.5% for p=7. The different results can be explained by several factors: different benchmark version (SPEC92 vs. SPEC95), inputs, and radically different architectures (e.g., the multiscalar processor's history information will likely omit some branches in the immediate past). Finally, Jacobson et al. include conditional branches in the path histories, which is probably responsible for the difference in *xlisp* (see section 3.3).

Chang et al. [CHP97] explore a limited range of two-level predictors for indirect branches and simulate the resulting speedups of selected SPECint95 programs for a superscalar processor. The misprediction rate of a BTB-2bc is reduced by half to 30.9% for *gcc* with a Pattern History Tagless Target Cache with configuration gshare(9). This predictor XORs a global 9-bit history of taken/non taken bits from conditional branches with the branch address, and uses the result as a key into a globally shared, tagless 512-entry history table. In the present study, a comparable non-hybrid predictor (p=3, tagless 512-entry) reaches a misprediction ratio of 31.5% for *gcc*, the best non-hybrid predictor (p=2, four-way associative 512-entry) has 28.1% misprediction rate, and the best hybrid predictor (p1=3, p2=1, four-way associative 512-entry) reaches 26.4%. These comparisons should be regarded with caution, since the two experiments differed in architectures (HPS vs. SPARC), compilers, and benchmark inputs.[1]

Emer and Gloy [EG97] describe several single-level indirect branch predictors based on combinations of the values of PC, SP, register number, and target address, and evaluate their performance on a subset of the SPECint95 programs. For these programs, the best predictor shown achieved a misprediction ratio of 30%, although the authors allude to a better predictor that achieves 15%.

Calder and Grunwald proposed the two-bit counter update rule for BTB target addresses [CG94] and showed that it improved the prediction rate of a suite of C++ programs. Chen et al. [CCM96] propose Partial Prefix Matching prediction for conditional branch prediction and show that a PPM predictor performs better than a two-level predictor for a similar hardware budget. Since a PPM predictor predicts for the longest pattern for which a prediction is available (choosing progressively shorter path lengths until a prediction is found), a hybrid predictor with different path length components can mimic this behavior.

Nair [Nair95] introduced path-based branch correlation for conditional branches and showed that a path-based predictor with two-bit partial addresses attained prediction rates similar to a pattern-based predictor with taken/not taken bits (for similar hardware budgets).

Many alternative implementations in this study were inspired by conditional branch predictors. We refer to [USS97] for a recent general overview, to [YP93] for a classification of two-level predictors, and [ECP96] for recent hybrid prediction results.

[1] We were unable to obtain the exact benchmark inputs used by Chang et al.

8. Conclusions and future work

We have explored a wide range of two-level indirect branch predictors, starting with unconstrained predictors with full-precision addresses and unlimited hardware resources. For a suite of large C++ and C programs totalling more than half a million lines of source code, the best unconstrained predictor achieved a misprediction rate of 5.8%, indicating that indirect branches are intrinsically predictable even though current hardware predictors (BTBs) do not predict them well. An exhaustive search of the design space established that a global history and per-address predictors perform best.

Subsequent experiments introduced resource constraints in order to evaluate whether realistic predictors could approach this performance with a limited hardware budget. Introducing limited-precision addresses (for a history buffer of 24 bits) increased the misprediction rate to 6.0%. Limiting table size (thus causing capacity misses) resulted in a further increase to a 8.5% misprediction rate for a 1K-entry table and 6.6% for a 8K-entry table. Restricting table associativity resulted in 11.7% and 8.5% misprediction rates for 1K and 8K tagless tables, respectively. Four-way associative tables of the same sizes reduce the misprediction rates to 9.8% and 7.3%, respectively. In comparison, an infinite-size fully-associative branch target buffer achieves a best-case misprediction rate of 24.9%. In other words, two-level prediction improves prediction accuracy by more than a factor three.

Combining two-level predictors with different path lengths in a hybrid predictor further improved prediction accuracy. For a 4-way associative table, the misprediction rate of the best hybrid predictor improved to 8.98% for 1K entries and 5.95% for 8K entries. We found that 2-bit per-pattern confidence counters achieve adequate meta-prediction performance and that combining a short and long path length predictor results in the best performance. Compared to an ideal BTB, an 8K-entry hybrid predictor improves prediction accuracy by a factor of more than four.

We also explored a variety of alternatives that resulted in inferior performance. In particular:

- Per-address or per-set history buffers perform worse than a global, shared history buffer.
- Updating targets on every miss lowers the performance, compared to updating only after two consecutive misses.
- Including conditional branch targets in the history pattern lowers prediction performance by pushing the more relevant indirect branch information out of the history buffer.
- Using bits other than the lower-order bits of target addresses results in lower performance.
- For limited-associative tables, the index part of the key pattern should contain bits from as many targets as possible, i.e., interleaving of target address bits performs better than concatenation.

The difference in performance between a BTB and the best practical two-level predictor becomes significant only for history tables larger than 64 entries. As the hardware budget allows larger history tables to be implemented, the path length of the best predictor grows. At 2048 entries, a hybrid predictor's miss rate of 7.8% outperforms that of a BTB by a factor of three. This result suggests that even for very high-ILP processors, indirect branches are less likely to severely constrain the achievable IPC if the transistor budget is large enough.

8.1 Future Work

Hybrid predictors can be further explored. We plan to simulate component predictors of different sizes and combine three or more components. Furthermore, the different components can use one shared table. Entries can be augmented with a “chosen” counter, which keeps track of the number of times an entry’s prediction is used by the hybrid predictor. This counter is consulted when updating table entries, so that seldom used entries can be recuperated by a different component, for better use of available hardware. This would allow each component to only use storage space for the branches it predicts best.

We also plan to study the positive interference observed in tagless, long path length predictors. Since many patterns predict the same target, a denser encoding may exist that uses fewer table entries while preserving prediction accuracy. Also, if a long path length predictor is used as component in a hybrid predictor, the more recent targets may be ignored, reducing the number of patterns that map to the same entries. This will remove the capability of the predictor to predict short term correlations, but these may be adequately predicted by the short path component with which it is combined.

A predictor could predict not only the target of a branch but also the address of the next indirect branch to be executed. This disambiguates branches that lie on different conditional branch control flow paths but share the same indirect branch path, and allows a predictor to run, in principle, arbitrarily far ahead of execution.

Acknowledgments. The authors would like to thank Raimondas Lencevicius for his comments on an early version of this paper, and Ralph Keller for the *jhm* benchmark. This work was supported in part by National Science Foundation CAREER grant CCR-9624458, an IBM Faculty Development Award, and by Sun Microsystems.

9. References

[AH96] Gerald Aigner and Urs Hölzle. Eliminating Virtual Function Calls in C++ Programs. *ECOOP ‘96 Proceedings*, Springer Verlag, July 1996.

[CGZ94] Brad Calder, Dirk Grunwald, and Benjamin Zorn. *Quantifying Behavioral Differences Between C and C++ Programs*. Technical Report CU-CS-698-94, University of Colorado, Boulder, January 1994.

[CG94] Brad Calder and Dirk Grunwald. Reducing indirect function call overhead in C++ programs. In *21st Symposium on Principles of Programming Languages*, pages 397-408, 1994.

[CHP94] Po-Yung Chang, Eric Hao, Yale N. Patt. Branch classification: A new mechanism for improving branch predictor performance. *MICRO ‘27 Proceedings*, November 1994.

[CHP95] Po-Yung Chang, Eric Hao, Yale N. Patt. Alternative Implementations of Hybrid Branch Predictors. *MICRO ‘28 Proceedings*, November 1995.

[CHP97] Po-Yung Chang, Eric Hao, Yale N. Patt. Target Prediction for Indirect Jumps. To appear in the *ISCA’97 Proceedings*.

[CCM96] I-Cheng K.Chen, John T.Coffey, Trevor N. Mudge. Analysis of Branch Prediction via Data Compression. *ASPLOS’96 Proceedings*.

[CK93] Robert F. Cmelik and David Keppel. *Shade: A Fast Instruction-Set Simulator for Execution Profiling*. Sun Microsystems Laboratories, Technical Report SMLI TR-93-12, 1993. Also published as Technical Report CSE-TR 93-06-06, University of Washington, 1993.

[D+96] Jeffrey Dean, Greg DeFouw, David Grove, Vassily Litvinov, and Craig Chambers. Vortex: An Optimizing Compiler for Object-Oriented Languages. *Proceedings of OOPSLA '96*, San Jose, CA, October, 1996.

[DH96] Karel Driesen and Urs Hölzle. The Direct Cost of Virtual Function Calls in C++. In *OOPSLA '96 Conference proceedings*, October 1996.

[DH97] Karel Driesen and Urs Hölzle. Limits of Indirect Branch Prediction. Technical Report TRCS97-10, Department of Computer Science, University of California Santa-Barbara,June 25, 1997 (http://www.cs.ucsb.edu/oocsb/papers/TRCS97-10.html)

[EG97] Joel Emer and Nikolas Gloy. A language for describing predictors and its application to automatic synthesis. To appear in the *ISCA'97 Proceedings,* July 1997.

[ECP96] Marius Evers, Po-Yung Chang, Yale N. Patt. Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the presence of context switches. *Proceedings of ISCA'96*.

[Intel97] Intel press release. *The Next Generation of Microprocessor Architecture: A 64-bit Instruction Set Architecture (ISA) Based on EPIC Technology*. Intel Corporation October 1997 (http://www.intel.com/pressroom/archive/backgrnd/sp101497.HTM)

[J+96] Quinn Jacobson, Steve Bennet, Nikhil Sharma, and James E. Smith. Control flow speculation in multiscalar processors. *HPCA-3 proceedings*, February 1996.

[KE91] David Kaeli and P. G. Emma. Branch history table prediction of moving target branches due to subroutine returns. *ISCA '91 Proceed*ings, May 1991.

[LS84] J. Lee and A. Smith. Branch prediction strategies and branch target buffer design. *IEEE Computer 17(1)*, January 1984.

[Nair95] Ravi Nair. Dynamic Path-Based Branch Correlation. *Proceedings of MICRO-28*, 1995.

[M+94] S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, and W. W. Hwu. Characterizing the Impact of Predicated Execution on Branch Prediction. *Proceedings of the 27th International Symposium on Microarchitecture*, December 1994, pp. 217-227

[MMN93] Ole Lehrmann Madsen, Birger Moller-Pedersen, Kristen Nygaard. *Object-Oriented Programming in the Beta Programming Language*. Addison-Wesley 1993.

[McFar93] S. McFarling, Combining Branch Predictors, *WRL Technical Note TN-36*, Digital Equipment Corporation, June 1993

[P+97] Yale N.Patt, Sanjay J. Patel, Marius Evers, Daniel H. Friendly, Jared Stark. One Billion Transistors, One Uniprocessor, One Chip. *IEEE Computer*, September 1997

[SLM95] Stuart Sechrest, Chich-Chieh Lee, and Trevor Mudge. The role of adaptivity in two-level adaptive branch prediction. *Proceedings of MICRO-29*, November 1995.

[USS97] Augustus K. Uht, Vijay Sindagi, Sajee Somanathan. Branch Effect Reduction Techniques. *IEEE Computer*, May 1997.

[YP91] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive branch prediction. *MICRO 24,* November 1991.

[YP93] Tse-Yu Yeh and Yale N. Patt. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History. *Proceedings of ISCA'93.*

Appendix : Detailed Data

Table A-1 shows the misprediction rates (in %) per benchmark and averages for a fully associative

tablesize	btb fullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4	btbfullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4
	AVG										idl									
32	28.11	30.71	32.50	30.28	25.98	22.62	30.71	32.50	30.28	25.98	6.09	6.76	6.39	6.15	7.87	8.06	6.76	6.39	6.15	7.87
64	26.83	24.26	26.30	23.60	19.77	18.53	23.89	25.45	22.76	19.77	2.99	5.20	6.38	5.55	5.33	5.02	5.39	5.64	5.91	5.33
128	25.76	20.56	22.22	19.09	16.98	15.56	19.28	20.36	17.81	16.66	2.45	4.80	4.54	3.73	2.74	3.77	4.68	5.62	5.17	3.64
256	25.13	16.99	18.06	15.15	13.73	12.47	15.89	15.76	14.31	13.29	2.41	3.40	3.13	3.69	2.67	1.23	3.05	3.52	2.90	2.10
512	25.01	13.74	15.92	12.59	11.35	10.40	13.64	13.03	11.65	10.90	2.40	2.05	3.26	2.18	1.49	1.05	2.44	2.11	1.52	1.17
1024	24.93	11.74	13.53	10.74	9.82	8.48	11.42	10.78	9.56	8.98	2.40	1.43	2.38	1.33	0.85	0.64	1.52	1.29	0.93	0.76
2048	24.92	10.27	11.48	9.49	8.52	7.76	9.98	9.41	8.42	7.82	2.40	1.13	1.80	0.82	0.69	0.62	1.19	0.99	0.67	0.46
4096	24.92	9.12	10.43	8.54	7.77	7.17	8.95	8.46	7.24	6.72	2.40	0.95	1.60	0.71	0.63	0.62	0.81	0.80	0.52	0.38
8192	24.92	8.45	9.68	8.02	7.27	6.57	7.76	7.37	6.40	5.95	2.40	0.69	1.01	0.67	0.63	0.44	0.60	0.61	0.40	0.37
16384	24.92	7.77	8.97	7.47	6.81	6.14	6.94	6.72	5.84	5.53	2.40	0.69	0.85	0.65	0.45	0.42	0.46	0.54	0.37	0.36
32768	24.92	7.09	8.46	7.07	6.57	6.02	6.31	6.26	5.50	5.21	2.40	0.55	0.81	0.50	0.45	0.42	0.43	0.43	0.36	0.35
	100-AVG										ijpeg									
32	14.97	18.92	18.07	16.82	16.81	15.86	18.92	18.07	16.82	16.81	1.26	26.62	1.52	1.64	0.82	0.32	26.62	1.52	1.64	0.82
64	13.31	15.73	16.96	14.68	14.29	13.76	16.14	15.98	14.69	14.29	1.26	26.63	51.15	0.31	0.31	0.31	1.41	0.54	0.41	0.31
128	11.72	15.01	13.98	12.33	11.77	13.63	15.17	15.45	14.03	12.35	1.26	0.92	0.56	0.31	0.31	0.39	26.14	34.23	0.31	0.29
256	10.57	12.49	12.01	12.11	11.20	9.85	12.32	12.07	11.57	10.08	1.26	0.69	0.56	0.65	0.39	0.39	0.57	0.56	0.35	0.29
512	10.32	9.84	12.27	9.84	9.09	8.20	10.16	9.46	8.63	8.22	1.26	0.44	0.39	0.39	0.39	0.39	0.48	0.49	0.37	0.37
1024	10.14	8.42	10.77	8.48	7.15	6.37	8.31	7.82	7.02	6.66	1.26	0.45	0.39	0.39	0.46	0.46	0.36	0.37	0.37	0.37
2048	10.11	7.41	9.21	6.97	6.37	6.12	7.20	6.77	6.18	5.96	1.26	0.45	0.39	0.46	0.46	0.51	0.36	0.37	0.38	0.38
4096	10.11	6.69	8.54	6.29	5.77	5.62	6.71	6.66	5.74	5.47	1.26	0.44	0.39	0.46	0.46	0.51	0.44	0.46	0.46	0.47
8192	10.11	6.21	6.91	5.92	5.62	5.26	5.88	5.89	5.13	4.83	1.26	0.50	0.46	0.46	0.51	0.56	0.44	0.46	0.47	0.47
16384	10.11	5.95	6.43	5.69	5.42	4.93	5.36	5.31	4.74	4.54	1.26	0.56	0.46	0.51	0.56	0.62	0.43	0.46	0.47	0.47
32768	10.11	5.56	6.10	5.68	5.26	4.86	4.92	4.90	4.46	4.42	1.26	0.56	0.46	0.56	0.56	0.62	0.43	0.47	0.47	0.48
	200-AVG										ixx									
32	39.38	40.82	44.86	41.81	33.84	28.40	40.82	44.86	41.81	33.84	46.58	33.15	56.21	51.11	30.07	24.30	33.15	56.21	51.11	30.07
64	38.42	31.58	34.30	31.24	24.48	22.61	30.54	33.58	29.68	24.48	45.75	24.47	32.91	24.86	18.10	15.94	31.72	30.00	23.33	18.10
128	37.79	25.32	29.28	24.89	21.44	17.22	22.79	24.57	21.06	20.36	45.70	24.60	27.55	17.93	15.37	12.10	22.09	25.43	20.18	19.02
256	37.61	20.84	23.24	17.77	15.91	14.72	18.95	18.92	16.67	16.04	45.70	18.74	20.58	15.47	12.57	10.34	17.91	18.55	14.13	13.32
512	37.61	17.08	19.05	14.94	13.29	12.28	16.63	16.09	14.23	13.20	45.70	13.88	21.38	13.48	10.63	10.25	14.28	13.72	10.84	9.48
1024	37.61	14.58	15.90	12.67	12.10	10.28	14.08	13.32	11.73	10.98	45.70	12.37	15.32	11.57	9.03	8.21	11.89	11.63	9.53	8.56
2048	37.61	12.73	13.43	11.65	10.36	9.16	12.37	11.67	10.34	9.42	45.70	10.90	12.51	9.82	8.47	6.94	10.81	10.54	7.43	6.06
4096	37.61	11.21	12.05	10.46	9.47	8.51	10.86	10.00	8.52	7.80	45.70	9.34	12.11	9.29	8.29	6.94	8.13	6.87	5.96	5.06
8192	37.61	10.37	12.06	9.82	8.69	7.69	9.37	8.64	7.49	6.91	45.70	8.86	11.28	8.94	7.11	5.86	7.32	6.45	5.37	4.91
16384	37.61	9.32	11.15	9.00	8.00	7.18	8.30	7.93	6.79	6.38	45.70	7.66	10.55	8.04	5.98	5.58	5.98	6.12	4.87	4.70
32768	37.61	8.39	10.48	8.26	7.68	7.01	7.50	7.42	6.39	5.88	45.70	6.79	10.14	6.70	5.94	5.58	5.71	5.58	4.81	4.15
	INFREQ-AVG										lcom									
32	31.78	31.68	32.32	31.88	20.85	17.95	31.68	32.32	31.88	20.85	5.18	12.99	5.80	5.24	4.81	4.45	12.99	5.80	5.24	4.81
64	31.78	27.61	34.80	18.16	17.74	17.54	27.65	24.51	19.88	17.74	4.71	4.67	4.78	4.03	3.84	3.62	5.11	4.84	4.30	3.84
128	31.78	23.16	20.75	17.96	17.54	15.66	26.24	28.73	16.79	21.46	4.46	4.75	3.90	3.61	3.47	3.49	4.48	4.31	3.73	3.72
256	31.78	19.92	18.07	17.52	15.58	14.98	15.74	16.96	15.37	17.03	4.25	3.93	3.43	3.37	3.25	3.27	3.73	3.35	3.36	2.96
512	31.78	15.26	20.34	16.44	15.18	14.97	14.72	14.12	11.98	11.31	4.25	3.25	3.26	2.80	2.60	2.53	3.18	2.83	2.66	2.56
1024	31.78	14.27	19.40	16.09	10.89	10.40	13.04	12.49	11.42	10.43	4.25	2.68	2.77	2.33	2.20	1.88	2.61	2.26	2.07	1.95
2048	31.78	13.54	18.06	12.86	10.67	10.16	12.24	11.89	10.06	10.09	4.25	2.17	2.43	1.85	1.68	1.65	2.14	1.86	1.73	1.69
4096	31.78	12.04	16.48	11.87	10.44	10.06	11.29	11.23	9.81	8.31	4.25	1.83	2.12	1.56	1.42	1.34	2.03	1.71	1.46	1.31
8192	31.78	12.12	14.41	11.76	10.50	10.81	10.29	10.10	8.02	7.84	4.25	1.62	1.71	1.44	1.36	1.34	1.69	1.40	1.20	1.11
16384	31.78	12.66	14.17	11.67	11.22	9.17	9.61	9.47	8.63	8.60	4.25	1.55	1.58	1.37	1.37	1.39	1.45	1.24	1.11	1.04
32768	31.78	11.14	13.98	11.89	11.07	9.10	9.47	9.20	8.30	8.90	4.25	1.43	1.49	1.39	1.35	1.39	1.31	1.20	1.05	1.02

Table A-1. Misprediction rates (per benchmark and averages) for different predictors and table sizes.

tablesize	btb fullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4	btbfullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4
	OO-AVG										m88ksim									
32	23.70	25.80	28.17	25.96	22.64	20.02	25.80	28.17	25.96	22.64	76.41	56.87	78.29	76.41	45.33	35.99	56.87	78.29	76.41	45.33
64	22.18	20.43	22.55	18.96	17.18	16.04	22.35	21.62	19.12	17.18	76.41	46.52	49.25	35.99	35.99	35.99	65.19	56.77	42.54	35.99
128	20.87	20.75	18.40	15.33	14.13	15.43	19.14	19.63	17.20	15.47	76.41	43.78	46.40	35.99	35.99	26.55	42.50	44.76	33.11	46.11
256	19.97	17.00	15.48	14.79	13.29	11.95	15.35	15.01	13.61	12.16	76.41	36.17	35.99	29.46	26.55	26.55	28.66	34.39	28.57	32.27
512	19.81	13.37	15.30	11.95	10.61	9.50	13.29	12.47	11.13	10.50	76.41	22.07	32.29	27.54	26.55	26.55	23.90	23.26	17.53	15.38
1024	19.69	11.29	12.63	9.91	9.47	7.93	10.85	10.15	9.01	8.49	76.41	20.16	30.44	27.54	14.40	14.40	19.26	18.53	16.47	13.56
2048	19.67	9.81	10.51	9.14	8.09	7.21	9.42	8.82	7.97	7.50	76.41	19.31	30.43	17.48	14.40	13.40	17.35	17.47	11.57	8.72
4096	19.67	8.67	9.68	8.21	7.38	6.70	8.51	7.97	6.97	6.48	76.41	15.41	30.43	14.58	14.40	13.40	11.62	10.73	8.75	2.17
8192	19.67	8.03	9.32	7.75	6.90	6.31	7.37	7.06	6.14	5.76	76.41	14.39	16.40	14.40	13.40	10.58	9.78	10.73	2.17	2.17
16384	19.67	7.43	8.57	7.21	6.49	5.91	6.51	6.42	5.62	5.35	76.41	14.43	16.40	13.40	10.58	3.07	6.73	8.76	4.00	4.00
32768	19.67	6.82	8.00	6.76	6.33	5.87	5.94	5.97	5.31	5.06	76.41	9.64	16.40	10.58	10.58	3.07	6.73	6.76	4.00	4.00
	C-AVG										perl									
32	34.91	36.72	37.28	35.94	27.18	23.20	36.72	37.28	35.94	27.18	31.80	60.62	31.81	31.80	36.31	22.80	60.62	31.81	31.80	36.31
64	34.54	30.25	34.76	26.10	21.67	20.83	27.51	29.29	25.41	21.67	31.80	45.24	45.27	49.74	22.76	22.75	9.32	31.76	31.75	22.76
128	34.27	21.65	25.78	22.75	20.46	15.76	22.92	25.36	17.99	20.41	31.80	0.37	40.74	36.21	22.72	0.34	0.33	4.85	0.40	0.29
256	34.25	18.43	20.96	16.75	15.16	14.31	16.42	17.20	15.63	16.43	31.80	0.35	22.73	0.33	0.32	0.29	0.28	0.30	0.28	0.26
512	34.25	14.90	18.84	15.23	14.10	13.69	14.58	14.20	12.40	11.56	31.80	0.31	0.33	0.32	0.29	0.28	0.27	0.30	0.28	0.27
1024	34.25	13.50	17.48	14.34	10.75	10.05	12.87	12.34	11.11	10.26	31.80	0.31	0.32	0.30	0.34	0.32	0.26	0.27	0.26	0.26
2048	34.25	12.43	15.87	11.56	10.08	9.57	11.74	11.31	9.75	9.31	31.80	0.30	0.31	0.34	0.33	0.37	0.25	0.26	0.27	0.27
4096	34.25	11.09	14.31	10.58	9.54	9.15	10.61	10.40	8.82	7.79	31.80	0.30	0.30	0.33	0.32	0.37	0.29	0.33	0.32	0.33
8192	34.25	10.75	12.45	10.19	9.31	8.98	9.46	9.08	7.50	7.12	31.80	0.34	0.35	0.33	0.37	0.40	0.29	0.33	0.33	0.32
16384	34.25	10.60	12.02	9.87	9.38	7.91	8.76	8.43	7.49	7.26	31.80	0.38	0.34	0.37	0.40	0.45	0.29	0.32	0.33	0.33
32768	34.25	9.42	11.73	9.82	9.08	7.72	8.31	8.06	7.11	7.22	31.80	0.38	0.33	0.41	0.40	0.45	0.30	0.33	0.32	0.34
	SPEC-AVG										porky									
32	34.05	35.89	34.75	34.29	25.64	21.58	35.89	34.75	34.29	25.64	21.70	28.88	26.38	23.76	22.52	19.41	28.88	26.38	23.76	22.52
64	34.04	30.36	34.64	25.27	20.80	20.39	26.32	27.33	24.13	20.80	21.21	19.74	21.75	17.41	15.30	13.97	21.81	21.97	19.21	15.30
128	34.02	20.11	25.21	22.61	20.19	14.91	22.26	24.53	16.54	19.45	21.08	19.24	17.53	13.97	13.15	9.68	17.41	18.88	15.02	16.11
256	34.02	17.20	20.64	15.99	14.49	13.68	15.32	16.08	14.79	15.82	20.80	14.94	15.96	12.43	10.22	9.12	13.56	13.68	10.75	11.04
512	34.02	13.72	18.12	14.80	13.78	13.48	13.52	13.26	11.47	10.70	20.80	12.28	12.53	10.16	9.01	8.90	11.99	11.40	9.94	9.28
1024	34.02	12.63	16.96	14.22	9.95	9.43	12.00	11.70	10.56	9.67	20.80	10.37	10.03	8.56	7.89	6.97	10.06	8.74	7.81	7.59
2048	34.02	11.72	15.46	11.00	9.49	9.02	11.05	10.81	9.23	8.99	20.80	9.30	8.96	7.78	6.88	5.13	8.97	7.84	7.04	6.99
4096	34.02	10.39	14.23	10.05	9.01	8.64	10.08	9.83	8.51	7.44	20.80	8.39	8.43	7.24	6.26	4.85	7.80	6.81	6.01	5.41
8192	34.02	10.12	11.76	9.71	8.67	8.56	8.97	8.76	7.09	6.76	20.80	7.22	9.13	6.98	5.30	5.05	6.83	6.20	5.36	5.05
16384	34.02	10.22	11.39	9.26	8.79	7.35	8.31	8.10	7.23	7.01	20.80	6.07	7.68	5.64	5.22	4.61	6.08	5.56	5.13	4.77
32768	34.02	8.95	11.13	9.32	8.51	7.13	7.92	7.73	6.82	7.05	20.80	5.37	6.53	5.19	4.99	4.61	5.29	5.76	4.92	4.29
	beta										self									
32	31.85	27.98	36.63	33.76	25.20	16.89	27.98	36.63	33.76	25.20	36.08	49.54	48.36	46.15	45.21	40.81	49.54	48.36	46.15	45.21
64	30.48	21.88	23.13	19.59	14.73	12.22	25.92	24.17	20.58	14.73	32.34	44.08	46.73	39.69	38.86	36.39	42.34	44.11	41.04	38.86
128	29.44	18.65	16.62	14.66	10.77	13.06	17.05	16.85	14.69	13.03	25.07	39.80	39.03	32.54	31.61	36.84	41.43	42.17	37.80	34.00
256	28.57	13.88	12.88	13.04	10.99	10.44	12.69	12.09	11.33	10.17	18.41	33.05	32.08	31.53	29.44	26.91	33.49	32.42	31.82	26.15
512	28.57	10.20	13.75	9.38	7.46	5.51	10.09	9.59	8.52	8.36	16.94	26.80	29.39	23.94	22.18	19.70	27.35	26.31	24.20	23.26
1024	28.57	7.84	10.64	6.04	5.61	3.19	7.21	6.53	5.47	4.92	15.88	22.37	24.27	19.48	18.61	16.62	22.53	20.60	18.00	17.11
2048	28.57	6.29	6.19	5.13	3.77	2.66	5.46	4.79	3.84	3.23	15.68	18.33	19.69	16.81	15.40	16.01	18.33	16.28	15.21	14.64
4096	28.57	4.42	4.78	4.12	3.28	2.63	4.45	3.78	2.77	2.20	15.68	15.49	16.84	14.03	12.90	13.44	17.94	16.31	14.06	13.38
8192	28.57	3.77	4.91	3.62	2.73	2.46	3.31	2.95	2.27	1.99	15.68	14.00	15.43	12.27	11.91	11.54	14.62	13.50	11.81	10.83
16384	28.57	3.49	4.46	2.81	2.51	2.28	2.58	2.58	2.03	1.88	15.68	13.51	13.73	11.55	11.43	10.50	12.44	11.15	9.92	9.25
32768	28.57	2.91	4.18	2.79	2.49	2.28	2.28	2.32	1.87	1.68	15.68	11.87	12.63	11.51	10.72	10.10	10.62	10.54	9.16	8.61

Table A-1. Misprediction rates (per benchmark and averages) for different predictors and table sizes.

tablesize	btb fullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4	btbfullassoc	tagless	assoc1	assoc2	assoc4	fullassoc	hybrid tagless	hybrid assoc1	hybrid assoc2	hybrid assoc4
	edg										jhm									
32	40.88	42.59	54.99	47.48	37.97	34.57	42.59	54.99	47.48	37.97	11.45	14.05	12.24	11.81	14.40	14.03	14.05	12.24	11.81	14.40
64	37.97	29.48	35.61	31.89	27.78	23.89	35.85	42.99	34.40	27.78	11.16	12.92	15.14	13.03	12.88	12.49	12.54	12.19	12.04	12.88
128	35.99	32.48	29.80	23.77	22.36	21.72	27.51	31.20	28.14	27.08	11.13	15.12	12.22	11.96	11.76	13.36	13.08	13.93	12.81	11.81
256	35.91	27.09	23.22	22.07	19.86	18.73	24.12	25.05	21.50	20.72	11.13	13.81	11.80	12.98	12.40	11.31	11.97	11.70	12.08	11.35
512	35.91	23.17	23.84	18.18	16.36	15.18	21.98	20.76	18.85	17.57	11.13	12.75	14.64	12.18	11.27	10.59	11.95	11.88	11.43	11.00
1024	35.91	19.63	21.13	15.16	16.34	14.40	18.93	16.80	14.94	14.46	11.13	11.93	13.57	11.18	11.33	9.67	11.24	11.30	10.81	10.35
2048	35.91	17.40	18.73	15.53	14.19	13.46	16.57	14.86	13.43	11.59	11.13	11.59	13.03	12.10	10.92	9.41	10.75	10.94	10.23	9.85
4096	35.91	15.94	14.82	14.27	13.28	12.75	14.31	14.41	11.04	10.20	11.13	11.08	12.83	11.42	10.22	9.30	10.07	10.29	9.24	9.16
8192	35.91	15.10	17.25	13.58	13.81	11.94	12.92	11.34	10.36	9.62	11.13	11.12	11.89	11.20	10.75	9.44	9.39	9.85	8.62	8.45
16384	35.91	13.24	16.39	14.15	13.52	11.86	11.90	10.75	9.32	9.00	11.13	10.33	11.40	11.38	10.29	8.75	8.89	9.31	8.58	8.40
32768	35.91	12.65	15.97	13.37	13.06	11.86	11.06	10.33	9.14	8.38	11.13	10.14	10.89	10.51	10.21	8.75	8.54	8.20	7.90	8.32
	eqn										troff									
32	36.87	37.65	39.37	37.60	33.44	32.77	37.65	39.37	37.60	33.44	17.50	21.21	22.12	18.05	20.20	19.48	21.21	22.12	18.05	20.20
64	35.82	32.30	32.68	29.38	29.16	28.00	35.99	31.82	29.57	29.16	15.16	18.57	19.47	17.08	16.45	16.70	20.30	19.85	16.12	16.45
128	34.78	38.22	28.85	26.17	25.56	29.27	33.25	31.42	28.89	24.95	13.70	21.56	15.39	13.44	12.72	17.32	18.76	18.07	16.52	12.91
256	34.78	34.08	26.35	27.28	25.97	25.58	27.99	26.61	24.53	21.95	13.70	17.20	13.14	13.27	12.06	9.38	13.76	13.20	11.61	10.43
512	34.78	28.22	26.77	23.17	21.20	18.69	26.06	24.72	22.39	21.04	13.70	10.93	12.69	10.28	9.64	8.29	12.24	9.70	8.67	8.37
1024	34.78	23.41	23.22	19.72	21.93	16.89	22.20	21.29	19.25	18.16	13.70	9.24	11.50	8.96	7.76	7.33	8.38	7.76	7.24	7.00
2048	34.78	20.18	19.82	20.29	17.62	15.27	19.65	19.09	18.59	17.85	13.70	8.39	10.17	7.68	7.37	7.20	7.46	7.00	6.95	6.75
4096	34.78	18.60	18.69	18.02	16.08	13.99	18.08	17.33	15.66	14.68	13.70	7.93	9.71	7.48	7.34	7.20	7.25	7.79	7.08	6.74
8192	34.78	17.22	20.24	17.23	15.01	13.52	15.64	15.55	13.42	12.55	13.70	7.79	8.29	7.41	7.26	7.13	6.88	7.05	6.82	6.54
16384	34.78	15.95	18.79	16.08	13.87	12.56	14.05	14.52	12.02	11.39	13.70	7.59	8.12	7.35	7.25	7.15	6.70	6.79	6.53	6.40
32768	34.78	14.81	17.39	14.74	13.71	12.56	12.71	12.84	11.26	10.61	13.70	7.45	7.91	7.53	7.16	7.15	6.56	6.88	6.50	6.49
	gcc										vortex									
32	65.96	54.84	68.66	67.18	51.35	48.07	54.84	68.66	67.18	51.35	20.19	19.08	20.21	20.22	13.01	10.98	19.08	20.21	20.22	13.01
64	65.89	47.92	48.75	45.84	43.52	41.52	53.17	52.32	48.93	43.52	20.19	13.42	14.88	12.67	11.30	10.71	15.11	17.13	12.93	11.30
128	65.74	43.71	43.90	41.51	40.11	34.36	41.91	43.35	40.10	42.02	20.19	14.86	12.38	12.01	10.69	12.17	10.94	12.22	10.36	14.92
256	65.70	36.81	40.94	33.73	31.43	28.55	36.09	36.19	34.16	34.80	20.19	13.16	12.18	16.78	13.10	12.08	10.15	10.17	10.61	12.78
512	65.70	31.49	34.77	29.89	28.09	27.15	31.71	32.16	28.83	26.38	20.19	12.09	25.94	16.40	12.56	12.07	10.70	9.72	7.97	7.68
1024	65.70	28.12	30.68	27.31	23.59	21.96	28.03	27.95	24.88	22.89	20.19	11.37	24.43	15.16	7.12	6.36	10.22	9.20	7.86	7.63
2048	65.70	24.72	27.50	22.64	21.29	20.31	24.85	24.30	21.82	19.97	20.19	11.29	19.80	11.68	6.89	5.90	9.77	8.78	6.99	7.81
4096	65.70	21.45	25.23	19.97	18.82	18.02	22.98	20.49	17.90	16.70	20.19	11.03	13.60	11.65	6.39	5.90	9.55	11.90	9.09	8.24
8192	65.70	20.04	21.24	18.07	16.50	14.60	19.26	17.64	15.31	13.95	20.19	10.83	19.57	11.65	7.12	10.98	9.38	8.16	7.69	7.50
16384	65.70	18.48	19.83	15.92	14.49	12.93	17.26	15.65	13.81	12.56	20.19	11.92	19.02	11.48	11.49	9.89	9.35	8.06	7.87	8.12
32768	65.70	15.83	18.79	14.64	13.20	11.71	15.16	14.81	12.41	11.72	20.19	11.34	18.34	12.63	11.38	9.89	9.33	7.94	7.58	8.15
	go										xlisp									
32	29.25	24.17	29.25	29.25	24.25	24.52	24.17	29.25	29.25	24.25	13.51	9.00	13.51	13.51	8.39	8.36	9.00	13.51	13.51	8.39
64	29.25	23.87	23.92	23.66	23.38	23.13	28.88	23.58	23.64	23.38	13.51	8.93	9.25	8.71	8.35	8.34	11.15	9.24	8.70	8.35
128	29.25	33.07	23.66	23.52	23.16	23.51	25.38	23.70	23.38	24.52	13.51	4.03	8.82	8.69	8.34	7.03	8.61	8.58	8.11	8.02
256	29.25	29.67	23.57	23.18	22.27	20.89	23.59	22.71	21.96	22.80	13.51	3.53	8.48	7.80	7.36	7.03	7.89	8.22	7.64	7.50
512	29.25	26.42	22.72	21.44	21.21	20.88	23.80	22.99	22.04	21.80	13.51	3.23	10.41	7.66	7.35	7.03	3.77	3.90	3.29	2.98
1024	29.25	25.08	22.34	21.25	21.57	20.39	22.32	21.86	20.98	20.18	13.51	2.90	10.12	7.62	2.17	2.09	3.59	3.73	3.09	2.77
2048	29.25	23.13	21.61	21.84	20.95	20.83	21.47	20.95	21.32	23.44	13.51	2.85	8.17	2.54	2.15	1.81	3.30	3.52	2.26	2.35
4096	29.25	21.29	21.51	20.81	20.52	20.45	23.56	21.84	20.93	22.37	13.51	2.84	8.15	2.53	2.14	1.81	2.15	3.04	2.10	1.83
8192	29.25	22.75	21.19	20.52	20.97	21.10	21.55	21.07	21.74	21.21	13.51	2.01	3.11	2.53	1.81	1.67	2.09	2.96	1.93	1.71
16384	29.25	23.73	20.80	21.31	22.23	23.09	21.91	20.60	22.19	21.81	13.51	2.06	2.90	1.83	1.76	1.37	2.22	2.84	1.95	1.78
32768	29.25	23.00	20.74	23.76	21.75	22.82	21.40	21.64	21.16	22.95	13.51	1.92	2.84	2.62	1.71	1.37	2.08	2.16	1.81	1.74

Table A-1. Misprediction rates (per benchmark and averages) for different predictors and table sizes.

BTB, two-level predictors with tagless, one-way, two-way, four-way and fully associative tables of given entry size, and dual pathlength hybrid predictors with 2-bit confidence counters. The pathlengths for the best predictor and component predictors is given in Table A-2. Note that the misprediction rates of individual benchmarks in Table A-1 may occasionally increase even for a larger table size, since the pathlength is chosen to minimize the AVG misprediction rate, and some benchmarks go against the general trend, especially for small tables of low associativity.

predictor type	32	64	128	256	512	1024	2048	4096	8192	16384	32768
tagless	1	1	3	3	3	3	3	3	4	5	5
assoc2	0	1	1	2	2	2	3	3	3	4	5
assoc4	1	1	1	2	2	3	3	3	4	5	5
fullassoc	1	1	2	2	2	3	4	4	5	6	6
hybrid tagless	1	0.2	1.4	1.3	3.1	3.1	3.1	3.7	3.7	3.9	3.9
hybrid assoc2	0	1.0	1.4	2.1	3.1	3.1	4.1	5.2	6.2	7.2	7.2
hybrid assoc4	1	1	2.0	2.0	3.1	3.1	5.1	6.2	6.2	7.2	8.2

Table A-2. Path length of best predictor for each associativity. Hybrid predictors have two path lengths, one for each component. When the same-size non-hybrid predictor is better, we only give the single path length.