Appears in the proceedings of the 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII), Cambridge, MA, Oct. 1996, pp. 128 -137.

Analysis of Branch Prediction via Data Compression

I-Cheng K. Chen, John T. Coffey, and Trevor N. Mudge
EECS Department, University of Michigan
1301 Beal Ave., Ann Arbor, Michigan 48109-2122
{icheng, scoffey, tnm}@eecs.umich.edu

Abstract

Branch prediction is an important mechanism in modern microprocessor design. The focus of research in this area has been on designing new branch prediction schemes. In contrast, very few studies address the theoretical basis behind these prediction schemes. Knowing this theoretical basis helps us to evaluate how good a prediction scheme is and how much we can expect to improve its accuracy.

In this paper, we apply techniques from data compression to establish a theoretical basis for branch prediction, and to illustrate alternatives for further improvement. To establish a theoretical basis, we first introduce a conceptual model to characterize each component in a branch prediction process. Then we show that current "two-level" or correlation based predictors are, in fact, simplifications of an optimal predictor in data compression, Prediction by Partial Matching (PPM).

If the information provided to the predictor remains the same, it is unlikely that significant improvements can be expected (asymptotically) from two-level predictors, since PPM is optimal. However, there are a rich set of predictors available from data compression, several of which can still yield some improvement in cases where resources are limited. To illustrate this, we conduct trace-driven simulation running the Instruction Benchmark Suite and the SPEC CINT95 benchmarks. The results show that PPM can outperform a two-level predictor for modest sized branch target buffers.

1. Introduction

As the design trends of modern superscalar microprocessors move toward wider issue and deeper super-pipelines, effective branch prediction becomes essential to exploring the full performance of microprocessors. A good branch prediction scheme can increase the performance of a microprocessor by eliminating the instruction fetch stalls in the pipelines. As a result, numerous branch prediction schemes have been proposed and implemented on new microprocessors [MReport95].

Many researchers focus on designing new branch prediction schemes solely based on comparing simulation results. However, very few studies address the theoretical basis behind these prediction schemes. Knowing the theoretical basis helps us to assess how good a prediction scheme is as well as how much more we can improve the existing predictors.

To establish a theoretical basis, we first introduce a conceptual system model to characterize components in a branch prediction process. Using this model, we notice that many of the best prediction schemes [Pan92, Yeh92, Yeh93, McFarling92, Chang94, Nair95b] use predictors similar to a "two-level" adaptive branch predictor [Yeh91]. Then, we demonstrate that these "two-level like" predictors are, in fact, simplified implementations of an optimal predictor in data compression, Prediction by Partial Matching (PPM) [Cleary84, Moffat90]. This establishes a theoretical basis for current two-level predictors that can draw on the relatively mature field of data compression.

In particular, the potential benefit of applying data compression techniques to branch prediction is readily apparent in the similarity of predictors used in both methods. In practice, the predictors used in branch prediction are only a very small subset of predictors developed in data compression. To illustrate the potential improvement using data compression techniques, we conduct trace-driven simulations. The results show that PPM outperforms equivalent two-level predictor in the Instruction Benchmark Suite (IBS) [Uhlig95] and the SPEC CINT95 [SPEC95] benchmarks. However, the improvement is not great, because two-level predictors are near optimal. In the case of modest size systems, the improvement is more significant.

This paper is organized into seven sections. In section 2 we introduce a conceptual system model to describe the process and components of branch prediction. We then use this model to summarize current popular branch prediction schemes. In section 3, we show that data compression is relevant to branch prediction because it also requires prediction.

In section 4, we develop a theoretical basis for two-level predictors by demonstrating that they are simplified versions of an optimal predictor, PPM. Section 5 considers the implication of the availability of optimal predictors, and shows that, in some cases, branch prediction can still benefit from data compression. We verify this with trace-driven simulation running the IBS and the SPEC CINT95 benchmarks. Section 6 discusses the potential benefits from data compression and further improvement in each component of our conceptual prediction model. Finally, we present conclusions and further work in section 7.

2. System Model and Overview of Branch Prediction

To explain branch prediction schemes, a conceptual view of a branch prediction scheme is introduced. This conceptual view allows us to compare various branch prediction schemes. It also enables us to focus and improve each component by clearly defining its function. This conceptual model elaborates on the one in [Young95]. Our model extends it to accommodate most popular branch prediction schemes.

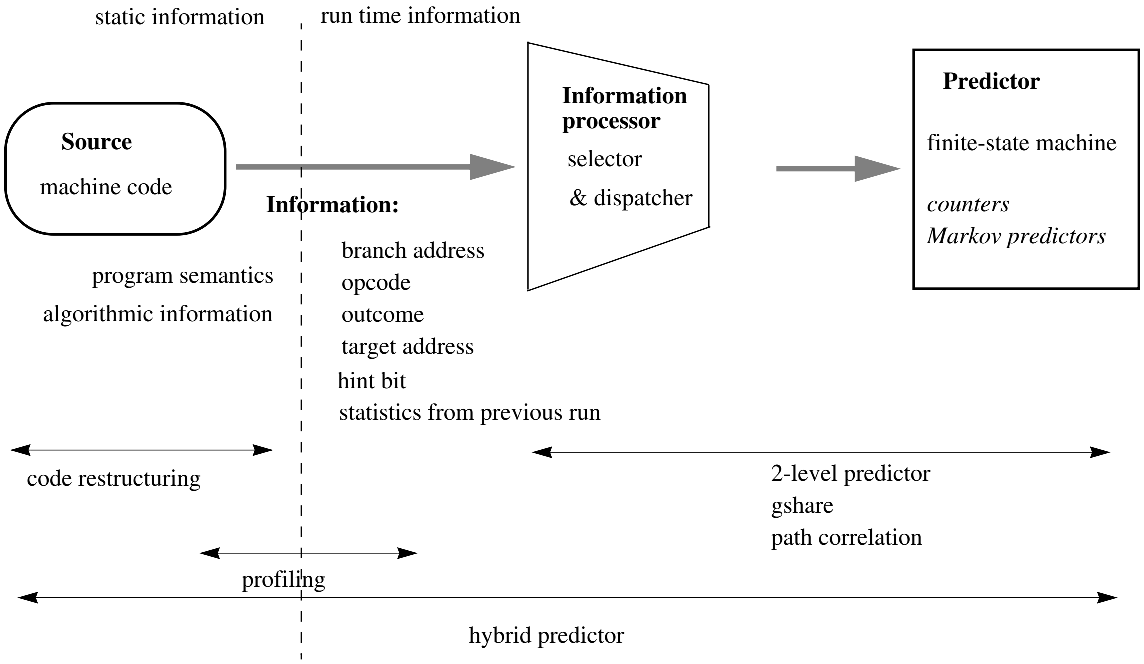


Figure 1: A conceptual system model for branch prediction

2.1 A general conceptual system model for branch prediction

The general conceptual model we introduce for branch prediction consists of three major components: a source, an information processor, and a predictor, as illustrated in Figure 1. Although some components are often combined in a hardware implementation, this three-part model is useful in explaining the principles behind different prediction schemes.

2.1.1 Source

The source is simply the machine code of the programs we are running. The source contains program semantics and algorithmic information. To aid branch prediction, this information can be explored and extracted during the compile-time. It can be stored and passed on to be used during execution. A hint bit in branch instructions is one means of passing this information. In addition, the source can be modified to produce more predictable branches using statistics from previous test-runs. This is how code restructuring and code profiling work.

2.1.2 Information processor

In a hardware implementation, the information processor is often combined with predictors and, hence, overlooked. However, the information processor plays a key role in the prediction process and thus deserves a close study. Conceptually, it can be subdivided into two components: selector and dispatcher.

2.1.2.1 Selector

The selector selects which run-time information should be used for branch prediction and encode it. This information can be branch address, operation code, branch outcome, target address, hint bits, or statistics from test-runs. Prediction accuracy depends heavily on the mix of run-time information that is employed.

Once the information is determined, the selector decides what formats to represent the information. For example, suppose branch outcomes and branch addresses are selected as information, the selector can combine the outcomes with addresses into one single stream or keep outcomes as individual streams classified by branch addresses. Good encoding can extract the essence of information, producing a concise and efficient representation to help prediction.

2.1.2.2 Dispatcher

The dispatcher determines how the information is mapped (fed) to the various predictors, since multiple information streams and predictors may exist in a prediction scheme. The mapping can be one-to-one, many-to-one, one-to-many, dedicated, or multiplexed (time-shared). Different mappings often have great influence on the final prediction accuracy.

2.1.3 Predictor

A predictor is simply a finite-state machine that takes input and produces a prediction. It does not need to know the meaning of the input. Common examples are a constant or static predictor, a 1-bit counter, a 2-bit up-down saturating counter [Smith81], and a Markov predictor. A Markov predictor forms the basis of recent two-level prediction schemes and is discussed in detail in Section 3. For the moment, a Markov predictor is simply a finite-state machine that generates predictions based on a finite number of previous inputs.

2.2 Overview of current branch prediction schemes

Using the conceptual model just introduced, we can summarize current popular branch prediction schemes in Table 1. This table describes the basic components used in each prediction scheme. It lists whether source modification or profiling are used for each prediction scheme. For the information processor, it describes both the information used by the selector and the way dispatcher maps information. Finally, the predictor used in each prediction scheme is also listed. From this table, we notice that many of the best prediction schemes [Pan92, Yeh92, Yeh93, McFarling92, Chang94, Nair95b] use Markov predictors. We will explain this further in Section 4.

Prediction scheme	Source modification or profiling	Information processor		predictor
		selector	dispatcher	
forward not-taken, backward taken	no	(address - target)	many-to-one	constant
2-bit counter	no	outcome, classified by address	one-to-one, mapped with address	2-bit counters
path correlation	no	target, in execution order	one-to-many, mapped with address	several Markov predictors
gshare	no	address, outcome, XOR together	one-to-one	a Markov predictor
GAg	no	outcome, in execution order	one-to-one	a Markov predictor
GAs	no	outcome, in execution order	one-to-many, mapped with address	several Markov predictors
PAg	no	outcome, classified by address	many-to-one, multiplexed	a Markov predictor
PAs	no	outcome, classified by address	many-to-many, mapped with address	several Markov predictors
PSg	no	outcome, classified by address	many-to-one, multiplexed	constant
branch correlation	yes	statistics from previous runs, hint bit	one-to-one, mapped with address	constant
hybrid predictor	yes	combinations of above	combinations of above	combinations of above

Table 1: Summary of current popular prediction schemes

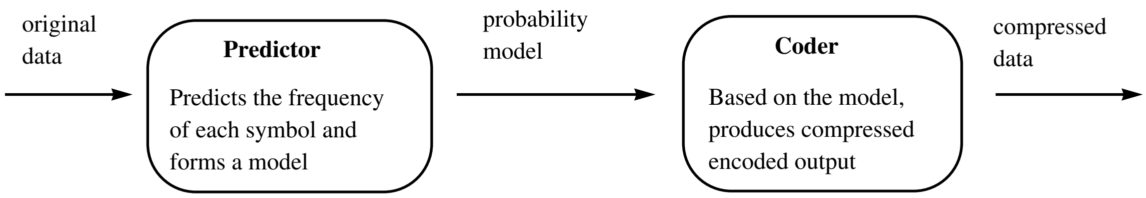


Figure 2: A two-step model for data compression

3. Data Compression and Prediction

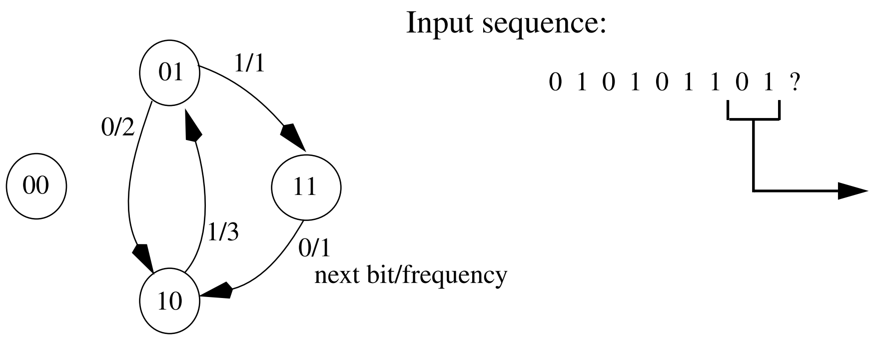
Like branch prediction, data compression relies on prediction. In data compression, the goal is to represent the original data with fewer bits. The basic principle of data compression is to use fewer bits to represent frequent symbols, while using more bits to represent infrequent symbols. Thus, the net effect is to reduce the overall number of bits needed to represent the original data. In order to perform this compression effectively, a compression algorithm has to predict future data accurately to build a good probabilistic model for the next symbol [Bell90]. Then, as shown in Figure 2, the algorithm encodes the next symbol with a coder tuned to the probability distribution. Current coders can encode data so effectively that the number of bits used is very close to optimal and, consequently, the design of good compression relies on an accurate predictor. The problem of designing efficient and general universal compressors/predictors has been extensively examined. In our experiments we draw on these techniques, adapting them to the new context of branch prediction.

3.1 Prediction by Partial Matching

Prediction by partial matching (PPM) is a universal compression/prediction algorithm that has been theoretically proven optimal and has been applied in data compression and prefetching [Cleary84, Krishnan94, Kroeger96, Moffat90, Vitter91]. Indeed, it usually outperforms the Lempel-Ziv algorithm (found in Unix *compress*) due to implementation considerations and a faster convergence rate [Curewitz93, Bell90, Witten94]. As described above, the PPM algorithm for text compression consists of a predictor to estimate probabilities for characters and an arithmetic coder. We only make use of the predictor. We encode the outcomes of a branch, taken or not taken, as 1 or 0 respectively. Then the PPM predictor is used to predict the value of the next bit given the prior sequence of bits that have already been observed.

3.1.1 Markov predictors

The basis of the PPM algorithm of order m is a set of $(m + 1)$ Markov predictors. A Markov predictor of order j predicts the next bit based upon the j immediately preceding bits—it is a simple Markov chain [Ross85]. The states are the 2^j possible patterns of j bits. The transition probabilities are proportional to the observed frequencies of a 1 or a 0 that occur given that the predictor is in a particular state (has seen the bit pattern associated with that state). The predictor builds the transition frequency by recording the number of times a 1 or a 0 occurs in the $(j + 1)$-th bit that follows the j-bit pattern. The chain is built at the same time that it is used for prediction and thus parts of the chain are often incomplete. To predict a branch outcome the predictor simply uses the j immediately preceding bits (outcomes of branches) to index a state and predicts the next bit to correspond to the most frequent transition



pattern	next bit	frequency count
00	0	0
	1	0
01	0	2
	1	1
10	0	0
	1	3
11	0	1
	1	0

Figure 3: Example of a Markov predictor of order 2

The Markov chain at left corresponds to the information collected from the input sequence in the table at right. Note that the chain is incomplete, because of 0 frequency count transitions.

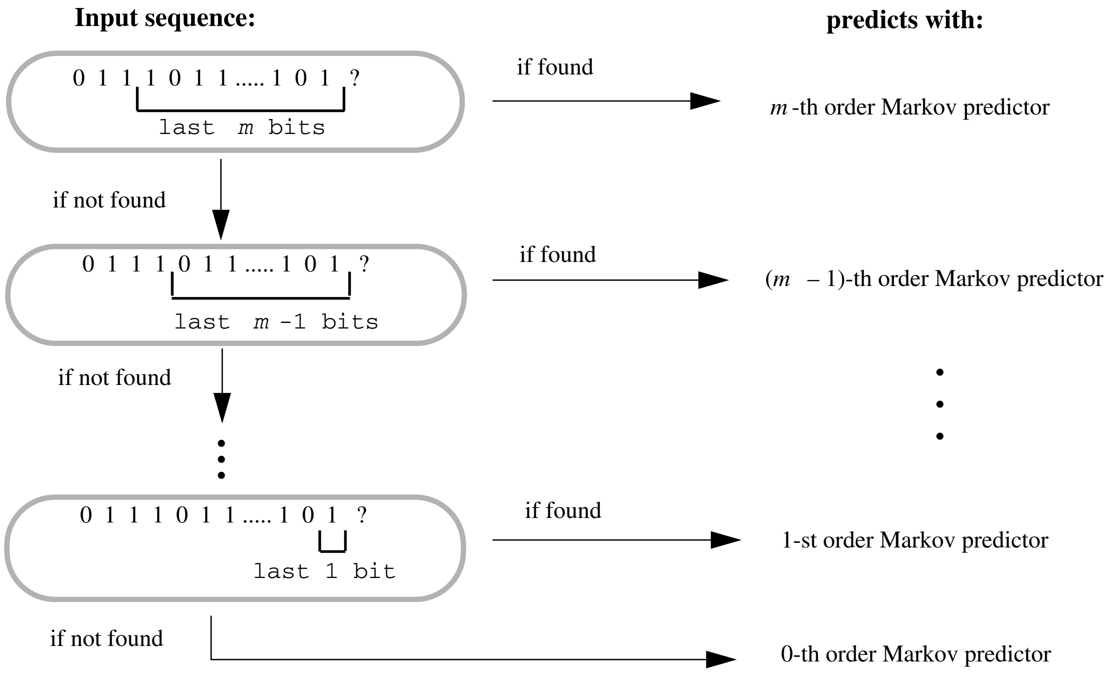


Figure 4: Prediction flowchart of a PPM predictor of order *m*

out of that state.

Figure 3 illustrates how a Markov predictor works. Let the input sequence seen so far be 010101101, and the order of Markov predictor be 2. The next bit is predicted based on the two immediately preceding bits, that is, 01. The pattern 01 occurs 3 times previously in the input sequence. The frequency counts of the bit following 01 are: 0 follows 01 twice, and 1 follows 01 once. Therefore, the predictor predicts the next bit to be 0 with a probability of 2/3. The (incomplete) 4-state Markov chain is shown at the left of the figure. Note that a 0-th order Markov predictor simply predicts the next bit based on the relative frequency in the input sequence.

3.1.2 Combining Markov predictors to perform PPM

We noted earlier that the basis of a PPM algorithm of order m is a set of $(m + 1)$ Markov predictors. The algorithm is illustrated in Figure 4. PPM uses the m immediately preceding bits to search a pattern in the highest order Markov model, in this case m. If the search succeeds, which means the pattern appears in the input sequence seen so far (the pattern has a non-zero frequency count), PPM predicts the next bit using this mth-order Markov predictor as described in the previous subsection. However, if the pattern is not found, PPM uses the $(m – 1)$ immediately preceding bits to search the next lower order $(m – 1)$-th order Markov predictor. Whenever a search misses, PPM reduces the pattern by one bit and uses it to search in the next lower order Markov predictor. This process continues until a match is found and the corresponding prediction can be made.

There are a number of variations on how the frequency

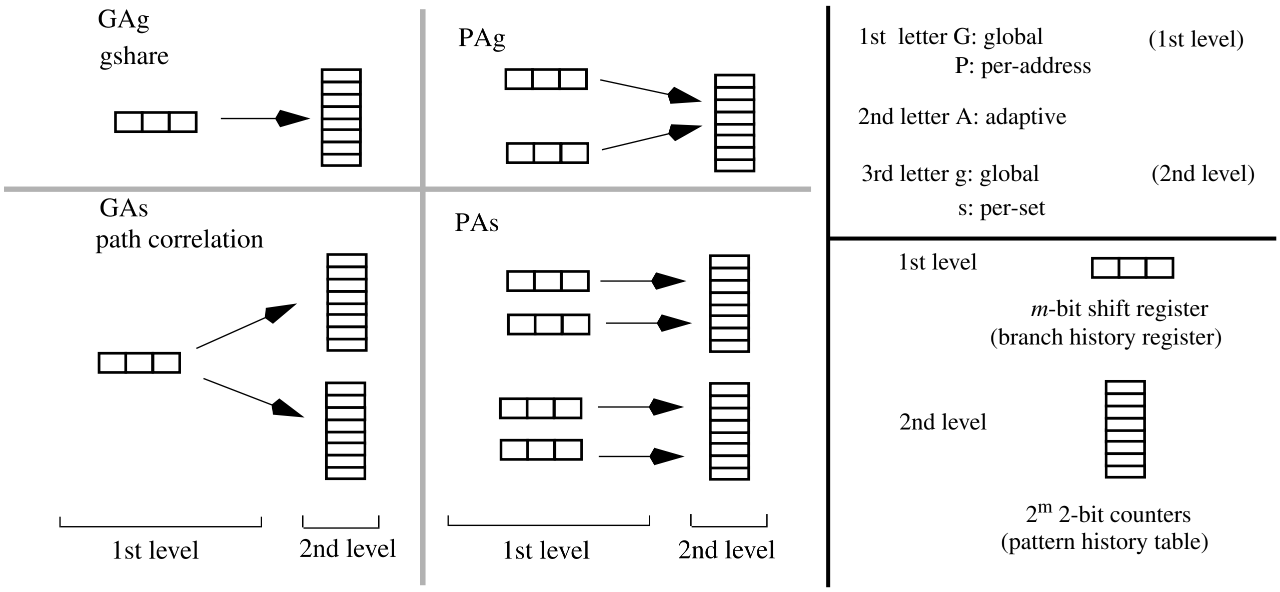


Figure 5: Popular variations of two-level predictors

information in the individual Markov predictors can be updated as the PPM process proceeds. In our experiments we use *update exclusion*. This means that we only update the frequency counters for the predictor that makes the prediction and the predictors with higher order. Lower order predictors are not updated.

4. Two-level Branch Prediction as an Approximation of PPM

In this section, we show that recently proposed two-level or correlation based predictors are approximations of, PPM, an optimal prediction algorithm.

4.1 Description of two-level predictor

Among the various branch prediction schemes, two-level or correlation based predictors are among the best. In addition, these predictors all share very similar hardware components. As Figure 5 shows, they have one or more shift-registers (branch history registers) to store history information in the first level and have one or more tables of 2-bit counters (pattern history tables) in their second level [Yeh91]. The contents of the first level shift-registers are typically used to select a 2-bit counter in one of the second-level tables. Predictions are made based on the value of the 2-bit counter selected.

4.2 Two-level branch predictors as Markov predictors

From the above discussion on two-level adaptive branch predictors and the one on Markov predictors in Section 3.1.1, it can be seen that there are strong similarities. Though different schemes of two-level branch predictors exist, they differ only in what information is used for history and what subsets of branch outcomes are used to index and update the counters. As a result, there exists a corresponding Markov predictor for each scheme.

Figure 6 shows the similarity between a two-level predictor and a Markov predictor. Both predictors behave exactly the same in the first level. They both use the last m bits of branch outcome to search the corresponding data structure. Note that an m-bit shift register serves two functions: first, it limits the information used for prediction to m previous outcomes and, second, it uniquely defines a finite-state machine in which each state has exactly two predefined next states. In the second level, the Markov predictor uses a frequency counter for each outcome, while the two-level predictor uses a saturating up-down 2-bit counter [Smith81]. Whenever a branch is taken/not taken, the 2-bit counter increments/decrements. The decision for a two-level predictor depends on whether the value of the counter falls in the positive half or the negative half. Similarly, a Markov predictor simply predicts the next branch to be the most frequent outcome based on two frequency counters. Both predictors are utilizing a majority vote via different implementations. The saturating counter is an approximation to this that can be realized in hardware efficiently.

An interesting illustration is to see how a two-level predictor, the per-address branch history register with global pattern history table (PAg), corresponds to a Markov predictor. This per-address scheme uses one table of 2-bit counters and multiple shift registers where each register records only outcomes of a particular branch. Although multiple shift registers exist, all shift registers operate the same and correspond to the same transition rule for a finite-state machine (state diagram). In addition, all shift registers share the same global table of 2-bit counters and, hence, share the same value (counter) in each state. Therefore, this per-address scheme uses one Markov predictor that is time-shared and updated among various branches.

4.3 Approximation to optimal predictors

Given the arguments in the previous sections, we now show that a two-level predictor is an approximation of an optimal predictor. As mentioned in Section 3.1, PPM is a theoretically proven optimal predictor consisting of a set of Markov predictors. Though performance is inferior at the beginning, a single Markov predictor can approach the performance of PPM in the long run (asymptotically) [Bell90]. Furthermore, we have shown that a two-level predictor is a simplified Markov predictor. Therefore, we can see that a two-level predictor is an approximation of an optimal predictor, PPM.

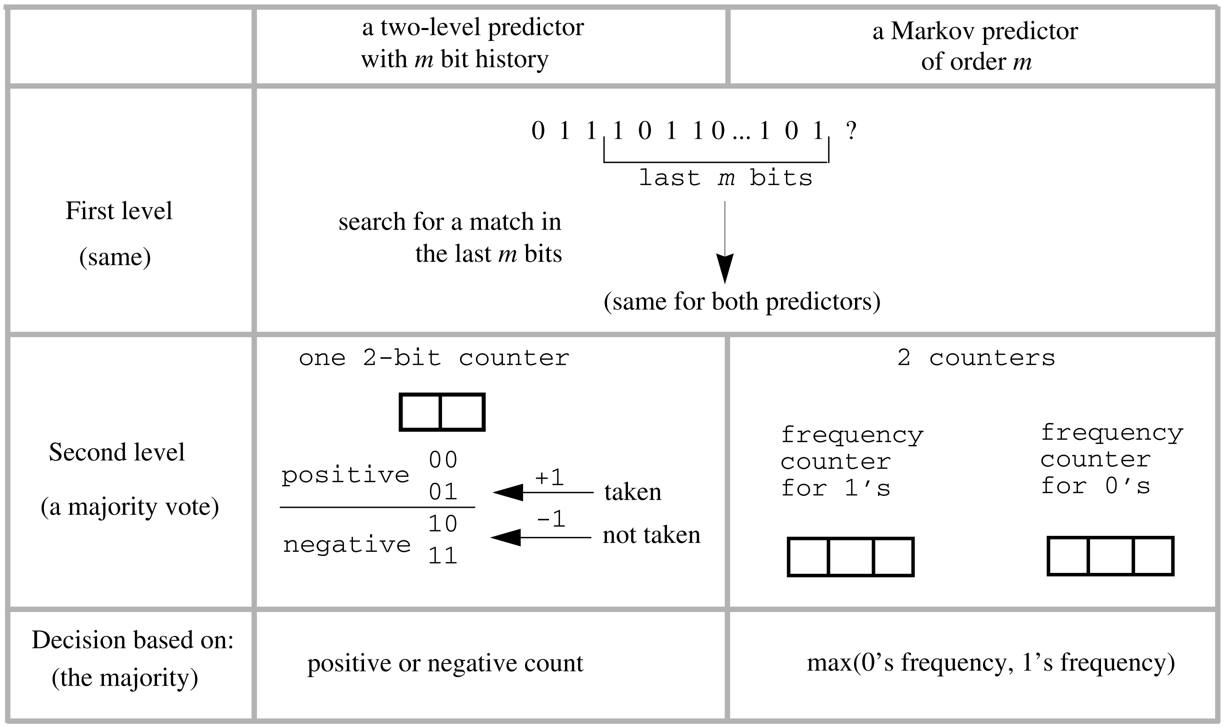


Figure 6: A two-level branch predictor vs. a Markov predictor

Under hardware implementation constraints, we think that a two-level predictor is a reasonable simplification of PPM. The complete PPM predictor can be viewed as a set of two-level predictors, having not one size of predictor (m) but a set that spans m down to 0 (a simple two-bit counter—equivalent to a per-address predictor with zero history length). These extra small predictors help to reduce "cold starts," i.e., lack of information at the training period. Although two-level predictors do not include small predictors, they still can perform well since cold starts are far less severe in branch prediction than in text compression. To see how cold starts differ in the two fields, we consider the number of all possible combinations of m outcomes. In branch prediction, there are 2^m possible combinations since a branch has only two possible outcomes (taken or not taken). In text compression, on the other hand, there are roughly 128^m possible combinations where 128 is the number of printable ASCII symbols. Compared to the large number of branches executed in typical programs, these 2^m cold starts are negligibly small and hardly decrease the overall prediction accuracy. Another simplification made by two-level predictors is the use of a 2-bit counter instead of an n-bit counter. This is a cost-effective choice, since two bits is the minimal number needed so that the direction of the predictor is not changed by the single exit in a loop statement [Lee84, Smith81].

As an aside, note that it is not coincidental that a 2-bit saturating up-down counter is the best among 4-state predictors [Nair95a]. This is because, with four states, one 2-bit saturating up-down counter is the best way to mimic the majority vote used in the Markov predictor. In the original Markov predictor, this voting prediction is done with two frequency counters (one for each outcome).

5. Impact of Optimal Predictors and Further Improvement

5.1 Implication of optimal predictors

Having shown that a two-level predictor is an approximation of an optimal predictor, we have established a theoretical basis for this type of branch predictor. Rather than just comparing simulation results, which does not tell us how well these predictors perform in general, we can now have a reasonable degree of confidence in the performance of two-level predictors. It is unlikely that, by improving the predictor component alone, we can generate significant improvement in prediction accuracy excepting in special cases discussed below. This is because two-level predictors already perform close to optimal under the constraints imposed by the information they are given. Of course, the inclusion of more information (e.g., knowledge about the program executing) can always be used to improve the prediction accuracy.

5.2 Illustration of modest improvements using PPM techniques

In this section, we illustrate that techniques from data compression can still, in some cases, yield modest improvements to branch prediction. To assess and confirm the potential improvement, we conduct trace-driven simulations. As input for the simulation, we use the Instruction Benchmark Suite (IBS) benchmarks [Uhlig95] and the SPEC CINT95 benchmark suite [SPEC95] for our simulation.

The IBS benchmarks are a set of applications designed to reflect realistic workloads. The traces of these benchmarks are generated through hardware monitoring of a MIPS R2000-based workstation. We use the traces collected under the operating system Ultrix 3.1, which include both kernel-level and user-level instructions.

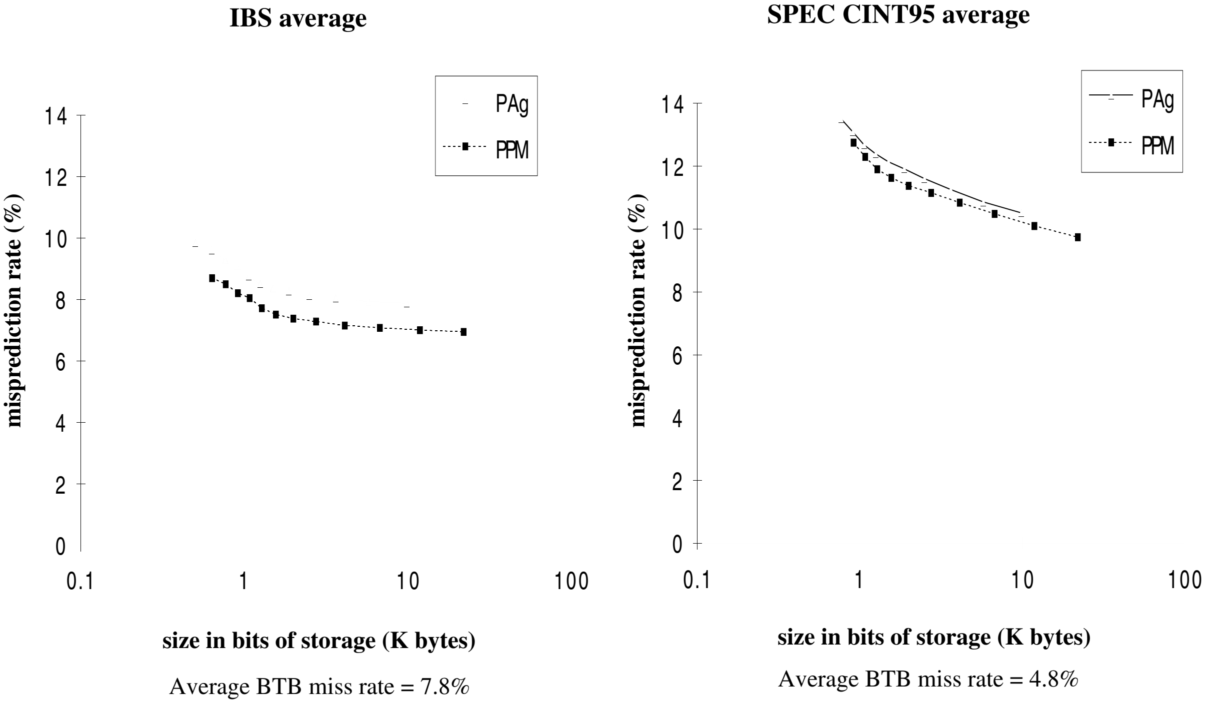


Figure 7: Misprediction rate for direct-mapped BTB with 1024 entries

For the SPEC CINT95 benchmark suite, we used ATOM [Eustace95], a code instrumentation interface from Digital Equipment Corporation, to collect our traces. The benchmarks are first instrumented with ATOM, then executed on a DEC 21064-based workstation running the OSF/1 3.0 operating system to generate traces. These traces contain only user-level instructions.

The statistics of traces from the IBS and the SPEC CINT95 are summarized in Table 2. All traces are identical in format and are used as input to our simulator.

	Benchmarks	static conditional branches	dynamic conditional branches
SPEC CINT95	compress	95	10,216,264
	gcc	15,647	24,048,361
	go	4,742	18,168,554
	ijpeg	902	40,854,598
	li	345	24,977,690
	perl	1,576	31,309,305
	vortex	5,963	24,979,201
IBS	groff	6,333	11,901,481
	gs	12,852	16,307,247
	mpeg_play	5,598	9,566,290
	nroff	5,249	22,574,884
	real_gcc	17,361	14,309,867
	sdet	5,310	5,514,439
	verilog	4,636	6,212,381
	video_play	4,606	5,759,231

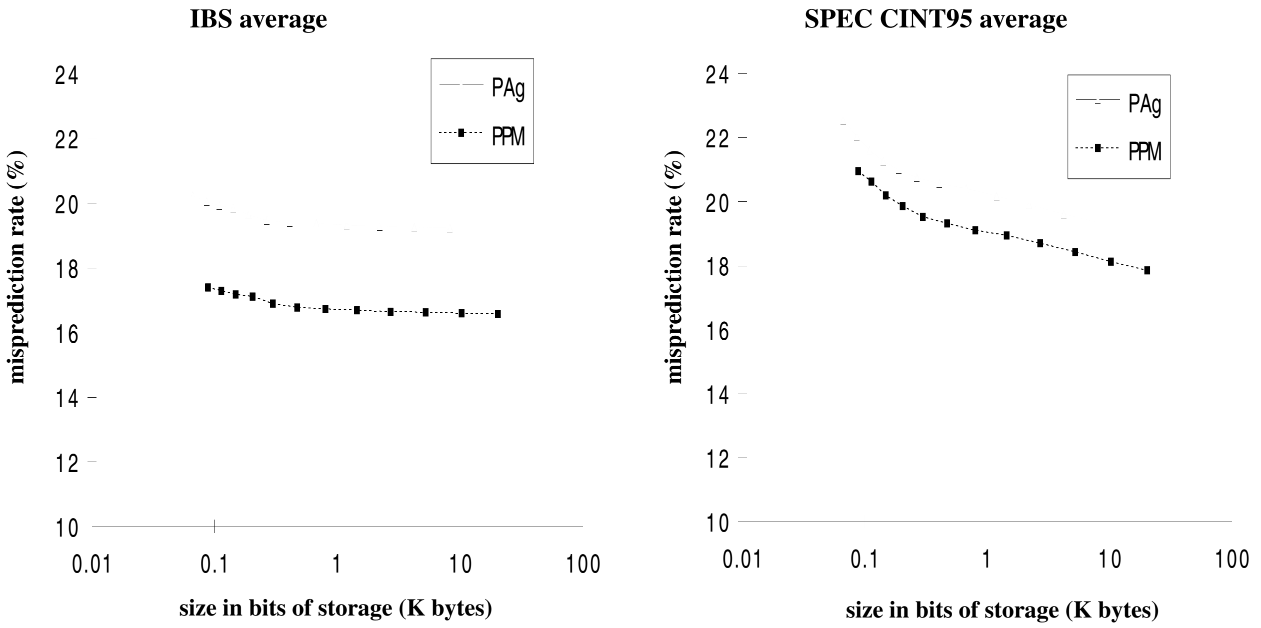
Table 2: Static and dynamic conditional branch counts in the IBS and SPEC CINT95 programs

Input to the SPEC CINT95 benchmarks was a reduced input data set; each benchmark was run to completion.

By using a set of small predictors, PPM can predict relatively well in situations where little history information is available, such as conflict misses or cold starts. In particular, this occurs in per-address prediction schemes where a finite branch-target buffer is used to record individual branch history, since the history of a particular branch may be replaced out by that of other branches due to contention in the finite buffer (conflict misses). For these cases, PPM can be used to alleviate the problem.

To illustrate potential improvement, we compare the PAg two-level predictor scheme and PPM. PAg means that the inputs are divided into per-address branch outcome streams, and then they are fed into one global predictor. Compared to global schemes, the advantage of PAg, and other per-address schemes, is that aliasing that may arise by mixing streams from different branch histories is reduced. However, conflict misses in the branch target buffer (BTB) become a more significant problem because only finite records of distinct branches can be maintained. This is due to limited buffer size; consequently, the history will be replaced and lost from time to time. The conflict miss problem gets worse as the number of distinct branches increases, since more individual branch outcome streams must be recorded.

Figure 7 shows the misprediction results for a direct-mapped BTB with 1024 entries. The vertical axis indicates branch misprediction rate, and the horizontal axis indicates the size of predictors. The left graph is the average of the IBS benchmarks, while the right graph is the average of the SPEC CINT95 benchmarks. As indicated by the relatively lower misprediction curve, PPM outperforms PAg, and the improvement of PPM is greater in IBS than in the SPEC CINT95. The small improvement in SPEC is due to its



Average BTB miss rate = 29.9%

Average BTB miss rate = 19.4%

Figure 8: Improved accuracy of PPM predictor with a direct-mapped BTB wit 128 entries

small miss rate (4.8%) in the BTB. This is because small miss rate implies little history information loss (occurred during misses); however, PPM is better than PAg only in cases when history information is not available, such as conflict misses and cold misses. The small BTB miss rate in the SPEC is because only a small number of distinct branches contribute to the vast majority of branch instances for most benchmarks [Sechrest96].

To see the effect of a small BTB, we reduce the number of entries from 1024 to 128. As shown in Figure 8, PPM again performs better than the PAg scheme for both benchmarks. Notice that, with a smaller BTB, the improvement of PPM over PAg is more pronounced. Also, as in the previous case, the improvement of PPM is greater in IBS than in the SPEC CINT95. The improvement of PPM is proportional to the BTB miss rate (history information lost).

As the results show, PPM performs better than two-level predictors. The improvement comes from a better mechanism for dealing with conflict (and cold) misses in which a set of Markov predictors rather than just the largest one are employed, as PAg does. If PPM cannot find a complete length of the branch history information, it reduces the length and search in the lower predictors. On the other hand, two-level predictors use incorrect history information, which can lead them to index to the wrong counters. The accuracy decreases because not only is the wrong counter selected for prediction, but it is also incorrectly trained. PPM solves the conflict miss problem more gracefully than two-level predictors. This is why PPM performs better when the miss rate is pronounced.

To conclude this section, we briefly describe some implementation details. The PPM in our simulation uses 2-bit counters as the PAg does. The PPM implementation is essentially a set of PAg systems with history registers of length m, m - 1, m - 2, etc. Thus, the second-level table in the PPM implementation requires $(2^m + 2^{m-1} + 2^{m-2} + ... + 2^0)$ 2-bit counters. All together, this adds to about twice the number of bits required in the PAg system. This is reflected in Figures 7 and 8.

6. Discussion of Further Improvement

Compared to the field of branch prediction, data compression is a mature field that has been well studied for decades. For example, two-level branch predictors were proposed in the early 1990s, while in data compression a superset of them (PPM) had been proposed and studied in the 1980s. The potential benefit of applying data compression techniques to branch prediction is readily apparent in the similarity of predictors used in both methods. Because the predictors serve the same purpose in both fields, they can be used interchangeably. As shown in Figure 9, predictors used in branch prediction are only a very small subset of predictors developed in data compression. By exploring the opportunity to borrow techniques from data compression, we may be able to improve branch prediction.

Although two-level predictors may perform close to optimal, as our previous experiments suggest there is still some room to improve the prediction accuracy. Consider the three components: the predictor, the information processor, and the source.

6.1 Improvement of the predictor

Optimal accuracy has not yet been achieved by two-level predictors since they are only a subset of optimal predictors. In addition, even if optimal predictors are available, it is still not clear how fast they can achieve optimal prediction accuracy. Therefore, we may further improve the predictor in the following ways.

6.1.1 Implementation of full-fledged optimal predictors

We demonstrated that two-level predictors have not achieved optimal accuracy using a Quicksort program whose optimal branch predictability can be analyzed exactly [Mudge96]. Currently, due to the limitation of hardware resources, it may not be cost-effective to implement full-fledged optimal predictors. However, as hardware budgets increase with technology advancement,

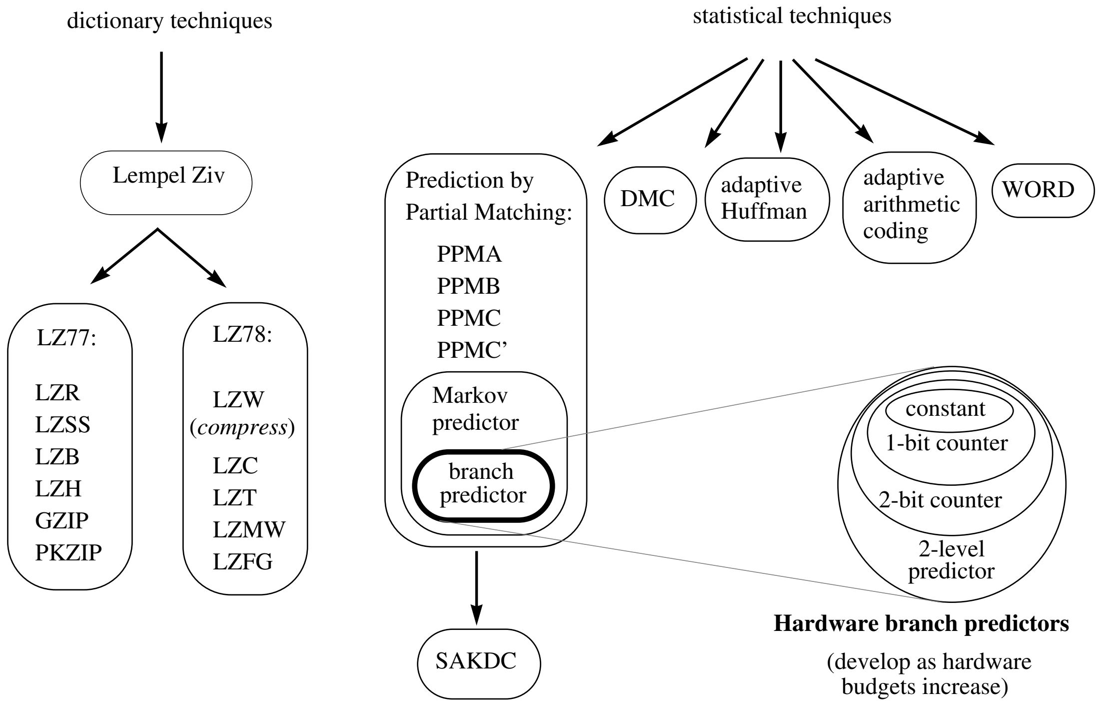


Figure 9: Branch predictors as a subset of predictors used in data compression

more features of optimal predictors can be added to achieve higher prediction accuracy. Examples would be the use of *n*-bit counters instead of 2-bit counters and the use of a variable length history instead of a fixed length. In the meantime, the design of predictors is an optimization under hardware constraints. Economic design and careful handling of details are required since every percent of improvement is important.

6.1.2 Design of other optimal predictors

There are several optimal predictors with different costs and levels of efficiency. Furthermore, depending on the type of application, an optimal predictor may have different levels of efficiency [Bell90]. For example, the Lempel-Ziv predictor (found in Unix *compress*) and PPM are both optimal predictors. While the Lempel-Ziv predictor has a faster prediction speed, PPM has higher accuracy in general. Yet in the long run, they can both achieve optimal accuracy. Therefore, depending on the application and the speed constraint, we may prefer one to the other.

6.1.3 Design of efficient non-optimal predictors

A non-optimal, yet efficient, predictor may have higher accuracy than an optimal predictor in some cases. This happens when programs end or change behavior too soon before an optimal predictor can reach optimal accuracy. Therefore, though an efficient non-optimal predictor can never reach optimal accuracy, it may achieve higher accuracy in short or fast-changing programs. An example in data compression would be Dynamic Markov Compression (DMC) [Bell90].

6.2 Improvement of the information processor

Even with optimal predictors, we can still increase accuracy by improving the information processor. Good information selection, encoding, and dispatching can extract the essence of branch behavior and, hence, improve prediction accuracy. In particular, this information processing is important since the predictor does not know the meaning of its input. Even using the same predictor, different information processing can result in prediction schemes with varied accuracy. Information describing branch behavior includes: branch address, branch outcome, operation code, target address, hint bits, and statistics from previous runs. How to best exploit and represent this information still remains to be studied. Examples of prediction schemes attempting to improve the information processor are the gshare scheme [McFarling92] and the path correlation scheme [Nair95b].

6.3 Improvement of the source

We can fundamentally improve the predictability of the branches by changing the source and, thereby, their behavior. A more predictable source can be derived by adding algorithmic knowledge and run-time statistics from test-runs. The goal is to decrease the entropy of the source by making the outcomes of branches more unevenly distributed. An example is code restructuring with profiling information [Calder94, Young94].

7. Conclusions and Further Work

In this paper, we establish the connection between data compression and branch prediction. This allows us to draw techniques from data compression to form a theoretical basis for branch prediction. In particular, we show that current two-level adaptive branch predictors are approximations of an optimal predictor, PPM. Based upon this theoretical basis rather than just simulation results, we can now have a reasonable degree of confidence in the performance of two-level predictors. Although two-level predictors are close to optimal if unlimited resources are available, PPM can still outperform two-level predictors when branch-target buffers are small. This is because PPM has better mechanisms for handling misses.

To illustrate directions for further improvement, we introduce a conceptual model, which consists of three components: a predictor, an information processor, and a source. For the predictor, we can borrow the rich set of predictors developed in data compression and apply them to branch prediction. However, since PPM is optimal, it is unlikely that significant improvement can be made by improving the predictor alone, except for the cases noted. Therefore, to further increase branch prediction accuracy, the focus should be on improving the information processor and the source.

Acknowledgments

This work was supported by Advanced Research Projects Agency under ARPA/ARO Contract Number DAAH 04-94-G-0327.

References

[Bell90] Bell, T. C., Cleary, J. G. and Witten I. H. *Text Compression*. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[Calder94] Calder, B. and Grunwald, D. *Reducing branch costs via branch alignment.* Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, 242-251, 1994.

[Chang94] Chang, P., Hao, E., Yeh, T. and Patt, Y. *Branch classification: a new mechanism for improving branch predictor performance*. Proceedings of the 27th Annual International Symposium on Microarchitecture, 22-31, November 1994.

[Cleary84] Cleary, J. G. and Witten, I. H. *Data compression using adaptive coding and partial string matching.* IEEE Transactions on Communications, Vol. 32, No. 4, 396-402, April 1984.

[Curewitz93] Curewitz K. M., Krishnan, P. and Vitter, J. S. *Practical prefetching via data compression.* Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, 257-266, May 1993.

[Eustace95] Eustace, A. and Srivastava, A. *ATOM: A flexible interface for building high performance program analysis tools.* Proceedings of the Winter 1995 USENIX Technical Conference on UNIX and Advanced Computing Systems, 303-314, January 1995.

[Krishnan94] Krishnan, P. and Vitter, J. S. *Optimal prediction for prefetching in the worst case.* Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, January 1994.

[Kroeger96] Kroeger, T. M. and Long, D. D. E. *Predicting file system actions from prior events.* Proceedings of USENIX Winter Technical Conference, January 1996.

[Lee84] Lee, J.K.F. and Smith, A. J. *Branch prediction strategies and branch target buffer design.* IEEE Computer, Vol. 21, No. 7, 6-22, January 1984.

[McFarling92] McFarling, S. *Combining branch predictors.* WRL Technical Note TN-36, June 1993.

[Moffat90] Moffat, A. *Implementing the PPM data compression scheme*. IEEE Transactions on Communications, Vol. 38, No. 11, 1917-1921, November 1990.

[MReport95] Microprocessor Report, Sebastopol, CA: MicroDesign Resources, March 1995.

[Mudge96] Mudge, T., Chen, I-C. K. and Coffey, J. T. *Limits to branch prediction.* Technical Report CSE-TR-282-96, University of Michigan, 1996.

[Nair95a] Nair, R. *Optimal 2-bit branch predictors.* IEEE Transactions on Computers, Vol. 44, No. 5, 698-702, May 1995.

[Nair95b] Nair, R. *Dynamic path-based branch correlation.* Proceedings of the 28th Annual International Symposium on Microarchitecture, 15-23, November 1995.

[Pan92] Pan, S-T., So, K. and Rahmeh, J. T. I*mproving the accuracy of dynamic branch prediction using branch correlation.* Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, 76-84, 1992.

[Ross85] Ross, S. M. *Introduction to probability models.* London, United Kingdom: Academic press, 1985.

[Sechrest96] Sechrest, S., Lee, C-C. and Mudge, T. *Correlation and aliasing in dynamic branch predictors.* Proceedings of the 23rd International Symposium on Computer Architecture, 22-32, May 1996.

[Smith81] Smith, J. E. *A study of branch prediction strategies.* Proceedings of the 8th International Symposium on Computer Architecture, 135-148, May 1981.

[SPEC95] SPEC CPU'95, Technical Manual, August 1995.

[Uhlig95] Uhlig, R., Nagle, D., Mudge, T., Sechrest, S. and Emer, J. *Instruction Fetching: Coping with Code Bloat.* Proceedings of the 22nd International Symposium on Computer Architecture, 345-356, June 1995.

[Vitter91] Vitter, J. S. and Krishnan, P. *Optimal prefetching via data compression.* Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 121-130, October 1991.

[Witten94] Witten I. H., Moffat, A. and Bell T. C. *Managing Gigabytes*. New York, NY: Van Nostrand Reinhold, 1994.

[Yeh91] Yeh, T-Y. and Patt, Y. *Two-level adaptive training branch prediction.* Proceedings of the 24th Annual International Symposium on Microarchitecture, 51-61, November 1991.

[Yeh92] Yeh, T-Y. and Patt, Y. *Alternative implementation of Two-Level Adaptive Branch Prediction.* Proceedings of the 19th International Symposium on Computer Architecture, 124-134, May 1992.

[Yeh93] Yeh, T-Y. and Patt, Y. *A comparison of dynamic branch predictors that use two levels of branch history.* Proceedings of the 20th International Symposium on Computer Architecture, 257-266, May 1993.

[Young94] Young, C. and Smith, M. *Improving the accuracy of static branch prediction using branch correlation.* Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, 232-241, 1994.

[Young95] Young, C., Gloy, N. and Smith, M. *A comparative analysis of schemes for correlated branch prediction.* Proceedings of the 22nd International Symposium on Computer Architecture, 276-286, June 1995.