Multi-stage Cascaded Prediction

Karel Driesen and Urs Hölzle
Department of Computer Science
University of California
Santa Barbara, CA 93106
{karel,urs}@cs.ucsb.edu
http://www.cs.ucsb.edu/oocsb

Technical Report TRCS99-05
February 12, 1999

Abstract. Two-level predictors deliver highly accurate conditional branch prediction, indirect branch target prediction and value prediction. Accurate prediction enables speculative execution of instructions, a technique that increases instruction level parallelism. Unfortunately, the accuracy of a two-level predictor is limited by the cost of the predictor table that stores associations between history patterns and target predictions. Two-stage cascaded prediction, a recently proposed hybrid prediction architecture, uses pattern filtering to reduce the cost of this table while preserving prediction accuracy. In this study we generalize two-stage prediction to multi-stage prediction. We first determine the limit of accuracy on an indirect branch trace using a multi-stage predictor with an unlimited hardware budget. We then investigate practical cascaded predictors with limited tables and a small number of stages. Compared to two-level prediction, multi-stage cascaded prediction delivers superior prediction accuracy for any given total table entry budget we considered. In particular, a 512-entry three-stage cascaded predictor reaches 92% accuracy, reducing table size by a factor of four compared to a two-level predictor. At 1.5K entries, a three-stage predictor reaches 94% accuracy, the hit rate of a hypothetical two-level predictor with an unlimited, fully associative predictor table. At 6K entries, accuracy increases to 95%, the limit achieved by an idealized twelve-stage cascaded predictor with an unlimited hardware budget. These results indicate that highly accurate indirect branch target prediction is now well within the capability of current hardware technology.

1. Introduction

Prediction of branch targets and load values side-steps control and data-flow dependencies, enabling speculative execution of instructions and increasing instruction level parallelism [HP95]. The importance of accurate prediction increases as the processor-memory gap grows, processor pipelines become deeper, and superscalar issue increases. Processor technology has followed these trends in the past and probably will do so in the foreseeable future [P+97].

Currently, highly accurate conditional branch prediction is achieved by variations of the two-level predictor architecture proposed by Yeh and Patt [YP91]. Two-level prediction increases prediction accuracy by correlating a history of taken/non-taken bits of recently executed branches with the direction of the current branch. Lipasti et. al. successfully applied two-level prediction to load value prediction [LWS96].

Two-level predictors also prove highly effective for the prediction of indirect branch targets [CHP97]. Indirect branches, which transfer control to an address (recently) loaded into a register, are hard 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. Furthermore, their behavior is often directly determined by data loaded from memory, such as in virtual function calls in C++ and Java. Since the popularity of these languages continues to grow, we expect that processors will execute indirect branches more frequently in the future. Even today, indirect branch misses can cause significant overhead. Without two-level prediction (using a simple branch target buffer or BTB), the overhead of virtual function calls in C++ programs is as high as 29% [DH96]. Similarly, Chang, Hao, and Patt show that for the SPECint95 programs *perl* and *gcc* the indirect branch overhead is approximately 15% and 8% [CHP97].

In this study we evaluate predictor architectures for indirect branches. (However, we believe that our conclusions will also apply to conditional branch prediction and value prediction, for reasons discussed in section 6.) The accuracy of two-level predictors depends on the size of the predictor table that stores associations between history patterns and predicted targets. Longer histories lead to higher prediction accuracy but also increase the number of different history patterns. This effect causes capacity misses, which deteriorate prediction accuracy even for large tables. In a recent study [DH98b], we reduced the required size of the two-level predictor by placing a small BTB in front of it. Many branches are perfectly predicted by this cheap first stage, so that their associated history patterns can be filtered out; only history patterns of branches that are hard to predict enter the second stage.

Here we investigate the accuracy of a natural generalization of this two-stage cascaded predictor by allowing any type of predictor in the first stage and any number of stages. First, we use the maximum number of stages, and unlimited, fully associative tables for each stage, to determine the limit of prediction accuracy reachable by this architecture. Secondly, we test two and three stage predictors for a wide range of table sizes, in order to study cost reduction for practical predictors.

This paper makes the following contributions:

- It demonstrates, for the first time, that idealized indirect branch predictors can exceed 95% prediction accuracy.
- It describes and evaluates a practical (4K) indirect branch predictor that achieves nearly 95% accuracy on average for our set of large C and C++ applications.
- It explains why cascaded predictors work so well, and quantifies the dramatic reduction in predictor table working set size achieved by cascading. This analysis suggests that conditional branch predictors or load value predictors could benefit from cascaded prediction as well.

The rest of this paper is organized as follows: in Section 2 we discuss the benchmark suite used, and Section 3 briefly reviews two-level and cascaded predictor architectures. Section 4 compares the accuracy of two-level and ideal cascaded predictors for various numbers of stages/path lengths, and Section 5 presents results for realistic predictors. Section 6 discusses related work, and we conclude in Section 7.

2. Benchmarks

We minimize misprediction rate using a reduced instruction trace consisting of indirect branch addresses and targets, and simulate only the indirect branch predictor. This allows us to explore two to three orders of magnitude more predictor configurations than a full cycle-level simulation would allow. Reductions in misprediction rate should lead to corresponding reductions in branch misprediction overhead (as demonstrated in [CHP97]).

Name	Description	Style	K lines of code	K # of indirect branches	instr. / indirect	cond. / indirect	virtual%	switch%	indirect%	1 target%	2 targets%	> 2 targets%	active branches	
													99%	100%
idl	IDL compiler[a]	OO	14	1,884	47	6	93.2	3.2	3.6	97.1	0.1	2.8	70	543
jhm	JHM[b] 6-12M	OO	15	6,000	47	5	93.6	1.2	5.2	58.7	1.4	39.9	34	155
self	Self-93 VM: 5-6M	OO	77	1,000	56	7	76.0	4.4	19.6	40.1	31.6	28.3	848	1855
xlisp	SPEC95	C	5	6,000	69	11	0.0	0.1	99.9	38.9	9.0	52.1	4	13
troff	GNU groff 1.09	OO	19	1,111	90	13	73.7	12.5	13.8	41.9	13.6	44.5	61	161
lcom	HDL[c] compiler	OO	14	1,738	97	10	63.2	36.8	0.0	33.5	54.0	12.5	87	328
AVG-100: instr/indirect < 100			24	2,955	68	9	66.6	9.7	23.7	51.7	18.3	30.0	184	509
perl	SPEC95	C	21	300	113	17	0.0	31.7	68.3	41.2	0.0	58.8	7	24
porky	scalar optimizer[d]	OO	23	5,393	138	19	70.6	23.8	5.6	15.6	8.1	76.3	89	285
ixx	IDL parser[e]	OO	11	212	139	18	46.5	52.2	1.3	37.1	6.4	56.5	91	203
edg	C++ front end	C	114	549	149	23	0.0	62.4	37.6	7.9	29.6	62.5	186	350
eqn	equation typesetter	OO	8	296	159	25	33.8	66.2	0.0	4.2	37.8	58.0	58	114
gcc	SPEC95	C	131	865	176	31	0.0	31.5	68.5	0.8	1.7	97.5	95	166
beta	BETA compiler	OO	73	1,006	188	23	0.0	2.3	97.7	18.7	28.1	53.2	135	376
AVG-200: 100 < instr/indirect < 200			55	1,232	152	22	21.6	38.6	39.9	17.9	16.0	66.1	94	217
AVG: instr/indirect < 200			40	2,027	113	16	42.4	25.3	32.4	33.5	17.0	49.5	136	352
AVG-OO: OO, instr/indirect < 200			28	2,071	107	14	61.2	22.5	16.3	38.5	20.1	41.3	164	447
AVG-C: C, instr/indirect < 200			68	1,928	127	21	0.0	31.4	68.6	22.2	10.1	67.7	73	138
m88ksim	SPEC95	C	12	300	1.8K	233	0.0	46.2	53.8	2.9	10.3	86.8	5	17
vortex	SPEC95	C	45	3,000	3.5K	525	0.0	30.7	69.3	23.1	16.9	60.0	10	37
ijpeg	SPEC95	C	17	33	5.8K	441	0.0	97.8	2.2	96.7	3.2	0.1	7	60
go	SPEC95	C	29	550	56K	7123	0.0	99.0	1.0	0.2	0.0	99.8	5	14
AVG-infreq: instr/indirect > 200			26	971	17K	2081	0.0	68.4	31.6	30.7	7.6	61.7	7	32

Table 1. Benchmarks and commonly shown averages (arithmetic means)

[a] SunSoft version 1.3
[b] Java High-level Class Modifier
[c] hardware description language compiler
[d] SUIF 1.0
[e] Fresco X11R6 library

Indirect branches occur frequently in some programs of widely used benchmark sets like the SPECint95 suite, although they remain less common than conditional branches. However, indirect branches are much more frequent in object-oriented languages. These languages promote a programming style in which late binding of subroutine invocations is the main instrument for clean, modular code design. Virtual function tables, the implementation of choice for most C++ and Java compilers, execute an indirect branch for every lately bound call. The C++ programs studied here execute an indirect branch as frequently as once every 50 instructions; other studies [CGB94] have shown similar results. Java programs (where all non-static calls are virtual) are likely to use indirect calls even more frequently.

Our main benchmark suite consists of large object-oriented C++ applications ranging 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 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 -msupersparc 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

[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.

predicted accurately with a return address stack [KK98]. All programs were run to completion or until six million indirect branches were executed.[1] In *jhm* and *self* we excluded the initialization phases by skipping the first 5 and 6 million indirect branches, respectively.

For each benchmark, Table 1 lists 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 source of the indirect branches (switch statements, virtual function calls, or indirect function calls). It also shows the percentage of indirect branches that during the entire run jump to one, two and more targets, as well as the number of branch sites responsible for 99%, and 100% of the branch executions. For example, only 5 different branch sites are responsible for 99% of the dynamic indirect branches in *go*. The SPECint95 programs are dominated by very few indirect branches, with less than ten interesting branches for all programs except *gcc*. 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 when optimizing predictor accuracy (by minimizing the AVG misprediction rate).

3. Predictor architectures

Figure 1 shows representative examples of the predictor architectures tested in this study. The simplest architecture is a *branch target buffer* (BTB). A selection of bits from the branch address serves as a key pattern into a predictor table, which stores the last target observed for this branch. We use tagged tables to distinguish table misses (pattern is absent) from prediction misses (pattern is there, but the stored target is wrong). For the unlimited, fully associative tables in Section 4, the tag consists of the complete key pattern. For the 4-way associative, limited tables employed in Section 5, part of the 24-bit pattern is used as a table index, and the rest is stored as a tag, indicating which of the 4 entries in the associativity set has a prediction for the pattern, if any. A predictor table closely resembles a 4-way associative cache [HP95].

A *two-level* predictor extends the BTB scheme by taking bits from the last p branch targets preceding the execution of the current branch and xor-ing these bits with the branch address. The parameter p is the *path length* of the two-level predictor. For longer path lengths, fewer bits are extracted from each target in order to fit into the 24-bit key pattern[2].

A *cascaded* predictor consists of several stages, each containing a two-level predictor with its own history buffer and predictor table. Successive stages use increasing path lengths (in [Dri99], we demonstrate the inferior accuracy of decreasing path lengths). The use of separate tables allows all stages to predict in parallel. In a final step, the predictor chooses the prediction from the last stage that did not encounter a table miss. This ensures that its target prediction is based on the longest available path history.

A cascaded predictor saves table space by using a *leaky filter* update rule[3]: a new history pattern enters a long path length stage only if none of the shorter stages predicted the branch correctly. This rule prevents easily predicted branches from occupying table space in an expensive, long path length stage. For example, branches with only a single target are perfectly predicted by a BTB, after the initial compulsory miss, so they do not need a long history pattern[4] (this is a substantial portion of all indirect branches, as shown in Table 1). As a result, the longer path length stage encounters fewer capacity misses, improving overall prediction accuracy.

[1] 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.

[2] In a previous studiy [DH98a], we showed that 24 bits suffice for negligible accuracy loss due to pattern interference for BTB and two-level predictors. These studies also investigate the effect of alternatives to predictor details chosen here. For the record, we use a two-bit counter update rule, reverse interleaving of target bits and 4-way associative tables with an LRU eviction policy.

[3] In [DH98b], we demonstrate the inferior accuracy of an alternative, strict filtering update rule.

[4] Since there is usually more than one path leading up to a branch, longer histories generate more patterns, and thus occupy more table entries, even for a branch with only a single target.

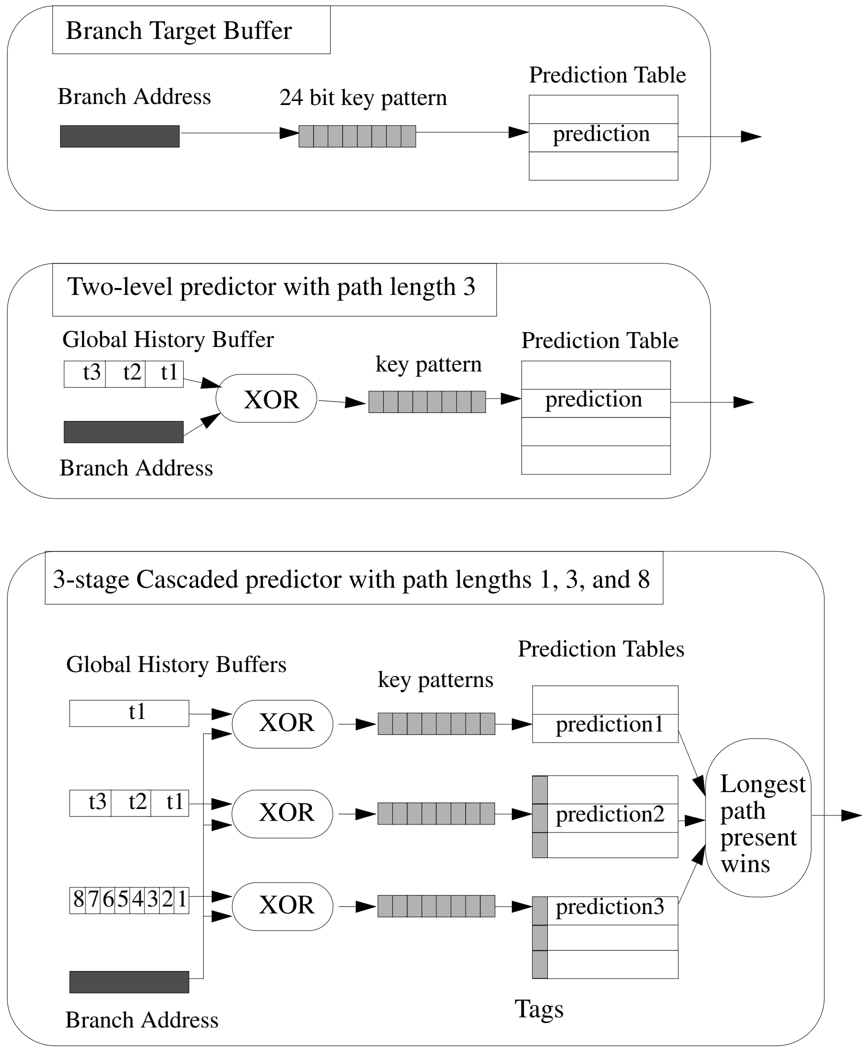

Figure 1. Representative examples of a branch target buffer, two-level predictor and cascaded predictor. A staged predictor looks the same as a cascaded predictor, but has a different update rule (every stage is updated, where a cascaded predictor prevents insertion of new patterns in later stages if an earlier stage predicts a branch correctly).

We also measure the accuracy of a cascaded predictor without filtering. We call this a *staged* predictor. Staged predictors, even without filtering, improve prediction accuracy compared to a two-level predictor because they reduce cold start misses. Longer path length two-level predictors are more accurate than short path length predictors, but they need a longer time to reach that potential since they store more patterns per branch. In a staged predictor, the early stages predict many branches accurately while the later stages are warming up. This also reduces the *effect* of capacity misses in later stages, since an earlier stage is likely to have a (less accurate) prediction available in the case of a late stage capacity miss.

In the next section we investigate predictor accuracy under ideal circumstances, in the absence of table interference (conflict misses) and capacity misses.

4. Ideal predictors

We use the term *ideal* for a predictor scheme with an unlimited, fully associative predictor table. The misprediction rate measured for such a predictor is free from the noise of conflict and capacity misses. This allows us to compare the intrinsic strength of different prediction schemes, and provides us with a limit beyond which prediction cannot be improved merely by spending more transistors on predictor tables.

In one respect, the predictors studied in this section are not ideal: they have limited 24-bit history buffers. History buffers must be kept small because they determine the cost of a table entry through the size of the tag. A small buffer represents each target with few bits, and this can cause pattern interference, reducing prediction accuracy. However, a 24-bit history buffer suffices for near-ideal accuracy. Beyond path length 12, a 24-bit history buffer causes extensive pattern interference because it stores only one bit for some targets. Therefore we do not use path lengths longer than 12. This restriction does not substantially reduce the potential accuracy, as we found during a preliminary experiment with a 30-bit buffer and path lengths up to 15.

Ideal cascaded predictors have a full complement of stages. For example, an ideal cascaded predictor of path length 6 has 7 stages, consisting of ideal two-level predictors with path lengths 0 (a BTB) up to 6. Similarly, and ideal *staged* predictor also has the maximum number of stages, but does not employ pattern filtering.

Terminology	Description
Unlimited ideal two-level of path length P	Two-level predictor with an unlimited history buffer, storing a concatenation of full precision addresses of the P most recent targets, with an unlimited, fully associative predictor table
Ideal two-level of path length P	Two-level predictor with 24-bit history buffer, storing the xor of (24 div P) bits of the P most recent targets, with an unlimited, fully associative predictor table
Ideal branch target buffer (BTB)	Ideal two-level predictor of path length 0
Ideal staged of path length P	A non-filtering staged predictor, with P+1 stages, consisting of ideal two-level predictors of pathlength 0,1,..,P
Ideal cascaded of path length P	An ideal staged predictor of pathlength P, with filtering of new patterns

Table 2. Ideal predictor terminology

Figure 2 shows misprediction rates for ideal two-level predictors, cascaded predictors and staged

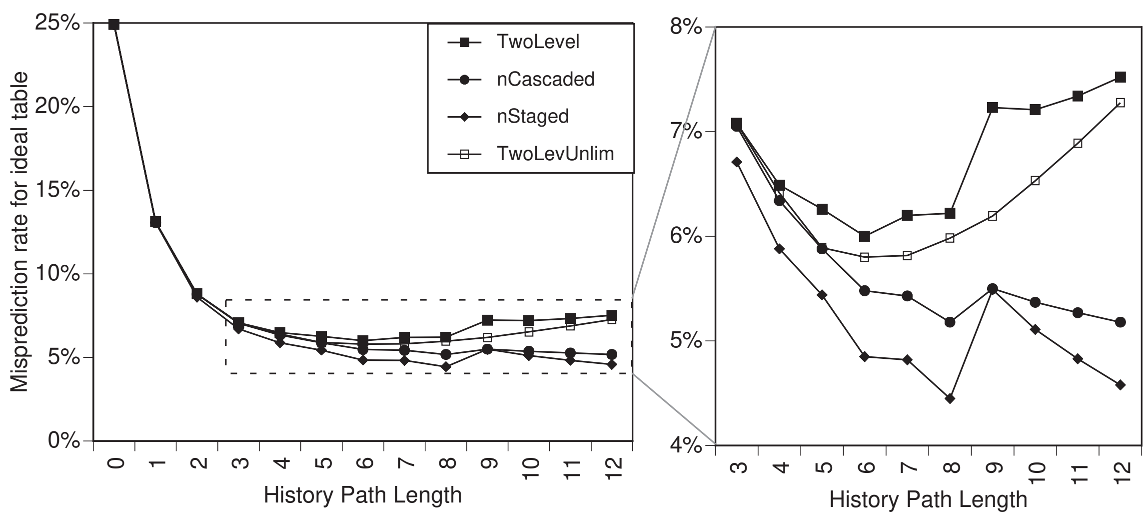

Figure 2. Misprediction rates for ideal two-level, fully cascaded and fully staged predictors, for path lengths 0 to 12 (cutout enlarged in right graph)

predictors for pathlengths from 0 to 12. Two-level prediction reaches a minimum misprediction rate of 6.0% at path length 6. Longer path lengths show increasing misprediction rates, because cold start misses start to negate the advantage of capturing longer-term correlations. The loss of accuracy due to pattern interference is small, as shown by the accuracy of an ideal two-level predictor with unlimited buffer size (from [DH98a]). Where path lengths are integer factors of the buffer size, enabling full buffer use, interference is negligible (path lengths 0 to 4, 6, 8 and 12). Only for path lengths that use less than the full history buffer does the misprediction rate increase noticeably. For example, path length 9 in a 24 -bit history buffer uses only two bits per target, for a total of 18 bits[1]. The difference in accuracy, represented by the "bump" in the curves, is strictly due to pattern interference. The curves for cascaded prediction and staged prediction show similar bumps, the most prominent at path length 9.

An ideal cascaded predictor is better than ideal two-level prediction at all path lengths. The short path length stages in a cascaded (and a staged) predictor immediately start predicting many branches accurately while the later stages are warming up, as explained in Section 3. This gives it an advantage over a two-level predictor of the same path length even in the absence of capacity misses.

Staged prediction reaches lower misprediction rates than cascaded prediction at all path lengths. This is to be expected, since a cascaded predictor uses filtering and therefore stores strictly less information than a staged predictor. Cascaded prediction merely economizes on the number of table entries required by staged prediction, and this has no beneficial effect for ideal predictors with unlimited tables.There simply are no capacity misses to reduce.

However, the reduction of table entry cost is dramatic, as shown in Figure 3. The graph shows the total number of table entries occupied in a two-level, cascaded and staged predictor. As the path lengths grows, a staged predictor's size grows exponentially, while a cascaded and two-level predictor show nearly linear growth. At path length 12, a staged predictor occupies 173325 entries, five times as much as a cascaded predictor (34572) and a two-level predictor (34076). The curves of cascaded and two-level prediction overlap almost completely, suggesting some systematic effect.We currently can not explain this similarity, but suspect that the exact reason for the close resemblance is to be found in information theory. In terms of branch prediction schemes, it means that a two-level predictor table cost is in the same ball park as a fully cascaded predictor. However, the latter eliminates cold start misses, while preserving the capacity to capture longer term correlations. In the next section we measure these benefits in the context of practical predictors.

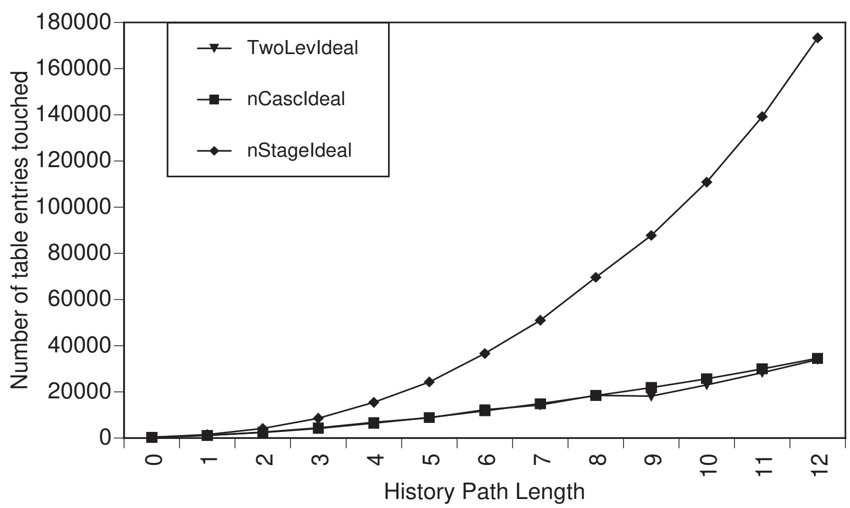

Figure 3. Total number of patterns stored by an ideal two-level, cascaded and staged predictor, for path lengths 0 to 12

5. Practical predictors

In this section we study practical cascaded predictor architectures with limited, 4-way associative tables and a small number of stages. Our aim is to reduce the number of table entries required to attain a given prediction accuracy. We also want to find out how close we can get to the prediction accuracy of the hypothetical ideal predictors of the previous section.

---

1 It is of course possible to fill the unused bit space with bits from a strict subset of the target addresses. This opens up a variety of choices as to which targets should be represented with 1 bit extra accuracy. In the reverse interleaved address projection employed [DH98a], this choice is unclear. To simplify our analysis, we left these bits unused.

5.1 Practical predictor tables

The main cost of predictor architectures lies in the amount of on-chip memory required to store predictions. In the previous section we saw that the total number of patterns generated by an ideal predictor grows as its path length increases. For example, a two-level predictor reaches a minimal misprediction rate of 5.8% at path length 6, by storing13210 pattern/target associations (averaged over the AVG benchmarks). Given a table entry size of about 60 bits (24-bit tag, 1-bit update counter, and 32-bit target address), the resulting data structure takes up about 800 K bits of memory, straining the capability of current processor technology. We want to reduce this memory cost while keeping misprediction rates as low as possible.

Reducing the size of a predictor table generates *capacity* misses: a pattern/target association, though it was stored, and would have correctly predicted the target of a branch, was evicted from the table by a pattern/target of a more recently executed branch. For smaller tables, the path length must be shortened to prevent extensive capacity misses. However, shorter path length predictors are less accurate. For every given table size, there is some path length which forms the optimal compromise between these opposing effects. We use simulation to determine this optimal path length, and the resulting misprediction rate, for table sizes from 32 to 32K entries.

One further limitation is necessary to allow limited size predictor tables to be implemented in silicon: a limited associativity (see Section 3). We use tables with associativity four, a common choice for memory caches .

5.2 Practical multi-stage predictors

A staged predictor must split up a given total table entry budget and allocate some part of it to each stage. Although it is conceivable to use one global predictor table for all stages, this would require an expensive multi-access table. Therefore each stage uses a separate single-access table. For a given total table entry budget, we have the option of using many stages with small tables, or few stages with large tables. Which one is better?

For a cascaded predictor, there are two opposite effects at work. On the one hand, partitioning table space over a large number of stages increases the likelihood that a benchmark’s working set overloads the capacity of any given stage, resulting in extensive capacity misses. This also reduces the efficiency of pattern filtering, since only correct predictions prevent patterns from occupying space in later stages. On the other hand, a large number of stages allows filtering to occur at a stage with a shorter path length, reducing the total number of patterns stored. For our benchmark suite, a small number of stages uses a given table entry budget more effectively than a full complement of stages. For example, a cascaded predictor with 9 stages, from path length 0 up to 8, with 128 entries per stage (total budget of 1152 entries) achieves a misprediction rate of 7.7%, while a 2-stage cascaded predictor with 512 entries per stage (total budget of 1024), and path lengths 1 and 8, reaches 6.8%.

We therefore focus on predictors with a small number of stages. From an analysis point of view, their superior accuracy is a windfall, since a small number of stages substantially reduces the number of different path length combinations to be simulated[1]. For 2-stage predictors without filtering (staged), we test all path length combinations P1P2, with 0 <= P1 < P2 <= 12, for each total table entry budget that is a power of 2 between 32 and 32K entries. Each stage uses half of the total table size.

We also determine optimal path length combinations for cascaded predictors. For the 2-stage cascaded predictor, we test the same range of path lengths as for the 2-stage non-filtering predictor. However, we also simulate table entry budgets halfway between powers of 2 (in order to compare with 3-stage predictors with same-size stages). In that case, two third of the entries is allocated to the last stage.

[1] An truly exhaustive study would require simulation of approximately #stages ^ (#tablesizes * #historylengths). For the range of table sizes, history lengths and number of stages explored in this study, this would lead to 15 ^ (10 * 15) ~= 2.6E176 different combinations, far too many to explore within the given time frame.

For the 3-stage predictor we test all pathlength combinations with 0 <= P1 < P2 < P3 <= 12, with P1 <= 4, and P2 <= 10, to reduce the number of combinations without removing good candidates. We simulate the same range of total table entry budgets as for the 2-stage cascaded predictor by using stage sizes X+X+X and X+X+2X, with X equal to a power of 2. Table 3 summarizes the terminology.

Terminology	Description
Two-level of size T	Two-level predictor with a 4-way associative predictor table of size T
2-staged of size T	A non-filtering staged predictor, using 2 two-level predictors of size T/2
2-staged cascaded of size T	A 2-staged predictor of size T, with pattern filtering. If T is not a power of 2, the stages have size T/3 and 2T/3
3-staged cascaded of size T	A 3-staged predictor with pattern filtering, using 3 two-level predictors of size T/3. If T is a power of 2, the stages have size T/4,T/4, and T/2

Table 3. Practical predictor configurations (path length combinations are tuned to each size T)

5.3 Results

Figure 4 shows misprediction rates that result from using the best path length combinations for each

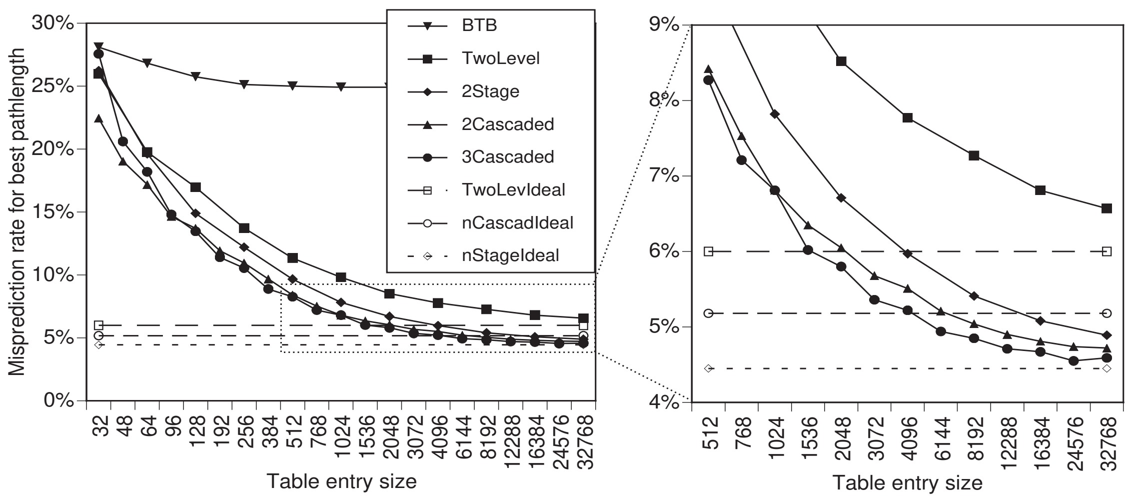

Figure 4. Misprediction rates for two-level, 2-stage, 2-stage cascaded, and 3-stage cascaded predictors

predictor configuration. The path length combinations themselves are listed in the appendix. For comparison we also show the misprediction rate of a BTB and the best ideal two-level, cascaded and staged predictors of the previous section (shown as dotted lines parallel to the x-axis since table size is not a factor).

Cascaded predictors perform better than two-level predictors at all table sizes in the explored range. In other words, any given misprediction rate is bought at a much lower cost. For instance, a 3-stage predictor of size 512 outperforms a two-level predictor of size 2K. At a fairly modest budget of 1.5K entries, a 3-staged cascaded predictor attains the same misprediction rate as an ideal two-level predictor with an unlimited table.

3-stage cascaded predictors consistently outperform 2-stage predictors, and 2-stage cascaded predictors outperform 2-stage (non-filtering) predictors. For limited budgets, cascaded prediction wins over staged prediction because pattern filtering reduces capacity misses, allowing a cascaded predictor to use longer path lengths for any given table size.

2-and 3-staged predictors seem to get asymptotically close to the accuracy of an ideal staged predictor for large, but still practical table budgets. A small number of stages clearly suffices to get almost arbitrarily close to ideal accuracy. A 3-stage cascaded predictor, at 4K table entries and higher, even improves upon the accuracy of an ideal cascaded predictor with a full complement of stages. This surprising result indicates that a small number of stages improves prediction accuracy. The label "ideal" for a fully staged cascaded predictor thus seems to be a misnomer. We are not entirely sure why this is the case. Filtering seems to be responsible for the reduced accuracy, since a non-filtering ideal staged predictor still outperforms all other predictors. A small number of stages does not filter out as many patterns, and therefore it may represent an intermediate between these extremes, at large table budgets[1], resulting in an intermediate misprediction rate.

Do 3-stage cascaded predictors form the best possible configuration for a limited budget ? We think not. Dividing the table entry budget over 8 stages gives worse performance than a 2-stage cascaded predictor, as shown above, so the optimal number of stages may lie somewhere in between 3 and 8, most likely at the lower end. However, the benefit of extra stages is not likely to improve performance by a large factor. The difference between 2 and 3 stage cascaded prediction is already fairly small, allowing a 3-stage predictor of size 3X to reach the same or better performance than a 2-stage predictor of size 4X, as soon as X is larger than 512. The added complexity of extra stages may not be worth the effort.

We did not show staged predictors in which stages differ in size by a large degree, although we did simulate 2-and 3stage predictors with the later stage(s) as much as a factor 32 larger than the earlier one(s). The results indicate that predictors with a table budget spread out evenly among (a small number of) stages perform best. This effect is already noticable for the 3-stage predictors shown here, since the curve in Figure 4 shows slight bumps for those configurations with total table budgets that are powers of two. In other words, a 3-stage predictor with stages of size X+X+X performs relatively better, in accuracy per table entry, than a 3-stage predictor with stage sizes X+X+2X.

6. Related work

Indirect branch prediction has been studied by Lee and Smith [LS84] (several forms of BTBs), Jacobson et al. [J+96] (path-based history schemes), Emer and Gloy [EM98] (single-level indirect branch predictors), and Chang et al. [CHP97] (two-level indirect branch prediction). In [CHP97], the resulting speedup of selected SPECint95 programs is measured by simulation for a superscalar processor. The misprediction rate of a BTB is reduced by half to 30.9% for *gcc* with a Pattern History Tagless Target Cache with configuration gshare(9), resulting in 14% speedup.

Kalamatianos and Kaeli [KK98] apply partial prefix matching (PPM) prediction to indirect branches, demonstrating excellent accuracy. A PPM predictor shortens a history pattern bit by bit, and looks it up in successively smaller stages. Each stage is half the size of its predecessor. The bits correspond to branch targets, so this scheme tests ever shorter path lengths. This resembles the prediction rule of a cascaded predictor. Cascaded prediction differs from PPM prediction because a cascaded predictor employs pattern filtering and uses a separate history buffer for each stage.The number of stages is also independent from pattern length, and each stage can use any table size. Although they demonstrate slightly better prediction performance on some of the benchmarks in this study, at least part of the improvement is due to dynamic classification of indirect branches into two classes: a class that correlates best with a history buffer which stores both conditional and indirect branch targets, and one that correlates best with only indirect branch targets. In this study we use a purely indirect branch target trace.

[EM98] proposed the YAGS architecture for conditional branch prediction. A YAGS predictor uses two kinds of predictor tables. A direct-mapped table (the *choice* table) stores the dominant direction of a branch using a 2-bit counter. Two tagged tables (*direction* tables) store a prediction for a pattern that represents history as taken/non taken bits. One of these is used for branches that are mostly taken, the other for branches that mostly not taken. Prediction in a direction table takes precedence over the prediction in the choice table, and patterns enter a direction table only if the choice table mispredicts. This scheme resembles a 2-stage cascaded predictor with a direct-mapped BTB as first stage. The main difference is that the second stage of a YAGS predictor consists of two separate

[1] Positive interference could also be responsible. However, the tables are tagged, so this can only be pattern interference.

tables. However, the authors agreed that this is not a requirement. A YAGS predictor shows better prediction accuracy than other conditional branch predictor schemes. We believe this is evidence that cascaded prediction is also likely to perform well on conditional branches.

7. Conclusions

We have studied the accuracy of a new hybrid predictor architecture, the multi-stage cascaded predictor, on a trace of purely indirect branches. Cascaded prediction delivers superior accuracy even in the absence of resource constraints, by exceeding the accuracy reached by any other predictor scheme previously tested on these traces. In the context of limited transistor budgets, cascaded prediction also provides superior accuracy, this time by reducing the cost of two-level prediction by a factor of four or more.

More specifically:

- Ideal cascaded prediction with unlimited, fully associative tables reaches a hit rate of 94.8%. Ideal staged prediction, without pattern filtering, reaches 95.5%. We believe this accuracy is close to the limit of predictability, using a pure indirect branch history, of the indirect branches in our benchmark suite.
- Cascaded predictors with a small number of stages closely approach this limit when using large but practical table entry budgets. In particular, a 4K entry, 3-stage cascaded predictor attains 94.8% accuracy. Surprisingly, a cascaded predictor with a small number of stages delivers better accuracy than a fully staged predictor: a 32K entry 3-staged predictor reaches 95.4% accuracy, better than an ideal (fully staged) cascaded predictor.
- At every table entry budget from 32 to 32K entries, multi-staged cascaded prediction delivers accuracy superior to two-level prediction. In particular, a 512-entry three-stage cascaded predictor reaches 92% accuracy, reducing table size by a factor of four compared to a two-level predictor. With only 1.5K entries, a 3-stage predictor reaches 94% accuracy, the maximum hit rate achievable by a hypothetical two-level predictor with an unlimited, fully associative predictor table.

We believe that cascaded prediction can also improve conditional branch prediction and load value prediction, because these applications suffer equally from cold start and capacity misses, and because recent related work [EM98] shows that a similar architecture delivers superior accuracy on conditional branches. It seems to be an idea whose time has come.

8. References

[CGB94] Brad Calder, Dirk Grunwald, and Benjamin Zorn. *Quantifying Behavioral Differences Between C and C++ Programs*. Journal of Programming Languages, pages 313-351, Vol 2, Num 4, 1994.

[CHP97] Po-Yung Chang, Eric Hao, Yale N. Patt. Target Prediction for Indirect Jumps. *ISCA'97 Proceedings,* July 1997.

[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.

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

[DH98a] Karel Driesen and Urs Hölzle. Accurate Indirect Branch Prediction. *ISCA '98 Conference Proceedings*, pp. 167-178, Barcelona, July 1998.

[DH98b] Karel Driesen and Urs Hölzle. The Cascaded Predictor: Economical and Adaptive Branch Target Prediction. *Micro'98 Conference Proceedings*, Dallas, Texas, December 1998.

[Dri99] Karel Driesen. *Software and Hardware Techniques for Efficient Polymorphic Calls.* PhD dissertation, University of California, Santa Barbara (in preparation).

[EM98] A.N.Eden and T.Mudge. The YAGS Branch Prediction Scheme. *Micro'98 Conference Proceedings*, Dallas, Texas, December 1998.

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

[HP95] Hennessy and Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1995.

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

[KK98] John Kalamatianos and David Kaeli. Predicting Indirect Branches via Data Compression. *Micro'98 Conference Proceedings*, Dallas, Texas, December 1998.

[LS95] James Larus and Eric Schnarr. EEL: Machine-Independent Executable Editing. In *PLDI '95 Conference Proceedings*, pp. 291-300, La Jolla, CA, June 1995. Published as *SIGPLAN Notices 30(6)*, June 1995.

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

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

[LWS96] Mikko H.Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value Locality and Load Value Prediction. *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS VII), October 1996, pp. 138-147.

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

[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

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

Appendix :Detailed Data

Table 4 shows misprediction rates and occupied table entries for ideal predictors with unlimited tables, for various path lengths.

	Table entries				Misprediction rates				
Path length	BTB	Twolevel	Cascaded	Staged	BTB	Twolevel	Cascaded	Staged	Unlimited Twoleve
0	356	356	356	356	24.9	24.9	24.9	24.9	24.9
1	356	1195	1191	1551	24.9	13.1	13.1	13.1	13.1
2	356	2617	2495	4169	24.9	8.8	8.8	8.6	8.8
3	356	4477	4266	8645	24.9	7.1	7.0	6.7	7.1
4	356	6863	6469	15508	24.9	6.5	6.3	5.9	6.4
5	356	8838	8940	24346	24.9	6.3	5.9	5.4	5.9
6	356	12324	11838	36670	24.9	6.0	5.5	4.8	5.8
7	356	14379	14935	51048	24.9	6.2	5.4	4.8	5.8
8	356	18580	18430	69629	24.9	6.2	5.2	4.5	6.0
9	356	18160	21914	87788	24.9	7.2	5.5	5.5	6.2
10	356	23061	25770	110849	24.9	7.2	5.4	5.1	6.5
11	356	28400	30003	139249	24.9	7.3	5.3	4.8	6.9
12	356	34076	34572	173325	24.9	7.5	5.2	4.6	7.3

Table 4. Total number of entries and misprediction rates for ideal BTB, two-level, fully cascaded and fully staged predictors, and for an ideal two-level predictor with unlimited history buffer (from [DH98a])

Table 5 shows misprediction rates of practical predictors and the best path length combination at various table budgets.

	Misprediction rate in%					Tuned path length combinations			
Total table size	BTB	Twolevel	2stage	2cascaded	3cascaded	Twolevel	2stage	2cascaded	3cascaded
32	28.1	26.0	26.2	22.4	27.6	1	0.2	0.2	0.1.2
48				19.0	20.6			0.2	0.2.3
64	26.8	19.8	19.6	17.2	18.2	1	0.2	0.2	0.1.3
96				14.7	14.8			0.2	0.2.5
128	25.8	17.0	14.9	13.7	13.5	1	0.2	0.2	0.1.3
192				11.9	11.4			0.3	0.2.8
256	25.1	13.7	12.2	11.0	10.6	2	0.2	1.6	0.1.5
384				9.7	8.9			0.3	0.2.8
512	25.0	11.3	9.7	8.4	8.3	2	1.5	1.8	0.2.8
768				7.5	7.2			1.6	0.2.8
1024	24.9	9.8	7.8	6.8	6.8	3	1.6	1.8	0.2.8
1536				6.4	6.0			1.6	1.4.12
2048	24.9	8.5	6.7	6.0	5.8	3	1.6	1.8	1.3.12
3072				5.7	5.4			1.8	1.4.12
4096	24.9	7.8	6.0	5.5	5.2	3	2.8	1.8	1.4.12
6144				5.2	4.9			1.8	1.4.12
8192	24.9	7.3	5.4	5.0	4.8	4	2.8	2.12	1.4.12
12288				4.9	4.7			2.12	1.6.12
16384	24.9	6.8	5.1	4.8	4.7	5	2.8	2.12	1.6.12
24576				4.7	4.6			2.12	3.4.12
32768	24.9	6.6	4.9	4.7	4.6	5	2.8	2.12	1.6.12

Table 5. Misprediction rates of a BTB, Twolevel, 2-stage, 2-stage cascaded and 3-stage cascaded predictor and their best path length combinations (a BTB's path length is 0), for various table entry budgets.

Table 6 shows misprediction rates per benchmark for selected predictor schemes. Headings represent predictor schemes. For example, *8Staged* is an 8-staged nonfiltering staged predictor, *3Casc.P0.2.8* is a 3-staged cascaded predictor with path lengths 0, 2, and 8 (the best combination for a total of 1K-entries, in three 4-way associative tables). *Oracle12* is a 12-stage nonfiltering staged predictor with a perfect metaprediction rule: instead of picking the component prediction with the longest path length, the *correct* prediction is chosen, if any component predicts the branch correctly. *3CascPerBench* is the misprediction rate reached by a 3-staged cascaded predictor (1K-entry in this case), which is tuned specifically to each benchmark (instead of choosing path lengths which minimize AVG misprediction rates).

Name			active branches		Ideal					1.5K 4-way assoc		1K-4way assoc				
	instr. / indirect	cond. / indirect	99 %	100 %	Oracle12	8Staged	8Cascaded	P6TwoLevelIdeal	BTB	3Casc.P1.4.12	2Casc.P1.6	3CascPerBench	3Casc.P0.2.8	2Casc.P1.8	TwoLevel.P3	BTB
idl	47	6	70	543	0.2	0.3	0.4	0.4	2.4	0.3	0.4	0.3	0.4	0.4	0.8	2.4
jhm	47	5	34	155	2.1	8.1	8.6	8.7	11.1	8.7	9.7	8.6	9.9	9.6	11.3	11.1
self	56	7	848	1855	3.1	7.4	8.0	10.1	15.7	12.2	12.9	12.2	12.4	13.9	18.6	16.3
xlisp	69	11	4	13	0.3	1.4	1.6	1.4	13.5	1.9	1.7	1.5	1.8	2.0	2.2	13.5
troff	90	13	61	161	3.9	6.7	6.8	7.1	13.7	7.7	7.3	7.2	7.9	7.8	7.8	13.7
lcom	97	10	87	328	0.4	0.8	0.9	1.4	4.2	1.2	1.2	1.3	1.3	1.4	2.2	4.2
AVG-100	68	9	184	509	1.7	4.1	4.4	4.8	10.1	5.3	5.5	5.2	5.6	5.9	7.2	10.2
perl	113	17	7	24	0.2	0.2	0.2	0.5	31.8	0.2	0.2	0.2	0.2	0.2	0.3	31.8
porky	138	19	89	285	1.6	4.2	4.3	4.5	20.8	4.6	4.7	5.0	6.2	5.0	7.9	20.8
ixx	139	18	91	203	1.5	3.5	4.0	5.6	45.7	3.3	4.5	3.3	4.2	4.0	9.0	45.7
edg	149	23	186	350	3.4	7.2	8.1	11.9	35.9	8.9	9.6	9.4	9.6	9.4	16.3	35.9
eqn	159	25	58	114	2.8	8.5	10.8	12.5	34.8	12.8	13.1	14.4	14.8	14.9	21.9	34.8
gcc	176	31	95	166	2.6	8.2	11.9	11.7	65.7	14.8	15.1	15.3	17.3	17.2	23.6	65.7
beta	188	23	135	376	0.6	1.3	1.5	2.3	28.6	1.7	2.1	2.6	2.6	2.6	5.6	28.6
AVG-200	152	22	94	217	1.8	4.7	5.8	7.0	37.6	6.6	7.1	7.2	7.8	7.6	12.1	37.6
AVG	113	16	136	352	1.7	4.5	5.2	6.0	24.9	6.0	6.4	6.3	6.8	6.8	9.8	25.0
AVG-OO	107	14	164	447	1.8	4.5	5.0	5.8	19.7	5.8	6.2	6.1	6.6	6.6	9.5	19.7
AVG-C	127	21	73	138	1.6	4.3	5.5	6.3	36.7	6.5	6.7	6.6	7.2	7.2	10.6	36.7
m88ksim	1827	233	5	17	1.0	5.7	6.7	3.1	76.4	3.0	3.0	2.0	7.7	5.8	14.4	76.4
vortex	3480	525	10	37	2.4	4.3	6.3	9.9	20.2	7.7	9.2	4.5	6.4	4.6	7.1	20.2
ijpeg	5770	441	7	60	0.2	0.2	0.2	0.6	1.3	0.3	0.3	0.2	0.2	0.3	0.5	1.3
go	56355	7123	5	14	4.7	23.2	23.8	22.8	29.2	21.7	22.0	20.6	21.9	23.0	21.6	29.2
AVG-infreq	16858	2081	7	32	2.1	8.4	9.3	9.1	31.8	8.2	8.6	6.8	9.1	8.4	10.9	31.8

Table 6. Missrates per benchmark for selected predictor schemes