Performance Evaluation of One and Two-Level Dynamic Branch Prediction Schemes over Comparable Hardware Costs

José Luis Pino and Balraj Singh

EECS Department, University of California, Berkeley, CA 94720
pino@EECS.Berkeley.EDU, bsingh@CS.Berkeley.EDU

ABSTRACT

Branch prediction has become an area of interest due to its effects on the performance of pipelined and superscalar processors. Various methods have been proposed to speculate the path of an instruction stream after a branch. In this paper, the performance of prediction schemes is evaluated by not only the accuracy of prediction but also by the amount of hardware the technique requires to reach that level of accuracy.

We model the configurations which were proposed by the authors of these schemes by allocating an equal number of bytes of memory to each and then mapping (if possible) the various tables needed by the scheme to that amount of memory. The total number of bytes per scheme was varied from 1 byte to 128 kilobytes for each of the different runs. The inputs to the various schemes were traces obtained by running the SPEC-92 benchmarks.

We also compare the finite state machines proposed in [1] to update the history bits in the 2 bit schemes and study the performance of one over the other.

1.0 INTRODUCTION

Most modern processors have a deep pipeline to decrease the average cycle time thus reducing the total execution time. Because of the pipeline structure, a processor decodes an instruction after it has fetched other sequential instructions. When a branch instruction is taken, the pipeline leads to fetches of sequential instructions after the branch. These instructions must be nullified since they are invalid. Since on an average 20 percent of the instructions are branches [5], these incorrect instructions lead to a substantial penalty on the performance of the processor. This effect is more significant in superscalar architectures which may issue more than one instruction per cycle.

To reduce the number of wasted instructions following a branch, processor designers try to move the calculation of the branch target as early in the pipeline as possible. Another technique used to reduce the cost of branch is to speculate on the direction and the target of the branch and move the execution in that direction. If the speculation is incorrect then the processor must nullify the instructions it partially executed in the wrong direction.

Speculation that is done at compile time and based on opcode information in the instruction is known as static branch prediction. These can be as simple as predicting all branches taken which can achieve up to 68 percent accuracy [6] or can include always predicting forward branches not taken and backward branches as taken [7].

When speculation is done at run-time it is known as dynamic branch prediction. This is discussed in further detail in section 2.0 of this paper.

This paper compares the performance of proposed dynamic branch prediction schemes over a broad range of hardware costs. For purposes of this study, the hardware cost is assumed to be the size of the tables for address tags and prediction bits. Thus, the hardware cost model does not account for the complexity of looking up an entry in a branch prediction table. To our knowledge this is the first time a study of this nature has been conducted.

In the following sections we will describe the various dynamic prediction schemes we used for the study. For each scheme, the hardware cost model will be given. Each hardware cost function given will have as one of its arguments the order of the number of bytes of memory available. We also discuss the our simulation methodology and implementation details before presenting our results and observations.

2.0 BRANCH PREDICTION SCHEMES

As described earlier, dynamic prediction schemes speculate on the basis of the run-time behavior of a branch or a set of branches. Schemes such as Decode History Table (DHT) [1], Branch History Table (BHT) [1] and a combination of both DHT/BHT (DHT-BHT) [1] can be classified as one-level dynamic prediction schemes as the prediction depends only on the history of the branch under consideration.

In schemes such as two-level adaptive training branch prediction [2][4] and prediction using branch correlation [3][4] a more complex approach is used in which the prediction is not only based on the past history of the branch under consideration but also the outcomes of recently executed branches in the instruction stream. We classify these as two-level dynamic branch prediction schemes.

We simulated all of the above branch prediction schemes using varied table sizes. The following sub-sections describe each of these schemes individually.

2.1 One-Level Branch Prediction Schemes

One-level branch prediction schemes only consider the history of the branch under consideration when speculating the outcome of the branch.

2.1.1 Decode History Table (DHT)

The Decode history table (DHT) is a direct mapped array of one or two bit entries which are probed to determine the outcome of a branch [1][6][7]. Figure 1 shows a simple implementation of a two-bit DHT.

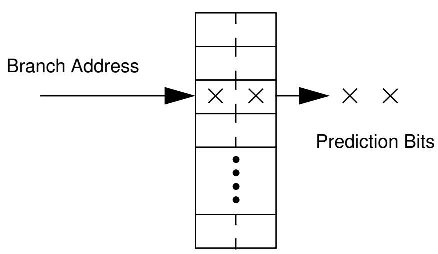

Figure 1. Two-Bit Decode History Table (DHT)

In a DHT many branches may map to the same element. The history bits are updated after the correct outcome of each branch is determined. For the simulations, we tested the DHT using both one and two bit histories. The table size of the DHT is expressed as:

$$tableSize_{one-bit}(order) = 2^{order+3} \quad \textbf{(EQ 1)}$$

$$tableSize_{two-bit}(order) = 2^{order+2} \quad \textbf{(EQ 2)}$$

2.1.2 Branch History Table (BHT)

The Branch history table (BHT) is a fully associative array of entries which stores the branch address as a tag and some history bits to speculate the outcome of the branch [1]. This is a very widely used branch prediction scheme and numerous commercial processors currently use it in their branch prediction units. Figure 2 illustrates a BHT with two bits of history.

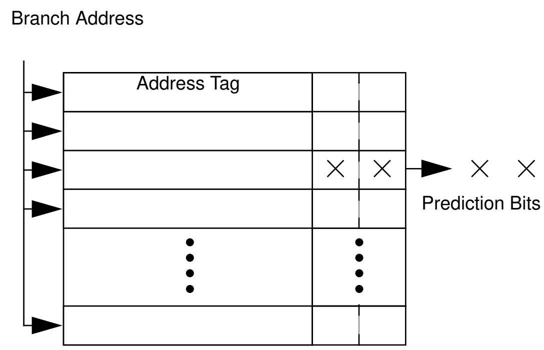

Figure 2. Branch History Table (BHT)

If there is no entry for the current branch in the table, the branch is predicted not taken and an entry is created for the branch. Once the correct outcome of the branch is determined the history bits for the branch are updated.

Since it has to store the branch address and has to do a fully-associative table lookup, the BHT is very costly in terms of hardware. In practice, for each branch instruction address tag there is an associated target address stored to be used in the event of branch. We have not accounted for this cost in our hardware model because it will not affect the prediction performance. This and the combined DHT-BHT techniques are the only branch prediction schemes that we have studied where the target addresses can be stored and retrieved when doing branch prediction. Finally, we implemented a random replacement policy for the BHT due to its simplicity and minimal hardware cost.

When computing the hardware cost for this method, we also do not include the cost of the history bits. The table size is given by (EQ 3).

$$tableSize(order) = 2^{order-2} \quad \textbf{(EQ 3)}$$

2.1.3 Combination DHT-BHT

Based on the DHT and BHT schemes, Kaeli, et al. [1] proposed an improvement by combining the two. Figure 3

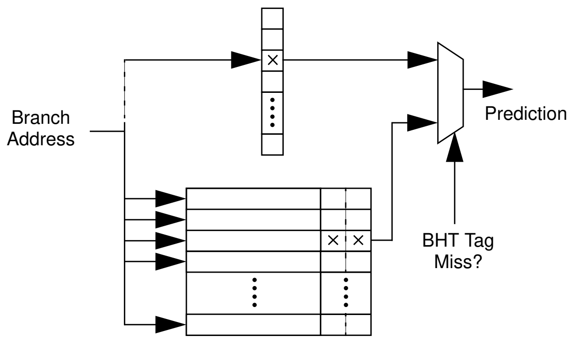

Figure 3. Combined DHT-BHT

gives an example of a combination DHT-BHT predictor. When a BHT misses, the DHT is used to predict the direction of the branch. Furthermore, predictions made by a BHT hit do not update the DHT.

For simplicity, the sizes of both the DHT and BHT in our simulation were set equal to each other. As in section 2.1.2, we do not account for the prediction history bits of the BHT; we also do not account for the cost of the DHT. Thus the function for computing the table sizes of the combined DHT-BHT is equivalent to the BHT which is shown in (EQ 3).

2.2 Two-Level Branch Prediction Schemes

In two-level adaptive training branch prediction and prediction using branch correlation, a two-stage approach is used in which the prediction is not only based on the past history of the branch under consideration but also the outcomes of recently executed branches in the instruction stream.

2.2.1 Correlation Based Prediction (BPT)

Pan, et. al. assert that in many integer workloads, control-flows are complex thus correlating sequential branches in an instruction stream [3]. To take advantage of this observation, they propose correlation based prediction (BPT). Here the prediction not only depends on the branch address but also the path which was taken by the instruction stream to reach the branch. This path is recorded in a global shift register which selects the appropriate set to predict from. Hence the last n sequential branch outcomes select which of the 2^n sub-histories is to be used for branch prediction. A two bit shift register based correlation scheme is shown in figure 4.

Since the shift register must be updated after each branch outcome, speculating across multiple branches in superscalar architectures is not possible. The function for the table size of each sub-history is shown in (EQ 4).

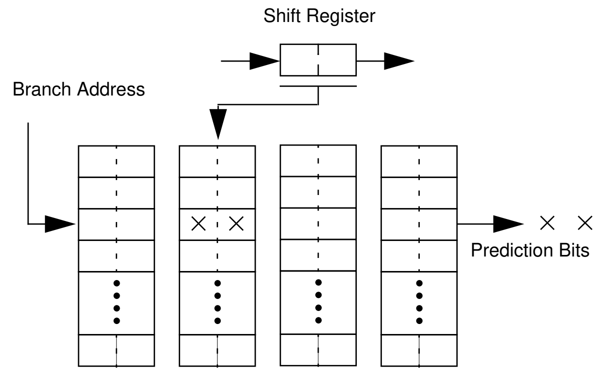

Figure 4. Correlation Based Prediction (BPT)

$$tableSize(order,shiftSize) = 2^{order + 2 - shiftSize} \quad \textbf{(EQ 4)}$$

2.2.2 Two-Level Adaptive Prediction (TLA)

The last scheme that study was proposed by Yeh and Patt in [2]. Their method, two-level adaptive prediction, also uses the concept of correlation. However the prediction is based on the history of successive branches whose address directly maps into a register in the correlation table (see figure 5). The direct-mapped address correlation register is used to address into the global pattern table. The typical sizes of the correlation registers proposed by Yeh and Patt are either six or twelve bits.

This scheme does not have the intuitive appeal which the other schemes do. It is also quite expensive to implement in both hardware and software (for simulation purposes).

The function for the global pattern table size with respect to correlation register size is given in (EQ 5) and the function for the correlation table size is given in (EQ 6).

$$ptSize(crSize) = 2^{crSize} \quad \textbf{(EQ 5)}$$

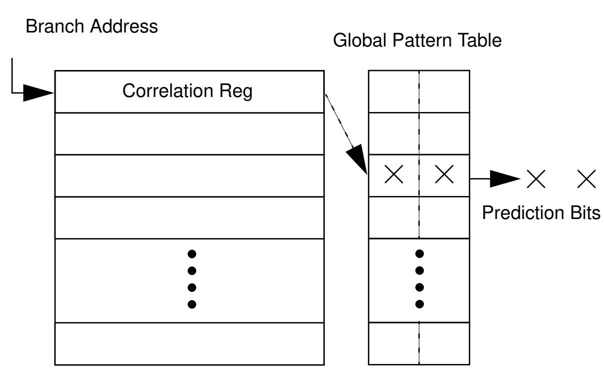

Figure 5. Two-Level Adaptive Prediction (TLA)

$$ctSize(order, crSize) = \left\lfloor \frac{2^{order+3} - 2^{crSize}}{crSize} \right\rfloor \quad \textbf{(EQ 6)}$$

2.3 Finite State Machines

To implement two-bit history schemes, one employs a finite state machine to update the history bits. Yeh and Patt in [2] suggest the finite state machines in figure 6. The second automaton in the most commonly used two-bit saturating up-down counter. In our simulation we compared all four for each of the two-bit history schemes to see if one automaton was clearly better than the others.

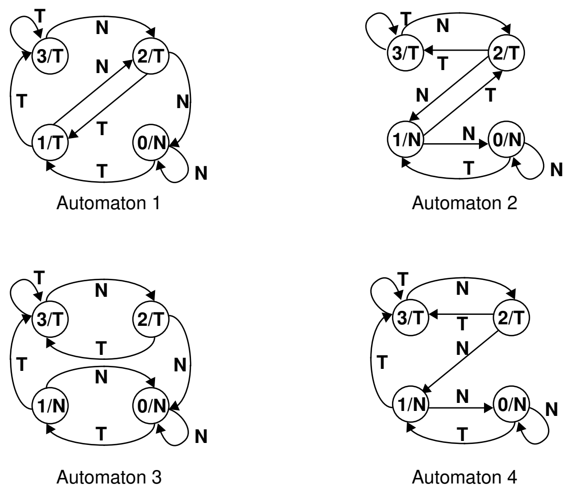

Figure 6. Finite state machines for updating 2-bit branch histories

To read the state transition graphs:

- In the center each is a state number and a prediction value associated with that state. A ‘N’ denotes predict not taken; whereas, a ‘T’ denotes predict taken.
- On a branch outcome, update the state by taking the appropriate ‘N’ or ‘T’ arrow to the next state.

3.0 SIMULATION TECHNIQUES

For purposes of this study, we chose to run trace driven simulation over cycle level simulations. We did this so that we could run real benchmarks through our simulator at a respectable speed. In cycle-level simulation, furthermore we would have to assume a hardware architecture. Thus, our results can be used as a basis for further study of any specific architecture.

We used SPEC-92 integer and floating point benchmarks running on a DEC MIPS as our test suite. Our traces were generated using Pixie. Unfortunately, the pixified gcc benchmark would core dump, so we could not use it in our study. The benchmarks used for the study are shown below.

Floating Point	Integer
doduc	compress
fppp	eqntott
mdljdp	espresso
ora	li
tomcatv	

We also chose to limit each trace to a total of 2,000,000 branches resolved for simulation speed. In figure 7, we compare the prediction ratios for the various schemes between runs to 2,000,000 and 6,000,000. A comparably small amount of error in the same direction is introduced because of this assumption.

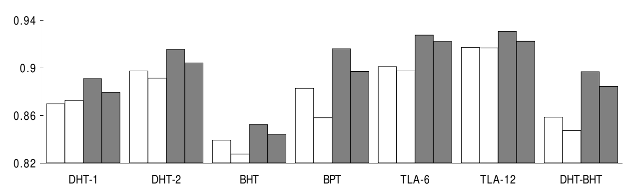

Figure 7. Prediction ratios of runs cutoff at 2,000,000 branches and runs cutoff at 6,000,000 branches. The white bars are for compress and the gray bars are for the eqntott.

4.0 SIMULATION RESULTS

For this and remaining sections, we will use the abbreviations shown below.

Symbol	Scheme
DHT-1	One bit decode history table
DHT-2	Two bit decode history table
BHT	Branch history table
BPT	Correlation based prediction, 2 bit shift register
TLA-6	Two level adaptive, 6 bit correlation register
TLA-12	Two level adaptive, 12 bit correlation register
DHT-BHT	Combined BHT and DHT-1

We found that branch prediction ratios were much higher for the floating point benchmarks. Schemes here achieved up to 98% accuracy for the fppp benchmark. On average,

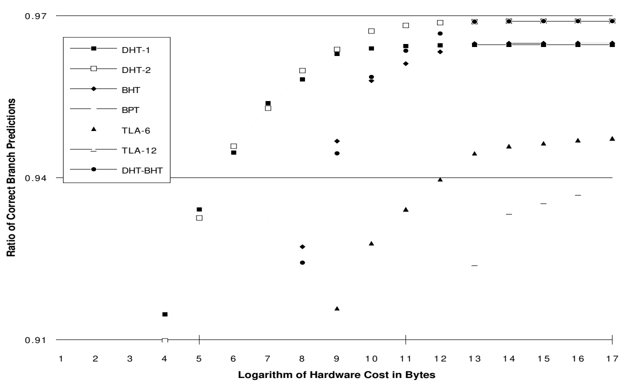

Figure 8. Average Prediction Ratio vs. Hardware Cost for Floating Point SPEC-92 Benchmarks

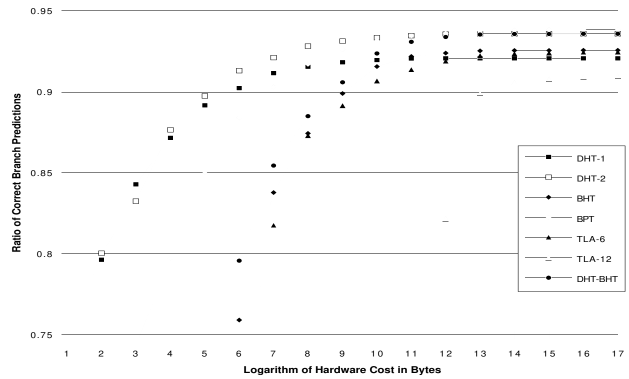

Figure 10. Average Prediction Ratio vs. Hardware Cost for both Integer and Floating Point SPEC-92 Benchmarks

for all the floating point benchmarks the best achieved was 97% by DHT-2, BHT, BPT and DHT-BHT. Figure 8, shows the average prediction ratio across all floating point benchmarks with respect to hardware cost.

However, it is interesting to note that DHT-2 achieves rates of more than 90% accuracy at a mere 16 bytes of total memory usage. At 128 bytes, DHT-2 achieves 95% accuracy. At the hardware limit all but the TLA, achieve comparable results. BPT requires 64 bytes before exceeds 90% accuracy.

The integer benchmarks achieve lower prediction accuracies than the floating point benchmarks. The average prediction ratios achieved over hardware costs is shown in figure 9.

The highest prediction ratio achieved was 94% correct for the li benchmark using BPT. It is interesting to note that the espresso benchmark was particularly difficult to predict and all the benchmarks at the hardware limit only achieved roughly 80% accuracy.

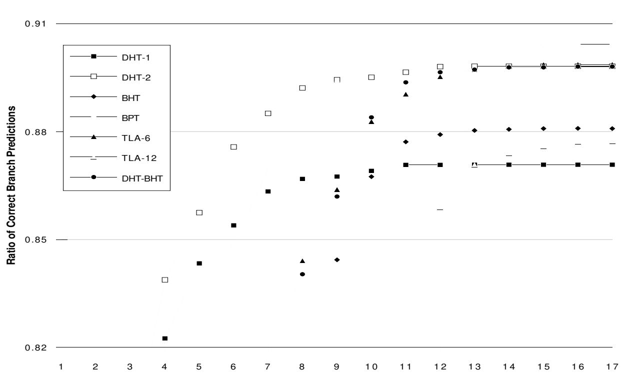

Figure 9. Average Prediction Ratio vs. Hardware Cost for Integer SPEC-92 Benchmarks

Over the integer benchmarks, the highest average achieved prediction rate was about 90%. With BPT being slightly better than the DHT-2, BHT, DHT-BHT and TLA-6. Again, it is interesting to note that DHT-2 performs well with small hardware cost. Using 16 bytes, DHT-2 achieves prediction ratios around 85%; whereas, BPT uses 128 bytes to achieve this.

In figure 10, all the benchmarks are averaged to give a overall prediction ratio for a given hardware cost.

On comparing the finite state machines shown in figure 6, we found that automaton 2 generally gave the best results. The only consistent exception to this were schemes with address tag buffers smaller than 128 entries (ninth order) which get better results with automaton 1.

5.0 PERFORMANCE ANALYSIS

In this section, we list possible sources of error in our assumptions and simulations.

Our simulation does not differentiate between conditional and non-conditional branches. This limitation was introduced because the traces we have consist of basic blocks rather than individual instructions with distinct opcodes.

We only account for the hardware cost in terms of amount of memory used by the prediction tables. We do not account for the hardware needed to access the entries nor the cost of updating the history bits.

Since we wanted this study to be architecture independent, we do not compare branch miss penalties in terms of processor cycles. However, this is an important metric for chip designers to consider before using any of the branch prediction schemes.

After we reviewed our simulation results, we realize that the way we set the combined DHT/BHT table size may not have been the optimal. For simplicity, we assumed that the DHT and BHT components had the same number of table entries. We now feel that a the DHT should have more table entries than the BHT. We believe that the prediction rates will remain lower than the DHT-2 alone. The main advantage of the combined DHT-BHT is that in the BHT we can store the target address and thereby reduce the number of wasted cycles on calculating the target address each time.

6.0 CONCLUSIONS

Given enough hardware, all prediction schemes perform roughly the same. Their relative differences are negligible considering standard deviations of approximately 2.5% from the mean of all benchmarks. The notable exceptions are TLA-6 and TLA-12 whose standard deviation are about 6%.

Our most surprising result is that DHT-2 has roughly 90% prediction accuracies for buffer sizes in the 32 - 64 byte regions.

Due to the high accuracy of the small DHT-2, we believe researchers should focus on small and simple branch prediction schemes. However, as is evident in the literature, researchers have been recently looking at increasingly complex branch prediction techniques. Our results demonstrate that these schemes do not improve prediction significantly and perform much worse for low hardware-cost implementations. Correlation based schemes overcome one level techniques only after they reach their saturation point.

Because of the low hardware-cost implementations possible with DHT, processor designers should investigate into designing prediction units per process rather than per processor. We have not studied the effect of interleaving instruction streams through a branch prediction simulation and further study in this area is warranted.

If target addresses are needed to reduce branch cycle penalties, we recommend combining a DHT-2 with a cache. The cache would store only taken branches using the branch instruction address as a tag and storing the associated target address. This is similar to the combined DHT/BHT proposed in [1] except that it is less complex. Again, in this case, multiple small DHT tables could be introduced to allow for multiple processes.

REFERENCES

[1] D.R. Kaeli, P.G. Emma, J.W. Knight, and T.R. Puzak, "Contrasting instruction-fetch time and instruction-decode time branch prediction mechanisms: achieving synergy through their cooperative operation," Eighteenth EUROMICRO Symposium on Microprocessing and Microprogramming (EUROMICRO 92), vol. 35, Paris, France, 1992, p. 401-8.

[2] T.-Y. Yeh and Y.N. Patt, "Two-level adaptive training branch prediction," Proceedings of the 24th International Symposium on Microarchitecture. MICRO 24, Albuquerque, NM, USA, ACM, 1991, p. 51-61.

[3] S.-T. Pan, K. So, and J.T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), vol. 27, Boston, MA, USA, 1992, p. 76-84

[4] T.-Y. Yeh and Y.N. Patt, "A comparison of dynamic branch predictors that use two levels of branch history," 20th Annual International Symposium on Computer Architecture ISCA '20, vol. 21, San Diego, CA, USA, 1993, p. 257-66.

[5] D.A. Patterson, and J.L. Hennessy, Computer Architecture a Quantitative Approach, Morgan Kaufmann, San Mateo, CA., 1990.

[6] J. Lee, and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, (January 1984), pp. 6-22.

[7] J. E. Smith, "A Study of Branch Prediction Strategies", Proceedings of the 8th International Symposium on Computer Architecture, (May. 1981), pp. 443-58.