A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History

Tse–Yu Yeh and Yale N. Patt
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

Recent attention to speculative execution as a mechanism for increasing performance of single instruction streams has demanded substantially better branch prediction than what has been previously available. We [1, 2] and Pan, So, and Rahmeh [4] have both proposed variations of the same aggressive dynamic branch predictor for handling those needs. We call the basic model Two-Level Adaptive Branch Prediction; Pan, So, and Rahmeh call it Correlation Branch Prediction. In this paper, we adopt the terminology of [2] and show that there are really nine variations of the same basic model. We compare the nine variations with respect to the amount of history information kept. We study the effects of different branch history lengths and pattern history table configurations. Finally, we evaluate the cost effectiveness of the nine variations.

1 Introduction

With the current movement toward deeper pipelines and wider issue rates, extremely high branch prediction accuracy becomes critical because a larger amount of speculative work needs to be thrown away after a branch misprediction. To improve branch prediction, several authors have suggested basing predictions on two levels of branch history information.

Lee and Smith [7] proposed collecting these two levels of history information statically. We [1] introduced the idea of dynamically collecting two levels of branch history, branch execution history and pattern history, to achieve substantially higher accuracy than any other scheme reported in the literature. We call our algorithm *Two-level Adaptive Branch Prediction.* Our predictor adjusts its prediction according to the behavior of the branch instructions at run-time. The first-level branch execution history is the history of the last k branches encountered. The second-level pattern history is the branch behavior for the last j occurrences of the specific pattern of these k branches. Prediction is based on the branch behavior for the last j occurrences of the current branch history pattern. The first-level branch execution history and the second-level pattern history are collected at run-time, eliminating the disadvantages inherent in Lee and Smith's method, that is, the differences in the branch behavior of the profiling data set and the run-time data sets.

In [2] we described three variations of Two-level Adaptive Branch Prediction, differentiating them by the manner in which the first-level of branch history information is kept (G, for global, or P, for per-address) and the manner in which the second-level pattern history tables are associated with this history information (g, for global, or p, for per-address). We suggested that history information can be kept in a single global register, or in separate per-address registers for each address that contains a branch instruction. We further suggested that a single global pattern table could contain the second-level history information, or each address that contains a branch instruction could contain its own second-level per-address pattern table. We identified the three schemes as GAg, PAg, and PAp and showed that PAg is the most cost-effective variation.

Pan, So and Rahmeh [4] proposed a model they called *Correlation Branch Prediction*, because the prediction of a branch depends on the history of other branches. In the terminology introduced in [2], this would be called GAp. They also introduced another variation, which we could label GAs, where the addresses that contain branch instructions are partitioned into subsets, each subset sharing the same second-level pattern table.

This paper describes and characterizes possible variations of the Two-Level Adaptive Branch Prediction model according to the manner in which the first-

level branch history information is kept (G, S, or P) and the manner in which the second-level pattern history tables are associated with this history information (g, s, or p). The variation S means addresses that contain branch instructions are partitioned into sets, each set sharing the same first-level branch history register. This yields nine variations of the model: GAg, GAs, GAp, PAg, PAs, PAp, SAg, SAs, and SAp. They are summarized, along with the reference to where they were first introduced, in Table 1.

Variation & Reference		Description
GAg	[2]	Global Adaptive Branch Prediction using one global pattern history table.
GAs	[4]	Global Adaptive Branch Prediction using per-set pattern history tables.
GAp	[4]	Global Adaptive Branch Prediction using per-address pattern history tables.
PAg	[1]	Per-address Adaptive Branch Prediction using one global pattern history table.
PAs	-	Per-address Adaptive Branch Prediction using per-set pattern history tables.
PAp	[2]	Per-address Adaptive Branch Prediction using per-address pattern history tables.
SAg	-	Per-Set Adaptive Branch Prediction using one global pattern history table.
SAs	-	Per-Set Adaptive Branch Prediction using per-set pattern history tables.
SAp	-	Per-Set Adaptive Branch Prediction using per-address pattern history tables.

Table 1: The Nine Variations of Two-Level Adaptive Branch Prediction

In this paper, we focus on comparing the prediction accuracies of the nine variations of Two-Level Adaptive Branch Prediction by using trace-driven simulations of nine of the ten SPEC89 benchmarks [1]. These variations are studied with respect to various history register lengths, branch history table configurations, and implementation costs.

This paper is organized in five sections. Section 2 describes in brief Two-Level Adaptive Branch Prediction and its nine variations. Section 3 discusses the simulation model and traces used in this study. Section 4 reports the simulation results and our analysis. Section 5 contains some concluding remarks.

[1]The Nasa7 benchmark was not simulated because this benchmark consists of seven independent loops. It takes too long to simulate the branch behavior of these seven kernels, so we omitted these loops.

2 Two-Level Adaptive Branch Prediction and Its Variations

2.1 Concept Summary

Two-Level Adaptive Branch Prediction uses two levels of branch history information to make predictions. The first level is the history of the last k branches encountered. We call the structure which keeps the history of the last k branches encountered the branch history register (BHR). Depending on which variation of the model we implement, the last k branches can mean the actual last k branches encountered (G), the last k occurrences of the same branch instruction (P), or the last k occurrences of the branch instructions from the same set (S). If several BHRs are used to keep branch history, the collection of BHRs is called the branch history table (BHT). If a branch is taken, then the branch history register records a "1"; if it is not taken, the branch history register records a "0".

The second level of the predictor records the branch behavior for the last j occurrences of the specific pattern of the k branches. Prediction is based on the branch behavior for the last j occurrences of the history pattern in question by using an automaton. Our previous study [2] has shown that a 2-bit counter is the most effective automaton among four-state automata. Since the history register has k bits, at most 2^k different patterns appear in the history register. Each of these 2^k patterns has a corresponding entry in what we have called the pattern history table (PHT).

Both branch history and pattern history are updated dynamically When the prediction of a branch is made, the conte .ts of its history register are recorded. After the branch is resolved, the result is used to update the entry indexed by the previously-recorded branch history register contents. The branch history register, on the other hand, is updated with the prediction right after the prediction is made. Since it usually takes several cycles to resolve a branch, updating the branch history speculatively allows the prediction to be made with up-to-date branch history. When an incorrect branch prediction is made, the branch history register should be restored with correct branch history. More details of the Two-Level Adaptive Branch Predictors are contained in [2] and [4].

2.2 Variations

The nine variations of Two-Level Adaptive Branch Prediction (Table 1) can be classified into three classes according to the way the first-level branch history is

collected. These three classes are shown in Figures 1, 2, and 3. They are characterized as follows:

Global History Schemes

In the global history schemes (shown in Figure 1) (also called Correlation Branch Prediction by Pan *et al.* [4]), the first-level branch history means the actual last k branches encountered; therefore, only a single global history register (GHR) is used. The global history register is updated with the results from all the branches. Since the global history register is used by all branch predictions, not only the history of a branch but also the history of other branches influence the prediction of the branch.

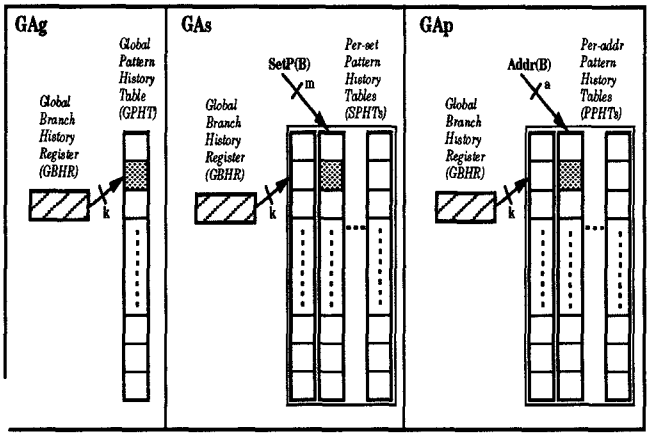

Figure 1: The three variations of global history Two-Level Adaptive Branch Prediction.

Per-address History Schemes

In the per-address history schemes (shown in Figure 2), the first-level branch history refers to the last k occurrences of the same branch instruction; therefore, one history register is associated with each static conditional branch to distinguish the branch history information of each branch. One such history register is called a per-address history register (PBHR). The per-address history registers are contained in a per-address branch history table (PBHT) in which each entry is indexed by the static branch instruction address. In these schemes, only the execution history of the branch itself has an effect on its prediction; therefore, the branch prediction is independent of other branches' execution history.

Per-set History Schemes

In the Per-set history schemes (shown in Figure 3), the first-level branch history means the last k occurrences of the branch instructions from the same subset; therefore, one history register is associated with a

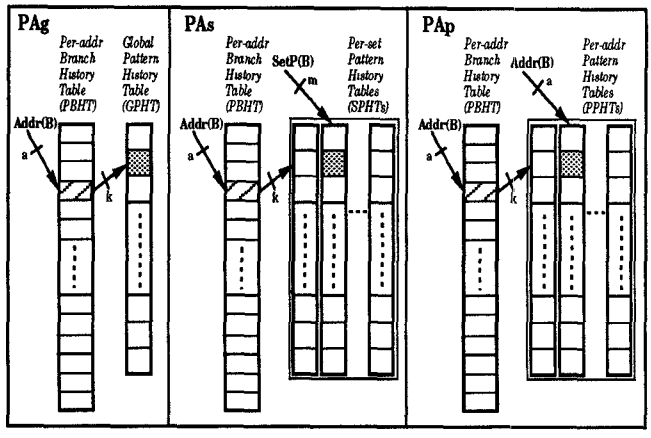

Figure 2: The three variations of per-address history Two-Level Adaptive Branch Prediction.

set of conditional branches. One such history register is called a per-set history register. The set attribute of a branch can be determined by the branch opcode, branch class assigned by a compiler, or branch address. Since the per-set history register is updated with history possibly from all the branches in the same set, the prediction of a branch is influenced by other branches in the same set.

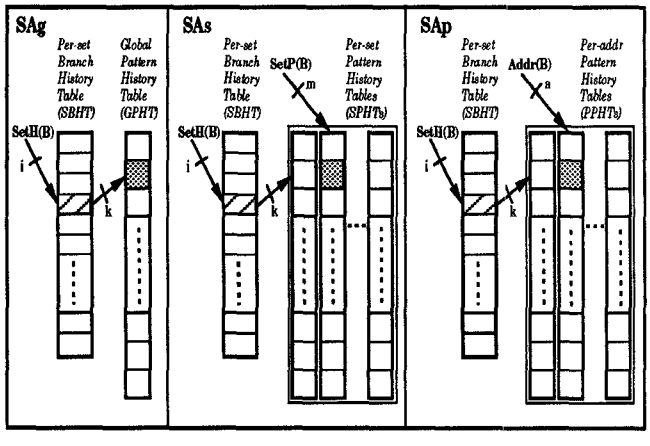

Figure 3: The three variations of per-set history Two-Level Adaptive Branch Prediction.

For each of the above classes, there are three variations when it is further classified by the association of the pattern history tables with branches. The association of the pattern history tables can be one pattern table for each branch (called per-address pattern history table (PPHT)), one pattern table for all the branches (called global pattern history table(GPHT)), or one pattern table for a set of branches (called per-set pattern history table (SPHT)). Indexing into the

multiple pattern history tables is done by using the low-order bits of the branch address, branch opcode, or the branch class passed from a compiler.

3 Simulation Methodology

3.1 Description of Traces

Nine benchmarks from the original SPEC89 benchmark suite are used in this branch prediction study. Table 2 lists the benchmark programs, their abbreviations, and testing data sets used in our simulations. These nine programs were compiled using the Green Hills FORTRAN 1.8.5 compiler or the Diab Data C Rel. 2.4 compiler with all optimizations turned on. The integer benchmarks include *eqntott*, *espresso*, *gcc*, and *li*. Since integer programs tend to have a higher branch frequency, each integer program was simulated for twenty million conditional branches or until the program completed execution. The floating point benchmarks include *doduc*, *fpppp*, *matrix*300, *spice*2*g*6, and *tomcatv*, each one of which was simulated for one hundred million instructions.

Benchmark & Abbreviation		Testing Data Set	Number of Dynamic Conditional Branches
eqntott	eqn	int_pri_3.eqn	20,000,000
espresso	esp	bca	20,000,000
gcc	gcc	dbxout.i	7,326,688
li	li	li-input.lsp	20,000,000
doduc	dod	doducin	7,717,746
fpppp	fpp	natoms	1,139,093
matrix300	mat	Built-in	3,451,457
spice2g6	spi	greycode.in	10,976,668
tomcatv	tom	Built-in	2,926,569

Table 2: Benchmark programs and their data sets.

3.2 Simulation Configurations and Assumptions

Trace-driven simulations were used in this study. A Motorola 88100 instruction level simulator generated instruction traces, which were used as inputs to a branch prediction simulator. The branch prediction simulator decoded instructions, predicted branches, and compared the predictions to the actual outcome to collect statistics of branch prediction accuracy.

A total of nine variations of the Two-Level Adaptive Branch Prediction were simulated with the following methodology:

- Each branch is distinguished by its address. The branch address is used to select a branch history register (BHR) in a branch history table (BHT) and to choose a pattern history table (PHT) in per-address or per-set pattern history tables.
- For GAs and PAs schemes, the low-order bits of the branch address are used as indices to map adjacent branches into different PHTs, so PHT contention is minimized. The low-order bits of the branch address are also used as the index of a BHR in a BHT of limited size to minimize the branch history conflicts between branches.
- For per-set history schemes, we use the branch address to classify branches into sets. The branches in a block of 1K bytes (256 instructions) are members of the same set. Four set history registers are used in our simulations, so that the set index is taken from bits <11:10> of a branch address. These four instruction blocks cover only 4K bytes; therefore, branches from different sets may share the same history register. The index to a pattern table ($SetP(B)$ in Figure 3) is a combination of the set index and the low-order bits of the branch address. We also tried classifying branches by opcode; however, the prediction accuracy is lower than using the branch address.
- Every scheme is configured with a branch history table for storing the instruction fetch addresses [3] of both conditional and unconditional branches. The branch history table is accessed by using the low-order bits of the branch address. For per-address history schemes, each branch history table entry also records branch history. Unless otherwise stated, the default configuration of the branch history table is 1024-entry, 4-way set-associative. The replacement policy of the branch history table is least-recently-used (LRU).
- Two-bit up-down counters are used in all the pattern history table entries for keeping second-level pattern history.
- A pattern history table entry is updated with the information from the trace right after a prediction is made with no delay because the exact branch resolution time is not known in our branch prediction simulator. However, we have shown that various reasonable pipeline delays in the pattern history update have negligible effects on prediction accuracy in [3].
- For the per-address history schemes, a backward taken, forward not-taken scheme is used for making predictions on branch history table misses.

We simplified the hardware cost estimate functions described in [2] and expanded them to all the variations of Two-Level Adaptive Branch Prediction. These functions do not consider the costs for target address fields in the branch history table because all the variations need those fields to store fetch addresses.

Scheme Name	History Register Length	Number of Pattern History Tables	Simplified Hardware Cost
GAg(k)	k	1	$k + 2^k \times 2$
GAs(k,p)	k	p	$k + p \times 2^k \times 2$
GAp(k)	k	b	$k + b \times 2^k \times 2$
PAg(k)	k	1	$b \times k + 2^k \times 2$
PAs(k,p)	k	p	$b \times k + p \times 2^k \times 2$
PAp(k)	k	b	$b \times k + b \times 2^k \times 2$
SAg(k)	k	s×1	$s \times k + s \times 2^k \times 2$
SAs(k,s×p)	k	s×p	$s \times k + s \times p \times 2^k \times 2$
SAp(k)	k	s×b	$s \times k + s \times b \times 2^k \times 2$

b is the number of entries in the BHT and s is the number of branch sets.

Table 3: Conditional branch predictor configurations and their estimated costs.

3.3 Performance Metric

In this paper we use prediction accuracy as the metric to evaluate the performance of branch predictors. The prediction accuracy is the percentage of correctly-predicted conditional branches. The pipeline delays of correct branch predictions on BHT misses are different from the pipeline delays of correct branch predictions on BHT hits. Therefore, prediction accuracy is shown in two major categories: BHT hits and BHT misses. When the branch prediction is correct on a BHT hit, no delay is incurred in instruction fetch after the predicted fetch address is fetched from the BHT. On a BHT miss, even if the conditional branch prediction is correct, the instruction fetch mechanism needs time to decode the branch instruction in order to have the next fetch address if the branch is taken. If the branch is not taken, a prefetch of the next sequential address is assumed. Therefore, there is no penalty for a correctly-predicted fall-through branch on a BHT miss.

When a prediction is incorrect, the processor discards the instructions which are fetched after the branch. Details of the instruction fetch mechanism are contained in [3].

The translation from prediction accuracy to machine performance is not direct [5]. However, higher prediction accuracy means machine pipelines stall less frequently for incorrect branch predictions.

4 Simulation Results

To evaluate the performance of the nine variations of Two-Level Adaptive Branch Prediction schemes, we first study the effects of the branch history length and the number of pattern history tables on their prediction accuracy. Secondly, we show the cost effectiveness of the variations of each class of schemes. Finally, from the schemes with implementation costs of about 8K and about 128K bits, we choose one configuration from each class of the schemes for comparison.

4.1 Effects of Branch History Register Length and Number of Pattern History Tables

4.1.1 Global History Schemes

Figure 4 shows the average prediction accuracy of integer (int) and floating point (fp) programs by using global history schemes with branch history lengths ranging from 2 to 18 bits. Each curve shows the prediction accuracy for a different number of pattern history tables (PHTs). These curves are cut off when the implementation cost exceeds 512K bits.

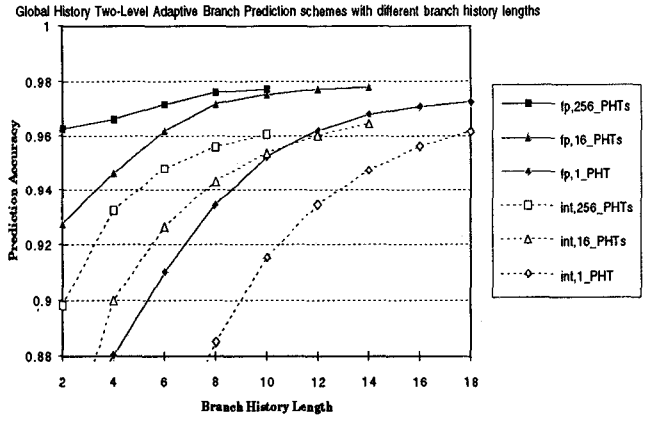

Figure 4: Global history schemes with different branch history lengths.

The performance of the global history schemes is sensitive to the branch history length. This can be seen from the rising trend of the curves. Using one pattern history table, the prediction accuracy of integer programs is still rising when an 18-bit branch history register is used. The prediction accuracy increases about 25 percent by lengthening the branch

history from 2 bits to 18 bits. Even when 16 PHTs are used, the prediction accuracy increases over 10 percent by lengthening the history register from 2 bits to 14 bits. When 256 PHTs are used, the prediction accuracy increases over 6 percent from 90 percent to about 96 percent. The prediction accuracy of floating point programs does not increase as much as integer programs, but is still significant.

Figure 5 shows the average prediction accuracy of integer and floating point programs by using global history schemes with the number of pattern history tables ranging from 1 to 1024. Each curve shows the prediction accuracy for a different branch history length.

The performance of the global history schemes is also sensitive to the number of pattern history tables. As the branch history becomes longer, the increase in the number of pattern history tables results in a smaller increase in accuracy.

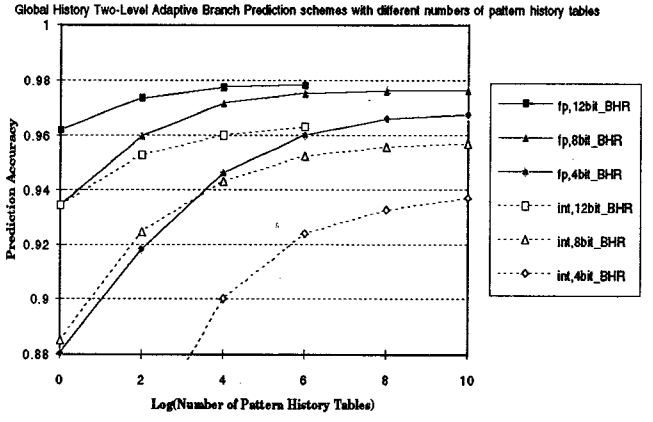

Figure 5: Global history schemes with different number of pattern history tables.

When the global branch history is used, the pattern history of different branches interfere with each other if they map to the same pattern history table. Together, Figures 4 and 5 show that prediction accuracy increases significantly by increasing either the branch history length or the number of pattern history tables. The reasons are that lengthening the global branch history register increases the probability that the history which the current branch depends on remains in the history register, and decreases the possibility of pattern history interference because longer branch history is used. Similarly, adding pattern history tables reduces the possibility of pattern history interference by mapping interfering branches into different tables.

4.1.2 Per-address History Schemes

Figure 6 shows the average prediction accuracy of integer and floating point programs by using per-address history schemes with branch history lengths ranging from 2 to 18 bits. Each curve shows the prediction accuracy for a different number of pattern history tables (PHTs). These curves are cut off when the pattern history table cost exceeds 512K bits.

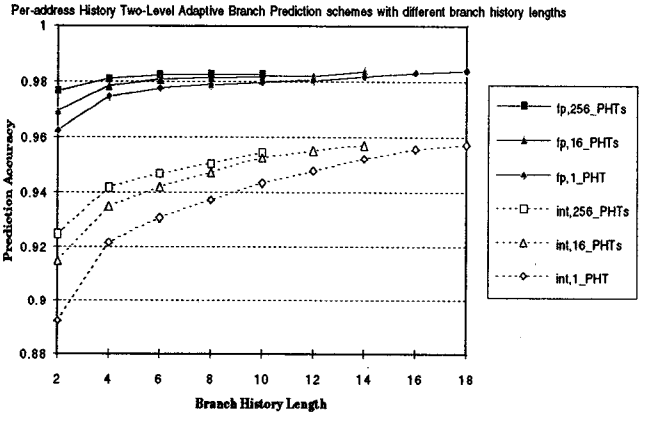

Figure 6: Per-address history schemes with different branch history lengths.

The prediction accuracy of per-address history schemes is not as sensitive to the branch history length as global history schemes. When one pattern history table is used, the average integer prediction accuracy increase is about 6 percent (89.2 percent to 95.7 percent) compared to 25 percent for global history schemes (72.4 percent to 96.1 percent). The curves of average floating point prediction accuracy for per-address history schemes are nearly flat when the branch history length is larger than 6-bit.

Comparing the asymptotic prediction accuracy in Figures 4 and 6, we see that the global history schemes have higher asymptotic averages than the per-address history schemes for integer programs while the per-address history schemes have higher asymptotic average than the global history schemes for floating point programs. This phenomenon is due to the large number of if-then-else statements in integer programs which depend more on the results of adjacent branches; therefore, the global history schemes perform well. The floating point programs, on the other hand, contain more loop-control branches which exhibit periodic branch behavior. This periodic branch behavior is better retained in multiple BHRs, so those branches are more predictable for the per-address history schemes.

Figure 7 shows the average prediction accuracy of integer and floating point programs by using per-address history schemes with the number of pattern history tables ranging from 1 to 1024. Each curve shows the prediction accuracy for a different branch history length.

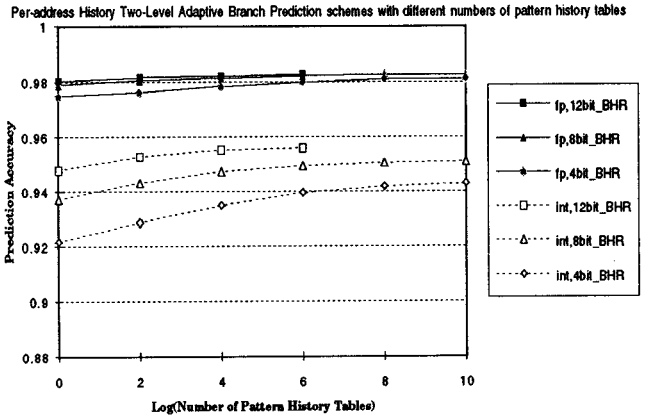

Figure 7: Per-address history schemes with different number of pattern history tables.

When the per-address branch history is used, the pattern history of different branches interfere less with each other when they map to the same pattern history table. As seen from Figure 7, attempting to reduce pattern history interference by increasing the number of pattern history tables results in only a small increase in prediction accuracy. Increasing the branch history length, on the other hand, is more effective in improving the average integer prediction accuracy but not the average floating point prediction accuracy.

4.1.3 Per-set History Schemes

Figure 8 shows the average prediction accuracy of integer and floating point programs by using per-set history schemes with branch history lengths ranging from 2 to 16 bits. Each curve shows the prediction accuracy for a different number of pattern history tables (PHTs). These curves are cut off when the cost of pattern history tables exceeds 512K bits.

By increasing the history register length, the average integer prediction accuracy increases significantly, similar to the behavior of global history schemes. However, the average floating point prediction accuracy does not improve significantly when 4×16 or 4×256 PHTs are used, similar to the behavior of per-address history schemes. The similarity in the behavior of per-address and per-set history schemes is due to the partitioning of branches into sets according to

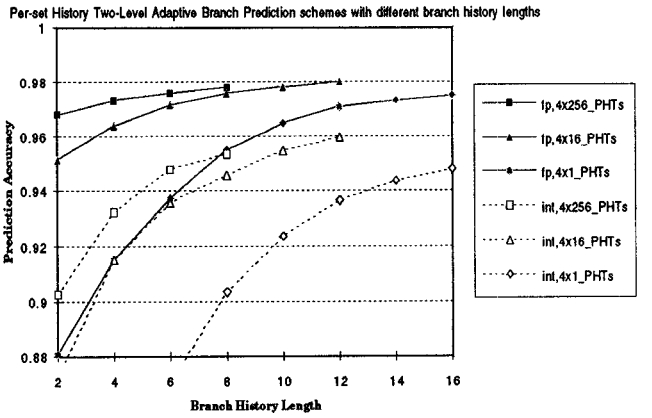

Figure 8: Per-set history schemes with different branch history lengths.

their addresses. The prediction of a branch is dependent on adjacent branches, as in global history schemes. However, the branches involved in the prediction of a branch is limited to the address space a set spans. The instruction block a set covers contains few branches in floating point programs because of their large basic block sizes; therefore, per-set history schemes become similar to per-address history schemes.

Figure 9 shows the average prediction accuracy of integer and floating point programs by using per-set history schemes with the number of pattern history tables in each set ranging from 1 to 1024. Each curve shows the prediction accuracy for a different branch history length.

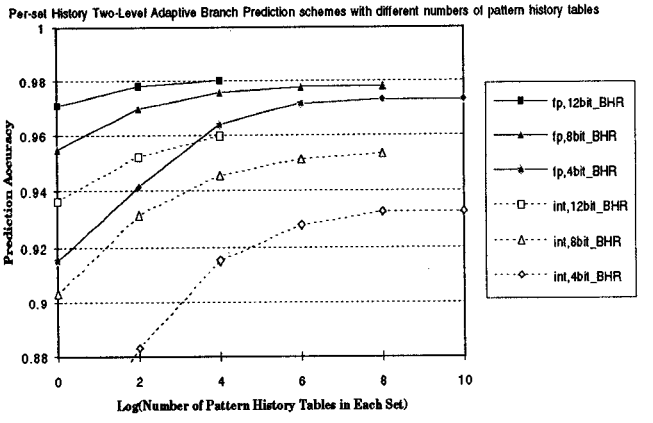

Figure 9: Per-set history schemes with different number of pattern history tables.

The performance improvement of per-set history schemes are less sensitive to the increase in the number of pattern history tables than the global history

schemes. However, it is more sensitive than the per-address history schemes. Since branches of the same set use their own pattern history tables to reduce pattern history interference, increasing the number of pattern history tables is not as effective.

4.2 Cost Effectiveness

4.2.1 Three Classes of Schemes

Figures 10, 11, and 12 illustrate the cost effectiveness of the three classes of Two-Level Adaptive Branch Prediction. Each curve shows the total average prediction accuracy for a different pattern history table implementation cost. The pattern history table implementation cost (in bits) of the schemes on each curve is labelled in the legend. On each curve, as the number of pattern history tables doubles, the branch history length is decremented by one bit. To calculate the total cost of a branch predictor, in addition to the pattern history table cost, a global history scheme needs to add the cost of a global history register, a per-address history scheme needs to add the cost of a branch history table, and a per-set history scheme needs to add the cost of a per-set branch history table.

The per-address history schemes are effective with low implementation costs. Their average prediction accuracy is over 96 percent with 2K bits, about 97 percent with 32K bits, and over 97 percent with 128K bits. The prediction accuracy increases about 1 percent at a cost of an extra 126K bits. On each curve, the average prediction accuracy decreases as the number of pattern history tables increases, which shows that the increasing the number of pattern history tables is less beneficial than increasing the branch history length. The knee is a branch history length of 4 bits.

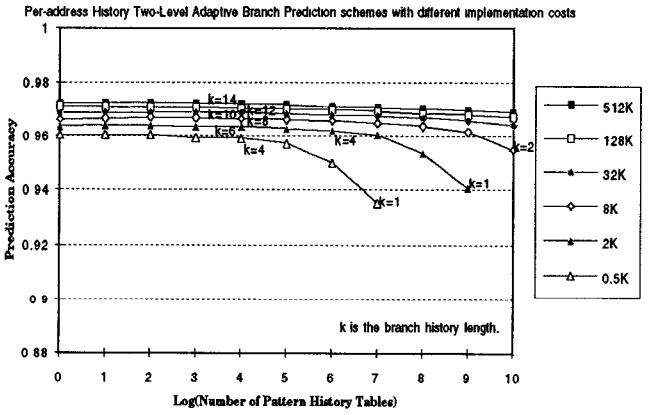

Figure 10: Per-address history schemes with different implementation costs.

The global history schemes require high implementation costs to be effective. Their best average prediction accuracy is only 94.5 percent with 2K bits but over 97 percent with 128K bits. The prediction accuracy increases about 2.5 percent at a cost of an extra 126K bits. By comparing the best prediction accuracy achieved with 512K bits, the global history schemes achieve a higher asymptote than the per-address history schemes. On each curve, as the number of pattern history tables increases, the average prediction accuracy initially increases due to the fact that increasing the number of pattern history tables reduces pattern history interference between branches. As the number of pattern history tables continues to increase, the average prediction accuracy levels off, then decreases because the accuracy gained by increasing the number of pattern history tables is more than offset by the accuracy lost by decreasing the branch history length.

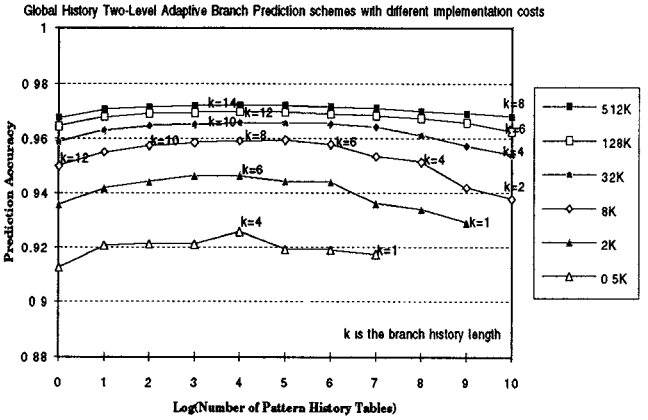

Figure 11: Global history schemes with different implementation costs.

The curves of Per-set history schemes show the same trend as those of global history schemes. However, with the same implementation cost, the per-set history schemes achieve lower prediction accuracy than the global history schemes.

4.2.2 Comparison

Figure 13 compares the cost effectiveness of the three classes of Two-Level Adaptive Branch Prediction given a fixed hardware budget of 8K bits for the costs of both branch history registers and pattern history tables. Each bar graph shows the contributions to prediction accuracy made under four situations: a miss in the BHT which results in the conditional

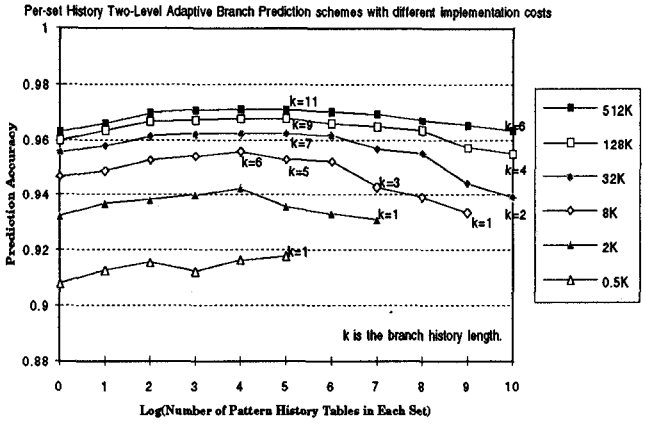

Figure 12: Per-set history schemes with different implementation costs.

branch fall-through (BHT_{miss}, CBR_{ft}), a miss in the BHT which results in the conditional branch taken (BHT_{miss}, CBR_{tk}), a hit in the BHT which results in the conditional branch fall-through (BHT_{hit}, CBR_{ft}), and a hit in the BHT which results in the conditional branch taken (BHT_{hit}, CBR_{tk}).

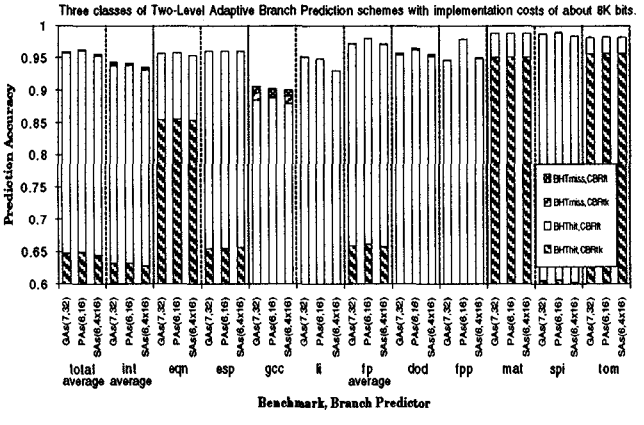

Figure 13: Comparison of the most effective configuration of each class of Two-Level Adaptive Branch Prediction with an implementation cost of 8K bits.

The most cost-effective configuration is chosen from each class. For global history schemes, the most cost-effective configuration is GAs(7,32). For per-address history schemes, the most cost-effective one is PAs(6,16). For per-set history schemes, the most cost-effective one is SAs(6,4×16). All three chosen schemes were simulated with a 1024-entry, 4-way set-associative branch history table.

PAs(6,16) achieves the highest average prediction accuracy among these three configurations. It outperforms the other two configurations on all the benchmarks except for *gcc* and *li*. On *gcc* it suffers from low prediction accuracy on BHT misses; however, it performs better than GAs(7,32) on BHT hits. On *li* it performs almost as well as GAs(7,32). GAs(7,32) achieves the second among these three schemes because of its low prediction accuracy of floating point programs. SAs(6,4×16) is the worst with prediction accuracy about 0.8 percent lower than that of PAs(6,16).

Figure 14 compares the cost effectiveness of the three classes of Two-Level Adaptive Branch Prediction given a higher hardware budget of 128K bits for the costs of both branch history registers and pattern history tables. For global history schemes, the most cost-effective configuration is GAs(13,32). For per-address history schemes, the most cost-effective one is PAs(8,256). For per-set history schemes, the most cost-effective one is SAs(9,4×32). All three chosen schemes were also simulated with a 1024-entry, 4-way set-associative branch history table.

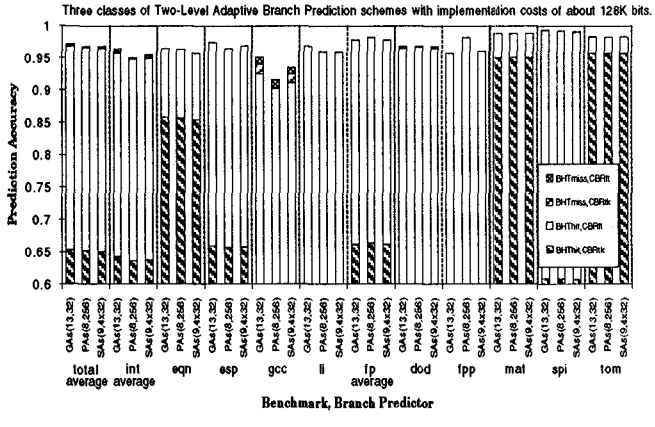

Figure 14: Comparison of the most effective configuration of each class of Two-Level Adaptive Branch Prediction with an implementation cost of 128K bits.

GAs(13,32) achieves the highest prediction accuracy among these three configurations. GAs(13,32) gains the most increase on *gcc* by increasing the history register length at a cost of an extra 120K bits. The increase is substantially better than that gained by PAs(8,256). PAs(8,256) gains little accuracy improvement with the extra bits, whereas SAs(9,4×32) improves its performance to do almost as well as PAs(8,256).

5 Concluding Remarks

We have characterized the global, per-address, and per-set history schemes (GAg, GAs, GAp, PAg, PAs, PAp, SAg, SAs, SAp) and compared them with respect to their branch prediction performance and cost effectiveness.

Global history schemes perform better than other schemes on integer programs but require higher implementation costs to be effective overall. Integer programs contain many if-then-else statements. Global history schemes make effective predictions for if-then-else branches due to their correlation with previous branches. On the other hand, when the global history is used, the pattern history of different branches interfere with each other if they map to the same pattern history table. Therefore, global history schemes require long branch history and/or many pattern history tables to reduce the interference for effective overall performance.

Per-address history schemes perform better than other schemes on floating point programs and require lower implementation costs to be effective overall. Floating point programs contain many frequently-executed loop-control branches which exhibit periodic branch behavior. This periodic behavior is better retained with a per-address branch history table. When the per-address branch history is used, the pattern history of different branches tend to interfere less with each other; therefore, fewer pattern history tables are needed.

Per-set history schemes have performance similar to global history schemes on integer programs; they also have performance similar to per-address history schemes on floating point programs. To be effective, however, per-set history schemes require even higher implementation costs than global history schemes due to the separate pattern history tables of each set.

With respect to the cost-effectiveness of different variations, PAs is the most cost effective among low-cost schemes. If, for example, 8K bits are available to implement the branch predictor, PAs(6,16) outperforms the other variations with an average prediction accuracy of 96.3 percent. However, on *gcc*, GAs(7,32) performs better because the backward taken, forward not-taken default on branch history table misses used in the PAs scheme is not effective. Among high-cost schemes, GAs is the most cost effective. If 128K bits are available to implement the branch predictor, GAs(13,32) achieves the best average prediction accuracy of 97.2 percent.

Acknowledgement

This paper is one result of our ongoing research in high performance computer implementation at the University of Michigan. The support of our industrial partners: Intel, Motorola, NCR, Hal, Hewlett-Packard, and Scientific and Engineering Software is greatly appreciated. In addition, we wish to gratefully acknowledge the other members of our HPS research group for the stimulating environment they provide, and in particular, for their comments and suggestions on this work. We are particularly grateful to Intel and Motorola for technical and financial support, and to NCR for the gift of an NCR 3550, which is a very useful compute server in much of our work.

References

[1] T-Y Yeh and Y.N. Patt, "Two-Level Adaptive Branch Prediction", *Proceedings of the 24th Annual ACM/IEEE International Symposium and Workshop on Microarchitecture*, (Nov. 1991), pp. 51-61.

[2] T-Y Yeh and Y.N. Patt "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proceedings of the 19th International Symposium on Computer Architecture*, (May. 1992), pp. 124-134.

[3] T-Y Yeh and Y.N. Patt "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution," *Proceedings of the 25th Annual ACM/IEEE International Symposium on Computer Microarchitecture*, (Dec. 1992), pp. 129-139.

[4] S-T Pan, K. So, and J.T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, (Oct. 1992), pp. 76-84.

[5] J.A. Fisher and S.M. Freudenberger, "Predicting Conditional Branch Directions From Previous Runs of a Program," *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, (Oct. 1992), pp. 85-95.

[6] J.E. Smith, "A Study of Branch Prediction Strategies", *Proceedings of the 8th International Symposium on Computer Architecture*, (May. 1981), pp.135-148.

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