Properties of Branch Prediction Schemes

Franz Edward Boas
fboas@das.harvard.edu

Under the direction of
Dr. Mike Smith
Division of Applied Sciences
Harvard University

1994

Summary

Pipelined processors operate quickly because they can start processing an instruction before the previous one has been completed. In order to do this, the computer must be able to predict what the next instruction will be before the previous instruction has been processed. Most pipelined processors use the two-bit scheme for program flow prediction. This project introduces the Markov and run-length schemes, and shows that they are more accurate than the two-bit scheme for some programs.

Abstract

Pipelined processors must accurately predict the direction of conditional branches to prevent pauses in program execution. The statistical properties of the behavior of branches and their implications towards branch prediction are explored in this paper. It was discovered that statistical information on the past behavior of a branch can be used to accurately determine the future behavior of the branch. For actual program branches, the length-two Markov scheme made up to 60% fewer prediction mistakes than the conventional two-bit scheme, and the run-length scheme made up to 68% fewer prediction mistakes than the two-bit scheme. The behavior of these two prediction schemes on simulated branches is used to explain why they outperform the two-bit scheme on actual programs.

Properties of Branch Prediction Schemes

1 Introduction

In order to operate quickly, computer hardware must accurately predict program flow so that the correct portions of the program can be fetched for future execution [3]. If the hardware predicts incorrectly, then processing must stop while the correct instructions are fetched. To simplify these predictions, programs are often divided into basic blocks, which are sections of code that are always executed together. Basic blocks end with a jump statement, a conditional branch, or right before a line referenced by a jump or branch statement, and they do not contain any internal branching or jumping statements. The following segment of code, for example, contains four basic blocks:

```
        basic block 1
        if x>10 then goto line a

        basic block 2
        goto line b

a:      basic block 3

b:      basic block 4
```

Program flow after a block which does not end in a conditional branch (blocks 2 and 3 in the example above) is easy to predict because the next block to be executed is the same every time. Blocks 2 and 3, for instance, are always followed by the execution of block 4. Program flow after a block which does end in a conditional branch (block 1 in the example), is not so easy to predict because the next block to be executed depends on the value of the test condition. If x>10 then the branch is taken and program execution continues at block 3. Otherwise, program execution falls through to block 2. Quickly and accurately predicting the direction of these conditional branch statements is essential in pipelined processors. Without a prediction mechanism, the processor would have to determine if x>10 before fetching the next instruction, thus creating a gap in the pipeline. With a prediction mechanism, however, the next instruction can be fetched before the value of x is even known.

History-based approaches to the branch prediction problem base their predictions on the previous behavior of the branch, and are classified as either dynamic or static. Static schemes predict the direction of each branch at compile time, and never change this prediction after compilation. This is usually done by running the program with test data, and then basing the prediction on the direction that was most often taken for each branch during the test run. One benefit of static schemes is that they improve the efficiency of global optimizers [6]. Dynamic schemes, on the other hand, modify their predictions as the program executes. One common implementation of this approach is the two-bit scheme, in which each conditional branch is associated with a two-bit saturating counter [4]. A two-bit counter value of 0 or 1 predicts the associated branch to fall through, while a counter value of 2 or 3 predicts the branch to be taken. Initially, each counter is seeded with a value of 2. Each taken branch increments a counter, and each fall-through branch decrements that counter. This allows the two-bit scheme to adapt to changes in a branch's behavior.

Many other solutions to the branch prediction problem have been suggested. Young and Smith developed a static scheme which improves static branch prediction accuracy by taking advantage of the correlation that often exists between the behavior of a branch and the path that was taken to get to that branch [1]. Their algorithm does this by creating another copy of a block of code if some paths to that block cause it to take more often and others cause it to fall through more often. After the block is duplicated and the correct paths are linked to each block, one copy can predict take and the other can predict fall through. Dynamic schemes can also take advantage of the correlation among the behaviors of different branches. One implementation involves recording the take / fall-through behavior of the last n branches executed. Each branch is associated with a table of 2^n two-bit counters, and the behavior of the last n branches determines which counter to use [5].

This paper ignores branch correlation information to concentrate on how much of a branch's behavior can be determined from its history alone. Three statistical properties of the take / fall-through sequences of branches are explored. Each of these properties can be used to predict future branch behavior. This is done by using the statistic to calculate the probability that the branch will be taken next. The branch should be predicted taken if this probability is greater than 0.5 and should be predicted fall through otherwise.

The first property under consideration is the percent of the time the branch is taken. The prediction scheme associated with this property, profile prediction, always predicts take if a branch was taken more than half of the time in the test, and always predicts fall through otherwise.

The second property associates a set of Markov chain probabilities with the behavior of the branch. Markov chains are sequences for which the terms depend probabilistically on the previous terms in the sequence. For a sequence of takes and fall throughs, the length n Markov chain probabilities would be the probabilities that a take occurs given the previous n terms in the self-history sequence. For example, if we let T stand for take and F for fall through, then the length two Markov chain probabilities would be: the probability TT is followed by a T, the probability TF is followed by a T, the probability FT is followed by a T, and the probability FF is followed by a T. As in profile prediction, these probabilities can be used to predict the subsequent behavior of the branch. Smith and Lee [2] consider Markov prediction briefly in their paper.

The final property is the run-length distribution. A run consists of one or more consecutive takes or fall throughs, so the run-length distribution is simply the distribution of the lengths of runs for a branch. If $f(n)$ gives the proportion of fall-through runs that are of length n, and x is the length of a fall-through stretch that hasn't ended yet, then the probability the branch will be taken next is prob(fall-through stretch is of length x given that fall-through stretch is at least of length x) which equals $\frac{f(x)}{\sum_{i=x}^{\infty} f(i)}$. If this probability is greater than 0.5 then the branch should be predicted taken. A similar method can be used to predict when a fall through will end a run of takes. The author has not found any reference to run-length prediction algorithms in the literature.

2 Materials and Methods

2.1 Artificially generated sequences

Each prediction method was first tested on artificially generated sequences because the branch behavior of actual programs is often complex and subtle. The program developed for these tests prompts the user for information describing a sequence, then calculates the accuracy of the two-bit, profile, Markov, and run-length distribution schemes on that sequence. The user describes the sequence to test in terms of the same statistical properties used by the branch prediction schemes: the percent of the time it is taken, the Markov chain probabilities, or the run-length distribution. A sample run of the program is shown in Section 7.1. In this project, the program was used to determine how the accuracy of each prediction scheme varies as a function of the proportion of takes for a sequence, and as a function of Markov chain probabilities. Information about the behavior of prediction schemes on these hypothetical sequences will allow us to better understand their behavior in actual programs.

2.2 Actual programs

The accuracy of the branch prediction algorithms for actual programs was determined not by actual implementations of the algorithms, but rather by simulations of how well the algorithms would have run had they been implemented. Simulations were used because they accurately model the behavior of branch prediction algorithms and allow for flexibility in the types of program statistics that are recorded.

Before the simulation could be run, a trace file containing the complete program flow needed to be generated. In this experiment, traces were generated using pixie, a tool available for tracing the execution of MIPS processors. Pixie generated a list of the addresses of the basic blocks for an executable file and created a modified version of this executable file that outputs an instruction trace when run. The basic block addresses and the instruction trace were then fed into a simulator written by Smith and his students to test his branch correlation algorithm. This simulator was modified by the author for the purposes of this project.

With these modifications, the simulator outputs statistics on the take / fall-through sequences of individual basic blocks in the program and the accuracy of the two-bit, profile, Markov, and run-length schemes for the program as a whole and for individual blocks. No one algorithm produces the most accurate prediction for all branches, suggesting a hybrid prediction algorithm in which each branch uses the prediction scheme which worked the best during the training runs. The simulator prints out the accuracy for this hybrid scheme as well. Information needed for run-length prediction requires much more memory than any of the other schemes. Although the amount of memory needed could be easily obtained by the simulator, it would be quite expensive to implement in an actual hardware version. Thus, the simulator only records run-length prediction information for the 5 most commonly executed blocks, meaning that a much smaller amount of memory is required.

The simulator steps through a training trace three times to collect the information needed by the branch prediction algorithms. On the first pass, the five most commonly executed blocks are found so that the simulator will know for which blocks it needs to collect run-length distribution information. On the second pass, the program tabulates branch statistics (percent taken, Markov chain probabilities, and run-length distribution for the five most frequently executed blocks). On the third pass, the two-bit, profile, Markov, and run-length schemes are simulated to determine the most accurate algorithm for each branch (needed for the hybrid scheme).

Finally, the program uses all of the branch prediction information gathered in the first three passes to predict program flow in an instruction trace which may be different from the sample trace used to train the prediction schemes. The results of the predictions are recorded as branch prediction accuracies.

Program	Description	Program input
a	nested for loops	none
b	square root calculation	none
c	bubble sort	inverted array
		random array
		sorted array
d	quick sort	inverted array
		random array
		sorted array
awk	pattern scanning and processing	extensive test of awk
		simple scanning and printing
compress	SPECint92 file compression	reference input
		15 page postscript paper
troff	GNU version 2.0 text formatter	csh manual page
		gcc manual page
espresso	SPECint92 boolean minimization	reference input (bca.in)
		reference input (cps.in)

Table 1: Programs tested

Table 1 shows the programs that were tested using the simulator. All possible combinations of input data used during training and input data used to test the prediction schemes were simulated.

3 Results and Data

3.1 Artificially generated sequences

Prediction accuracies for the artificially generated sequences are graphed in Figures 1 through 5. Figure 1 shows the accuracy of *n*-bit and profile schemes for randomly-generated sequences as the probability taken varies from 0 to 1. Profile prediction is the best possible prediction scheme for completely random sequences because the future behavior of the sequence is independent of the previous behavior. N bit schemes for which *n* is large behave much like profile prediction because the n bit counter will make the same prediction as the profile scheme and will not change predictions unless there is a long run of incorrect predictions. N bit schemes for small *n* do not perform as well because they are easily influenced by runs of incorrect predictions.

The graphs for the accuracy of length 1, 2, 3, and 4 Markov chain prediction and for the accuracy of the run-length scheme are superimposed in Figure 2. Each of these schemes behaves exactly the same as the profile prediction scheme. Markov chain prediction attempts to improve prediction accuracy by basing its decision on the previous behavior of the branch, instead of just predicting the same thing every time as the profile scheme does. With random data, however, there is no sequential correlation to take advantage of, so the Markov scheme predicts the same every time. If the proportion taken is less than 0.5, then the probability that a take run continues is less than 0.5 and the probability a fall-through run continues is greater than 0.5, causing the run-length distribution scheme to predict fall through every time. Likewise, if the proportion taken is greater than 0.5, the run-length scheme will always predict take, again resulting in the same prediction as the profile scheme.

The accuracy of prediction schemes on sequences generated from length one Markov chain probabilities is diagrammed in Figure 3. In each graph, the x-axis is the probability that a fall through is followed by a take, and the y-axis is the probability that a take is followed by a fall through. The left-hand side of the graph corresponds to sequences with a low fall-through-to-take transition probability and thus long fall-through runs. The right-hand side of the graph corresponds to sequences with short fall-through runs. Likewise, the bottom of the graph corresponds to sequences with a low take-to-fall-through transition probability and thus long take runs. The top of the graph corresponds to sequences with short take runs. N-bit schemes work best for sequences with long runs that it can adapt to and worst for rapidly fluctuating sequences. Thus, they work well in the lower-left-hand corner where both take and fall-through runs are long. The profile scheme is most accurate when the sequence consists of almost all fall throughs or all takes. Because of this, it is most accurate in the upper-left-hand corner of the graph which corresponds to sequences of mainly fall throughs, and the lower-right-hand corner of the graph which is mostly takes. The Markov scheme does not work as well for the region in the center of the graph because sequences with Markov probabilities around 0.5 are less sequentially correlated and thus harder to predict than those with Markov probabilities near 0 or 1. The behavior of the run-length scheme is very similar to that of the Markov scheme because it too uses the recent behavior of the branch to predict its future behavior.

Figure 4 shows the data from Figure 3 in a slightly different form. The axes in Figure 4 still represent the F to T and T to F transition probabilities, but the color of the points in the graph now show the relative accuracies of the two-bit, profile, and length two Markov schemes. The black squares are regions where the Markov scheme outperformed all the others, and the white region is where no scheme had a statistically significant (95% confidence) lead. The run-length scheme was not considered in this graph because for these artificially generated sequences, its behavior is identical to that of the Markov scheme. Markov prediction is removed

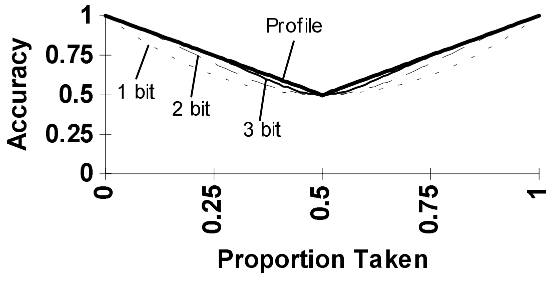

Figure 1: Accuracy of n bit and profile schemes for random sequences.

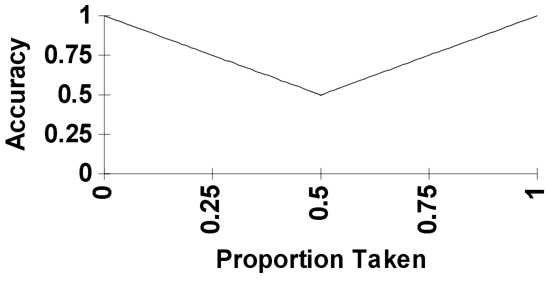

Figure 2: Accuracy of Markov and run-length schemes for random sequences

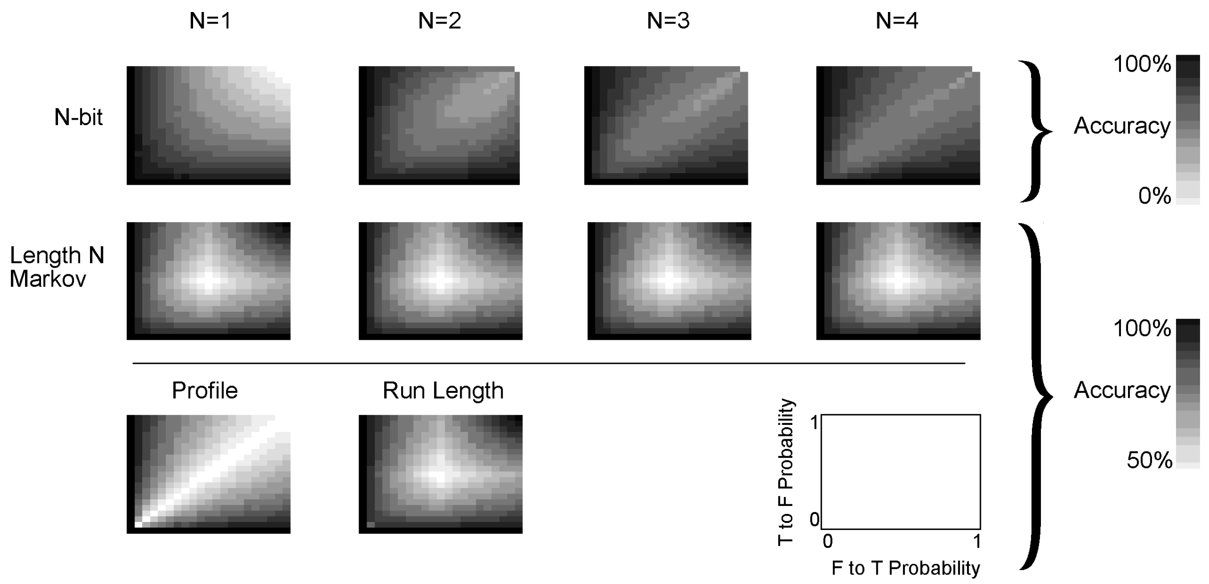

Figure 3: Accuracy of prediction schemes as a function of transition probabilities

from the competition in Figure 5. Here, the black region shows where the two-bit scheme did best, the gray region shows where no scheme had a statistically significant lead, and the white region shows where the profile scheme did the best.

If we assume that length one Markov chains adequately describe branch behavior in actual programs, then we can use the data shown in Figures 3, 4, and 5 to predict how well these algorithms will perform on real branches. Figure 3 shows us that the behavior of the Markov and run-length schemes on these simulated sequences is indistinguishable, so we can predict that Markov and run-length prediction will be equally accurate for actual programs.

For sequences which correspond to the white region in Figure 4, the Markov, run-length, and profile schemes are equally accurate, and the two-bit scheme may be slightly less accurate. This is because the white region comprises sequences for which the probability an F is followed by a T (the F to T transition probability) and the probability a T is followed by a T (1 - the T to F transition probability) are either both greater than 0.5, or both less than 0.5. Thus, information about whether the branch was take or fall through on its last execution does not affect the Markov or run-length scheme's prediction. Both schemes will make the same prediction every time, just like the profile scheme. The two-bit scheme may perform slightly worse because it could be lead astray by runs of two or more incorrect predictions.

In the black region of Figure 4, however, the Markov and run-length schemes will outperform the others because of their ability to use sequential correlation information. Thus, we can predict that overall, the Markov and run-length schemes will be the most accurate non-hybrid schemes. The hybrid scheme should of course perform the best overall.

3.2 Actual programs

		Two-bit	Profile	Markov	Run-length	Hybrid
Same	**Average rank**	4	4.5	2	3.5	1
	Average accuracy	91.94	90.02	93.81	90.88	94.25
Different	**Average rank**	3	4.5	2.2	4	1.2
	Average accuracy	91.94	83.81	87.35	84.1	92.61

Table 2: Comparison of branch prediction scheme accuracies

The results of the simulation on actual programs are shown in Table 2. Same refers to the simulations in which the same trace was used for training and for testing prediction schemes, and different refers to the simulations in which different traces were used. The hybrid scheme was the most accurate every time for the same simulations since it uses the most accurate scheme for each branch. In the different simulations, the hybrid scheme was beaten once by the two-bit scheme, suggesting that the properties of the branches in the training traces were sufficiently different from the properties of the branches in the testing traces that the best scheme for a branch in the training trace was often not the best scheme for the same branch in the testing trace.

As was predicted, the Markov scheme has the second highest average rank for both the same and different simulations. It is interesting to note that despite having a higher average rank than the two-bit scheme for the different simulations, the Markov scheme has a lower average accuracy. This is due to a single test in which the two-bit scheme performed well, but the profile, Markov, and run-length schemes all performed poorly. This test, the same one that caused the hybrid scheme to be beaten, shows that dynamic schemes are essential in programs for which no reliable training data is available or for which the range of possible inputs is very broad.

The prediction that the Markov scheme would outperform the two-bit and profile schemes came from Figure 4, which shows that the Markov scheme outperformed the two-bit and profile schemes for certain ranges of F to T and T to F transition probabilities, and tied with the profile scheme for others. The relevance of Figure 4 to actual branches is demonstrated in Figure 6, which shows the relative accuracies of prediction algorithms in terms of the transition

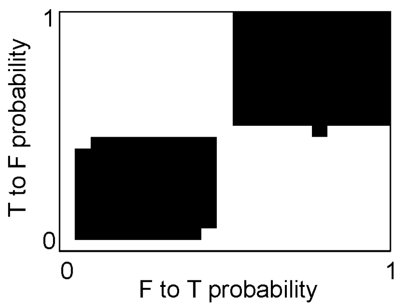

Figure 4: Relative accuracies of two-bit, profile, and length 2 Markov schemes as a function of the transition probabilities.
Black = Markov outperformed others
White = No scheme had a statistically significant lead (95% confidence)

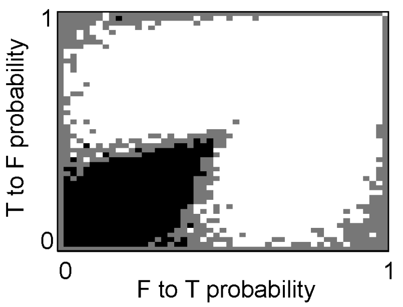

Figure 5: Relative accuracies of two-bit and profile prediction schemes as a function of transition probabilities.
Black = Two-bit scheme outperformed others
White = Profile scheme outperformed others
Gray = Neither scheme had a statistically significant lead (95% confidence)

probabilities in actual branches. Based on the information in Figure 4, the Markov scheme should do the best relative to the two-bit and profile schemes in Region 1, which corresponds to the black region in Figure 4. The Markov scheme should be tied with the profile scheme in region 2, which corresponds to the white region in Figure 4. This is approximately what we see in Figure 6. The Markov scheme was the most accurate in 83% of the branches in region 1. A tie for most accurate occurred in 76% of the branches in region 2.

The two-bit scheme has the fourth highest average rank for the same simulations and the third highest average rank for the different simulations. The two-bit scheme was expected to perform better in relation to the other schemes for the different simulations because it is dynamic and can adapt to changing program conditions whereas profile, Markov, and run-length prediction are static and unable to adapt. However, the two-bit scheme's average rank being lower than the Markov schemes average rank for the different simulations suggests that most of the programs tested in this experiment had fairly consistent behavior when given different inputs.

The run-length scheme performed noticeably worse than the Markov scheme in both the same and different simulations. However, since run-length distributions were only collected for the five most commonly executed blocks, the run-length scheme accuracy reflects the accuracy of using run-length prediction on those five blocks, and profile prediction in all of the other blocks. The run-length scheme was more accurate than the profile scheme in both the same and different simulations, meaning that for the five most commonly executed blocks, the run-length scheme must have outperformed the profile scheme significantly. To determine how well the run-length scheme actually performed requires recalculating branch prediction accuracy statistics so that they only reflect the contribution of the five most commonly executed blocks. These recalculated results are shown in Table 3.

		Two-bit	**Profile**	**Markov**	**Run-length**
Same	**Average rank**	3.1	2.9	1.8	1.3
	Average accuracy	90.63	89.44	91.61	93.2
Different	**Average rank**	2.5	2.5	2.8	2
	Average accuracy	90.63	84.83	85.64	86.62

Table 3: Comparison of branch prediction scheme accuracies for the five most commonly executed blocks

The relationships between the two-bit, profile, and Markov schemes have not changed much from Table 2. (The average ranks are consistently higher in Table 3 because the hybrid scheme is not considered there.) However, we can see from the table that the run-length scheme is the most accurate in terms of average rank for the blocks on which it is actually used. The two-bit scheme had a higher average accuracy for the different simulations because of the single program in which it performed much better than the other schemes. The behavior of the run-length and

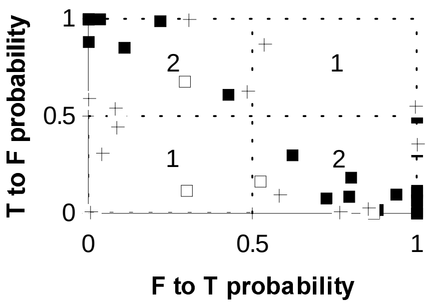

Figure 6: Relative accuracy of the length two Markov, two-bit, and profile schemes in terms of transition probabilities for actual programs.
■ = Tie
□ = Two-bit outperformed others
+ = Markov outperformed others

Markov schemes was remarkably similar for the artificially generated length one Markov sequences, suggesting that actual program branches contain nuances that cannot be modeled using length one Markov chains. Specifically, length one Markov sequences always have exponential run-length distributions, so any branch which has non-exponential run-length distributions could not be properly represented by a length one Markov sequence.

In last place for both the same and different simulations in Table 2 is the profile scheme. The profile scheme is the only one of the 5 schemes which does not take advantage of past history.

4 Discussion

The properties of prediction schemes were explored in this paper not only in the context of actual programs, but also in the more simplified context of artificially generated sequences. In this simplified context it is possible to quantify the behavior of branch prediction schemes and to determine the properties of sequences for which a prediction scheme does well or poorly.

It is important to keep in mind that branch prediction algorithms must be implemented in hardware and must be able to make predictions quickly. This means that these algorithms can not be extremely complicated. The hybrid scheme may be to complicated to be a useful branch prediction tool. The Markov scheme, however, should be quite easy to implement in hardware. Each branch needs to be associated with a two-bit number which gives the past history of the branch, and a fixed four bit number whose nth digit gives the prediction for when the two-bit number equals n. The run-length algorithm would be a little harder to implement. If run lengths of up to 2^n are recorded, then it requires n bits to store the length of the current run, one bit to tell if the run is take or fall through, and two 2^n bit numbers for prediction information. Due to the large amount of storage space involved, the run-length scheme could only be implemented for a few branches in each program. Some other algorithm, such as Markov prediction, could be used for the rest.

A possible future experiment would be to find an algorithm that would generate sequences of takes and fall throughs which closely modeled branch behavior in actual programs. This algorithm would allow realistic prediction scheme simulation in which the experimenter could control the parameters.

The simulations of the prediction schemes on actual programs showed that the length two Markov scheme outperforms the commonly-used two-bit scheme. Implementing a Markov scheme in hardware would be an interesting future project that would allow measurements on how well a Markov scheme would do in actual practice. One deficiency of the Markov and run-length schemes is that they cannot adapt to new program conditions. It is possible to make the Markov and run-length schemes adaptable: instead of associating a fixed prediction with each branch history or run-length, associate a two-bit counter. This would be another interesting possible future experiment.

5 Conclusion

The accuracy of the two-bit, profile, Markov, run-length, and hybrid prediction schemes under different conditions was measured in this project. The prediction schemes were first tested on sequences with statistical properties specified by the experimenter in order to better understand the properties of sequences for which each scheme works best. The data from these tests suggested that both the Markov and run-length schemes would be more accurate than the two-bit and profile schemes.

Each prediction scheme was then used to predict the direction of actual program branches. As predicted, the Markov scheme outperformed the two-bit, profile, and run-length schemes for the programs in which the input and output data were the same. The run-length scheme, which was only used on the five most commonly executed branches due to the relatively large amount of memory it requires, was more accurate than the Markov scheme for those branches. However, both the Markov and run-length schemes do not predict as well when the input used during training is different from the input used during the test of the prediction scheme. Further research is needed to determine if making the Markov and run-length schemes dynamic would improve prediction accuracy.

6 Acknowledgments

I would like to express my appreciation to Dr. Mike Smith of the Division of Applied Sciences at Harvard University for providing me with the exciting opportunity to work at the cutting edge of research in the computer field and thus further my knowledge of the subject. Dr. Smith generously allowed me use of his computer lab, and provided a great deal of guidance and encouragement on my project.

I thank Eli Olinick for his helpful comments on my paper.

I would also like to thank the Center for Excellence in Education for inviting me to participate in the Research Science Institute. RSI was without a doubt the most influential experience in my scientific career.

7 References

(1) C. Young and M. Smith, "Improving the Accuracy of Static Branch Prediction Using Branch Correlation," *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

(2) J. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, 17(1), Jan. 1984.

(3) M. Lam and R. Wilson, "Limits of Control Flow on Parallelism," *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

(4) J. Smith, "A Study of Branch Prediction Strategies," *Proceedings of the 8th Annual International Symposium on Computer Architecture*, Jun. 1981.

(5) S. Pan, K. So, and J. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proceedings of the 5th Annual International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1992.

(6) P. Chang, S. Mahlke, and W. Hwu, "Using Profile Information to Assist Classic Code Optimizations," *Center for Reliable and High-Performance Computing Report CRHC-91-13*, University of Illinois Urbana-Champaign, Urbana, IL, Apr. 1991.

8 Sample Program Output

8.1 Artificial Sequence Generator

```
(1) Specific sequence or (2) Matrix of sequences: 1
Training sequence description
```

```
(1) Random, (2) Markov, (3) Distribution of run lengths, (4) File: 3
Length 1  T: 0
Length 2  T: 1
Length 1  F: 0
Length 2  F: 1
(1) Display sample  (2) continue: 1
11001100110011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001100110011
00110011001100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100
Run on another sequence? (0) no (1) yes: 1
Other sequence description
(1) Random, (2) Markov, (3) Distribution of run lengths, (4) File: 2
Number of executes to trace back: 2
FF
Probability followed by T: .9
TF
Probability followed by T: .1
FT
Probability followed by T: .9
TT
Probability followed by T: .1
(1) Display sample  (2) continue: 1
01100110011001001100011001001100110011001100011001100110011001100110011001
10011000110010011000101100110011001100110110011001100110011001100110011001
10011000110011001100110011000110011001100110011110011001110011001100110011
00110110011001001110011001001100100100110011001100100110011001100110011001
10010011011001100100110011001100110011001100110011000110010011001100110011
00110011001100110000110010011001100110011001100110011001100110011001100110
01100100110011001100110011010011
Statistics to compile
Number of bits to use in n bit prediction: 2
Number of executes to trace back for Markov info: 2
Number of trials: 100000
 Sequence statistics
T:  50000 / 100000 = 0.500000
Markov info
FF  24999 / 24999 = 1.000000
TF      0 / 25000 = 0.000000
FT  24999 / 24999 = 1.000000
TT      0 / 25000 = 0.000000
 Stretch distribution
Length     Take  Fallthru
 1          0     0
 2      25000  24999
 3          0     0
 4          0     0
 5          0     0
 6          0     0
 7          0     0
 8          0     0
 9          0     0
10          0     0
11          0     0
12          0     0
13          0     0
14          0     0
```

```
15          0     0
16          0     0
17          0     0
18          0     0
19          0     0
20          0     0
21          0     0
22          0     0
23          0     0
24          0     0
>=25        0     0
25000 take stretches with average length 2.000000
24999 fall through stretches with average length 2.000080
Prediction accuracies
2-bit accuracy: 25001 / 100000 = 0.250010
   State 0      0 / 100000 = 0.000000
   State 1      24999 / 100000 = 0.249990
   State 2      50000 / 100000 = 0.500000
   State 3      25001 / 100000 = 0.250010
Profile accuracy: 50067 / 100000 = 0.500670
Markov accuracy: 90106 / 100000 = 0.901060
Run distribution accuracy: 88037 / 100000 = 0.880370
```

8.2 Simulator

```
 Ran /home/attic/fboas/xsim/src/xsim-hp on espresso - Mon Aug  1 23:31:14 1994
 Total Cycles = 512354457
STATIC counts:
  Instructions = 38322   BasicBlocks = 9788
  Calls = 2720 ( 7.10%)   Loads = 9317 (24.31%)   Stores = 4182 (10.91%)
  NOPs = 4740 ( 1.12x bigger)
DYNAMIC counts:
  Instructions = 440361833   BasicBlocks = 90787444
  Calls = 1781829 ( 0.40%)   Loads = 91365523 (20.75%)   Stores = 9547439 ( 2.17%)
  NOPs = 71992624 (CPI =  1.16)   Tspeedup =  4.36
  NOTE: statistics assume no h/w interlocks

 Block 2486
FF     1828 /    2677 = 0.682854
TF    15775 /   17603 = 0.896154
FT    15826 /   17603 = 0.899051
TT  3410741 / 3426567 = 0.995381
Taken 3444172 / 3464452 = 0.994146
17603 fallthru stretches with an average length of 1.15
17604 take stretches with an average length of 195.65
2-bit scheme: 3442727 / 3464452 = 0.993729
Profile: 3444172 / 3464452 = 0.994146
Markov: 3444172 / 3464452 = 0.994146
Run-length: 3444339 / 3464452 = 0.994194

[Information on other blocks omitted]

 Block 2486
    Length       Take   Fallthru
         1       1777      15775
         2        402       1364
         3        339        263
         4        506         17
         5        493        184
         6        464          0
         7        126          0
         8       292         0
         9       180         0
        10        51         0
        11        67         0
        12       103         0
        13       376         0
        14        55         0
        15         6         0
        16        22         0
        17       116         0
        18       144         0
        19       476         0
        20        50         0
        21        26         0
        22        67         0
        23         0         0
        24         0         0
      >=25     11466         0

[Information on other blocks omitted]

2-bit: 66914532 / 73559777 = 0.909662
Profile prediction: 64403870 / 73559777 = 0.875531
Markov prediction: 68433068 / 73559777 = 0.930306
Run-length prediction: 65611140 / 73559777 = 0.891943
Best-of-each prediction: 68468110 / 73559777 = 0.930782
BRANCH PREDICTION ACCURACIES -
  Profile Data: No previous profile information.
  ALL branches - predict taken = 64.23% correct
  ALL branches - profile prediction = 64.23% correct
  ALL branches - best static prediction = 88.01% correct
  CONDITIONAL branches - predict taken = 62.88% correct
  CONDITIONAL branches - profile prediction = 62.88% correct
  CONDITIONAL branches - best static prediction = 87.55% correct
  CONDITIONAL branches - static corr. pred. (no htree) = 88.34% correct
  CONDITIONAL branches - heuristic static corr. pred. (no htree) = 88.34% correct
  CONDITIONAL branches - hardware corr. pred. = 87.31% correct
```