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

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. 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 [CHP97].

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. 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]).

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, and *beta*, a compiler for the Beta programming language [MMN93], written in Beta. We also measured the

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

Name	Description	Style	K lines of code	K # of indirect branches	instr. / indirect	virtual%	switch%	indirect%	1 target%	2 targets%	> 2 targets%	active branches	
												99%	100%
idl	IDL compiler[a]	OO	14	1,884	47	93.2	3.2	3.6	97.1	0.1	2.8	70	543
jhm	JHM[b] 6-12M	OO	15	6,000	47	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	76.0	4.4	19.6	40.1	31.6	28.3	848	1855
xlisp	SPEC95	C	5	6,000	69	0.0	0.1	99.9	38.9	9.0	52.1	4	13
troff	GNU groff 1.09	OO	19	1,111	90	73.7	12.5	13.8	41.9	13.6	44.5	61	161
lcom	HDL[c] compiler	OO	14	1,738	97	63.2	36.8	0.0	33.5	54.0	12.5	87	328
AVG-100: instr/ind < 100			24	2,955	68	66.6	9.7	23.7	51.7	18.3	30.0	184	509
perl	SPEC95	C	21	300	113	0.0	31.7	68.3	41.2	0.0	58.8	7	24
porky	scalar optimizer[d]	OO	23	5,393	138	70.6	23.8	5.6	15.6	8.1	76.3	89	285
ixx	IDL parser[e]	OO	11	212	139	46.5	52.2	1.3	37.1	6.4	56.5	91	203
edg	C++ front end	C	114	549	149	0.0	62.4	37.6	7.9	29.6	62.5	186	350
eqn	equation typesetter	OO	8	296	159	33.8	66.2	0.0	4.2	37.8	58.0	58	114
gcc	SPEC95	C	131	865	176	0.0	31.5	68.5	0.8	1.7	97.5	95	166
beta	BETA compiler	OO	73	1,006	188	0.0	2.3	97.7	18.7	28.1	53.2	135	376
AVG-200: 100 < instr/ind < 200			55	1,232	152	21.6	38.6	39.9	17.9	16.0	66.1	94	217
AVG: instr/indirect < 200			40	2,027	113	42.4	25.3	32.4	33.5	17.0	49.5	136	352
AVG-OO: OO, instr/ind < 200			28	2,071	107	61.2	22.5	16.3	38.5	20.1	41.3	164	447
AVG-C: C, instr/ind < 200			68	1,928	127	0.0	31.4	68.6	22.2	10.1	67.7	73	138
m88ks	im SPEC95	C	12	300	1.8K	0.0	46.2	53.8	2.9	10.3	86.8	5	17
vortex	SPEC95	C	45	3,000	3.5K	0.0	30.7	69.3	23.1	16.9	60.0	10	37
ijpeg	SPEC95	C	17	33	5.8K	0.0	97.8	2.2	96.7	3.2	0.1	7	60
go	SPEC95	C	29	550	56K	0.0	99.0	1.0	0.2	0.0	99.8	5	14
AVG-infreq: instr/indirect > 200			26	971	17K	0.0	68.4	31.6	30.7	7.6	61.7	7	32

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

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[1].

For each benchmark, Table 1 lists the number of indirect branches executed, the number of instructions 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`. Four of the SPEC benchmarks execute more than 1,000 instructions per indirect branch. Since the impact of branch prediction will

[1] See technical report for compilation details [DH99].

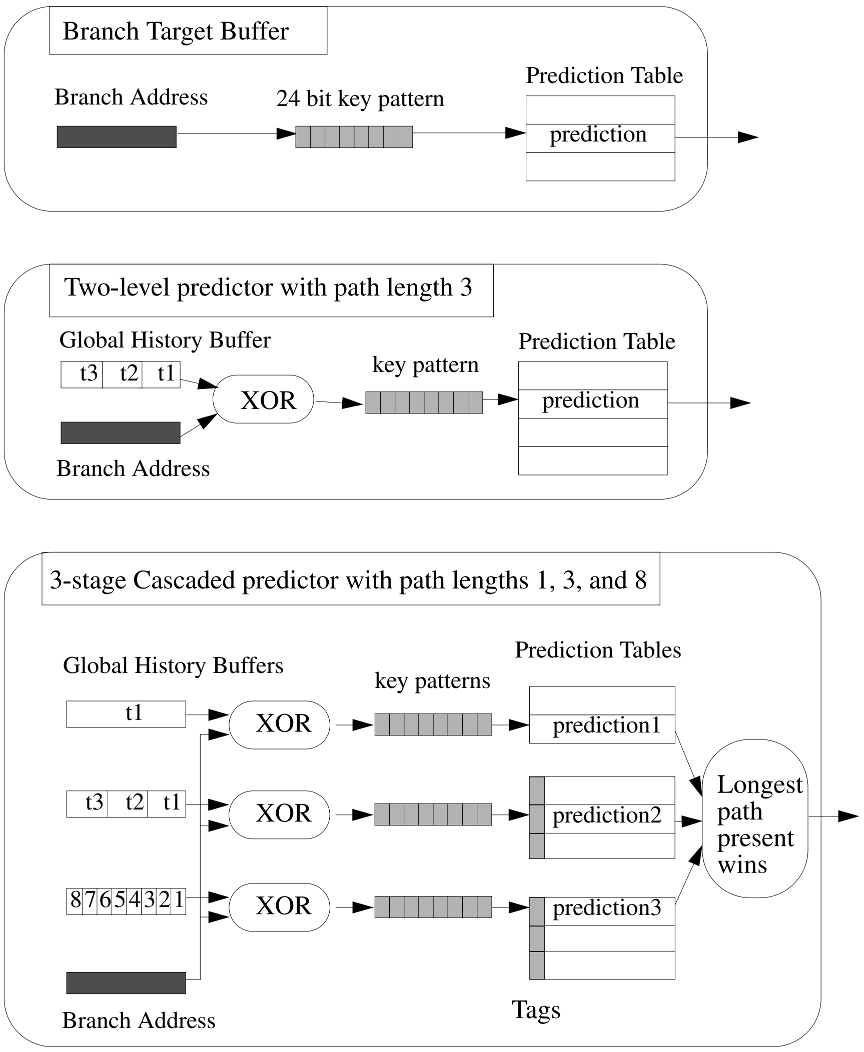

Figure 1. Representative examples of a branch target buffer, two-level predictor and cascaded predictor. A staged predictor looks the same as a cascaded predictor, but has a different update rule (every stage is updated, where a cascaded predictor prevents insertion of new patterns in later stages if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

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]. Note that only the second and later stages of the cascaded predictor need tags in order to function correctly. However, for easy comparison we use identical, tagged tables in all schemes.

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[1].

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: 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 (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.

We also measure the accuracy of a cascaded predictor without filtering. We call this a *staged* predictor. Staged predictor 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.

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 thus free from the noise of conflict and capacity misses[2]. Ideal cascaded predictors have a full complement of stages. For example, an ideal cascaded predictor of path length 6 has 7

[1] We use a two-bit counter update rule, reverse interleaving of target bits and 4-way associative tables with an LRU eviction policy, as in the two-level indirect branch predictor implementations in [DH98a].

[2] In one respect, the predictors studied in this section are not ideal: they have limited 24-bit history buffers. 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 (see [DH98a]), as we found during a preliminary experiment with a 30-bit buffer and path lengths up to 15 (also see the technical report for more details [DH99]).

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.Table 2 shows the ideal configurations, and the path length

Table 2. Ideal predictors

Terminology	Description	Best P	Miss%
Ideal two-level	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	6	6.0%
Ideal BTB	Ideal two-level predictor of path length 0	N/A	24.9%
Ideal staged	A non-filtering staged predictor, with P+1 stages, consisting of ideal two-level predictors of pathlength 0,1,..,P	8	4.5%
Ideal cascaded	An ideal staged predictor of pathlength P, with filtering of new patterns	8	5.2%

that minimizes the misprediction rate over the benchmark set (AVG). 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. As a result, an ideal cascaded predictor is better than ideal two-level prediction at all path lengths. 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 economizes on the number of table entries required, which does not increase accuracy for unlimited tables.

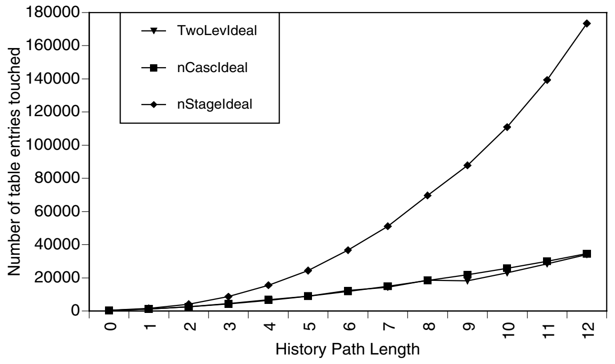

Figure 2. Total number of patterns stored by an ideal two-level, cascaded and staged predictor, for path lengths 0 to 12

However, the reduction of table entry cost is dramatic, as shown in Figure 2. The graph shows the total number of table entries occupied in a two-level, cascaded and staged predictor. As the path length grows, a staged predictor's size grows exponentially, while a cascaded and two-level predictor show nearly linear growth. In the next section we measure these benefits in the context of practical predictors.

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.

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 6.0% at path length 6, by storing 12324 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 capacity miss occurse when a pattern/target association, stored previously, 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 for practical predictors: 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. Given a limited number of stages, we cannot use each path length between 0 and 12. Instead, we need to choose the two or three path lengths that minimize misprediction rate for 2- and 3-stage predictors. We detemined the best path length combinations for all configurations shown in Table 3. We focus on predictors with a small number of large stages, since they outperform predictors with a large number of small stages for any given total number of entries (see the technical report [DH99] for path lengths and alternative configurations).

5.3 Results

Figure 3 shows misprediction rates that result from using the best path length combinations for each predictor configuration[1]. For comparison we also show the misprediction rate of a BTB and the best ideal two-level, cascaded and staged predictors

[1] [DH99] contains the list of optimal path length combinations per table size, the precise data for all graphs shown in this paper, and per-benchmark misprediction rates for selected predictor schemes.

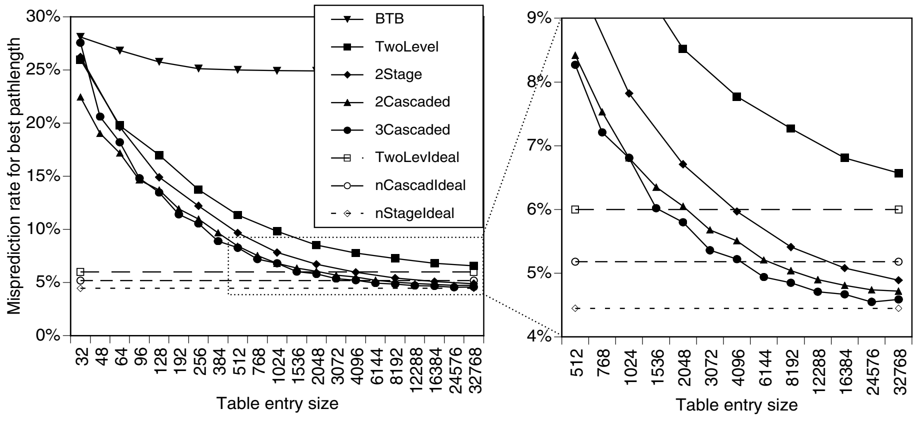

Figure 3. Misprediction rates for 2-stage, 2-stage cascaded, and 3-stage cascaded predictors

Table 3. Practical predictor configurations (with tuned path length combinations)

Terminology	Description
Two-level	Two-level predictor with a 4-way associative predictor table of size T
2-staged	A non-filtering staged predictor, using 2 two-level predictors of size T/2
2-staged cascaded	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	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

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 12K table entries and higher, even improves upon the accuracy of an ideal cascaded predictor with a full complement of stages. Filtering seems to be responsible for the reduced accuracy, since a non-filtering ideal staged predictor still outperforms all other predictors.

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 [EG97] (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 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 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.
- 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

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

[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.

[DH99] Karel Driesen and Urs Hölzle. *Multi-stage Cascaded Prediction.* Technical Report TRCS99-05, Department of Computer Science, University of California, Santa-Barbara, February 12, 1999.

[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.

[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.

[CHP97] 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.

[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.