The Schemes and Performances of Dynamic Branch predictors

Chih-Cheng Cheng

Abstract

The techniques of Instruction Level Parallelism (ILP) and pipeline have been used well to speed up the execution of instructions. The conditional branches are the critical factor to the effectiveness of a deep pipeline since the branch instructions can always break the flow of instructions through the pipeline and result in high execution cost. In order to achieve better CPU performance, many schemes of branch prediction have been utilized. These schemes sometimes can be categorized as program-based predictors vs. profile-based predictors, or static vs. dynamic schemes. This paper focuses on the study of the dynamic branch predictors since the dynamic approach of branch prediction has been developed much more than the static approach of branch prediction. However, their performances always have new and interesting discoveries based on different benchmarks and architectures. I studied as much documentation as I could within my very limited time, and basically perform comparisons between different techniques of dynamic branch prediction, and organize my findings in this paper.

Introduction

Deep pipelining has been one of the more effective ways of improving processor performance. However, when higher degrees of instruction level parallelism have been applied, poorer performance of CPU might occur that is a result from the unresolved branches stalls. For conditional branches, the target instruction cannot be fetched until the target address has been calculated and the conditional branch has been resolved, whereas the unconditional branch is not resolved until the target address is calculated. Whether with conditional or unconditional branches, pipeline stalls could occur once the number of cycles is taken to resolve the branch. The performance of CPU decreases when the pipeline bubbles increase. To solve these problems, predicting the branch direction and providing effective availability of target addresses for execution are two good methods. This paper will mainly focus on the schemes of predicting branch directions. By predicting, pre-fetching, and initiating execution of the branch target address before the branch is resolved, the execution penalty will be substantially reduced.

The static scheme of branch prediction is just predicting whether all branches are taken or not taken. This measurement of performance was reported by Lee and Smith [LS84]. The static strategy can provide up to 68 percent accuracy. In addition, Su and Zhou [SZ95] had more measurements on this strategy and concluded the static predictor has the worst performance. Nevertheless, according to the static scheme, dynamic branch prediction strategies have been studied extensively and proved that it has better performance than static strategy. All of the dynamic predictors have much better performance than static predictors and can at least achieve 90 percent accuracy as reported by McFarling [M93]. Dynamic schemes are different from the static ones in the sense that they use the run-time behavior of branches to make predictions. Hennessy and Patterson [HP96] introduced the two-bit prediction scheme. Moreover, Lee and Smith [LS84] proposed a better structure for it. McFarling [M93] referred to it as the bimodal branch prediction. Yeh and Patt {YP91] discussed these variations. The basic idea is to use 2-bit saturating up-down counters to collect history information that is then used to make predictions. Based on this concept, Yeh and Patt [YP91] developed the two-level adaptive branch prediction scheme. McFarling [M93] even explored more approaches to achieve better accuracy of branch prediction. Due to the fast progress of computer technology, Su and Zhou [SZ95] showed different aspects of performance analysis.

The following sections first introduce those well-known schemes of dynamic branch predictors. After knowing the schemes, each branch prediction performance is then explicitly presented through the comparison chart. Their performance analyses are made in the same section. Before concluding this paper, I present different aspects of performances running on updated benchmarks. The conclusion can then be based more accurately on these various experimental data.

Schemes of Dynamic Branch prediction

➢ Two-bit Counter Branch Prediction Scheme

The two-bit counter scheme always assigns a two-bit counter to the prediction cache buffer for each entry when each conditional branch occurs. The counter value is between 0 and 3. The counter is incremented when branch is predicted as taken; otherwise the counter is decremented if the prediction is not taken. Therefore, a prediction must be wrong twice before it is changed. The most significant bit determines the taken decision of prediction. Hennessy and Patterson [HP96] noted three-bit or higher counters do not make much significant than two-bit counter does. Lee and Smith [LS84] also refer to it as two-bit saturating up-down counters. The buffer is responsible for collecting history information, and applies the information to making the prediction. The state of branch's entry in the buffer is therefore changed dynamically when the branch instructions are executed. The states in a two-bit prediction scheme are illustrated in the figure 1.

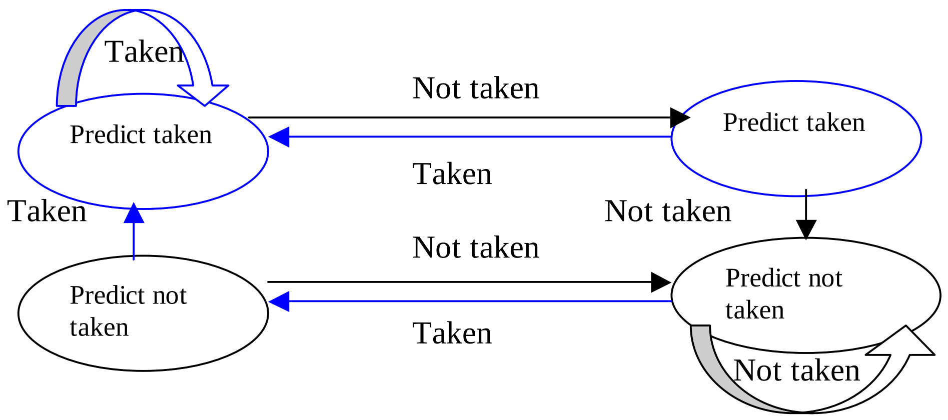

Figure 1. The states in a two-bit prediction scheme

➢ **Bimodal Branch Prediction Scheme**

Bimodal scheme takes advantage of two-bit counter branch behavior. The state of counters is stored in a counter table that records all the branches history. Each branch will then map to a unique counter. The branch history table is indexed by some bits of branch address. The mapping will not be any problem if the counter table is large enough for all branches, says more than 128K bytes; but for the smaller tables, the performance of accurate prediction can be impeded by sharing same counter with multiple branches. However, the smaller size of predictor still performs more effective than the two-bit scheme. The bimodal predictor structure is shown in figure 2.

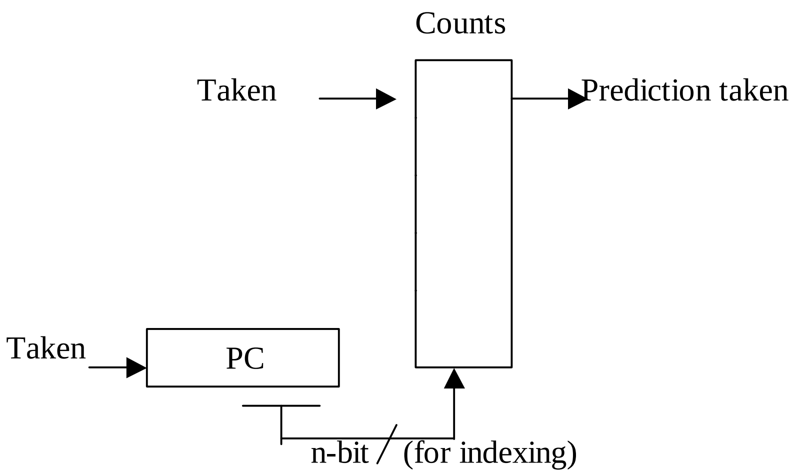

Figure 2. Bimodal Predictor Structure

➢ Correlated Branch Prediction Scheme

One of solutions for the problem of bimodal branch on the mapping collisions is the scheme of correlated prediction schemes. This correlated scheme uses two branch history tables, one is for keeping the recent branches history records and the other one is for keeping the state of branches in each entry contained 2-bit counter. They are so-called history table and counter table. In other word, the correlated scheme takes advantage of relationship between different branch instructions that is certain repetitive branch pattern of several consecutive branches. The structure is illustrated in figure 3. The local, global, and global selection branch predictors are all considered as correlated branch prediction scheme.

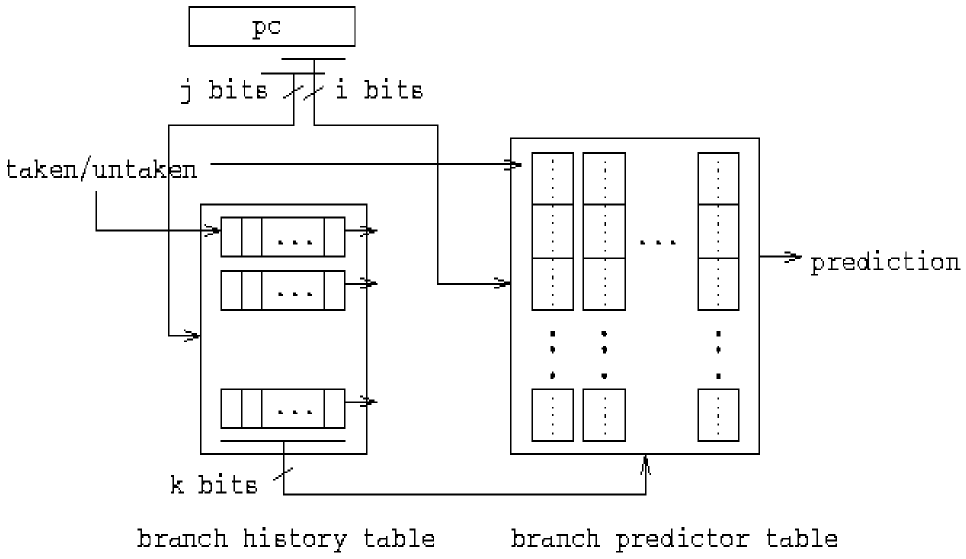

Figure 3. Correlated Branch Prediction Scheme

➢ Local Branch Prediction Scheme

The local branch prediction scheme is one of correlated schemes. Its first table records the history of branches by n-bit shift register. Pan, So and Rahmen [PSR92] used 2-bit shift register for exploiting the correlation between two consecutive branches. McFarling took advantage of this approach at more flexible way. In order to avoid the same history records, he expanded the number of bits according to the higher degree branch. For instance, the four times loops will be confused with the 3 times loop if the number of bits is only limited at three. Both records would the same: (110). However, the expression will be very clear when the number of bit is expended up to four bits: (1110) and (0110). These history records are able to indicate the branch direction. The first table is indexed by the few bits of PC of branch. Yeh and Patt [YP93] considered per_address as the method of using the lower order bits of PC, whereas considered per_set as the method of using high or middle range bits of PC. Likewise, the second table, counter table, is indexed by the content of first table. Figure 4 shows the structure of local predictor scheme.

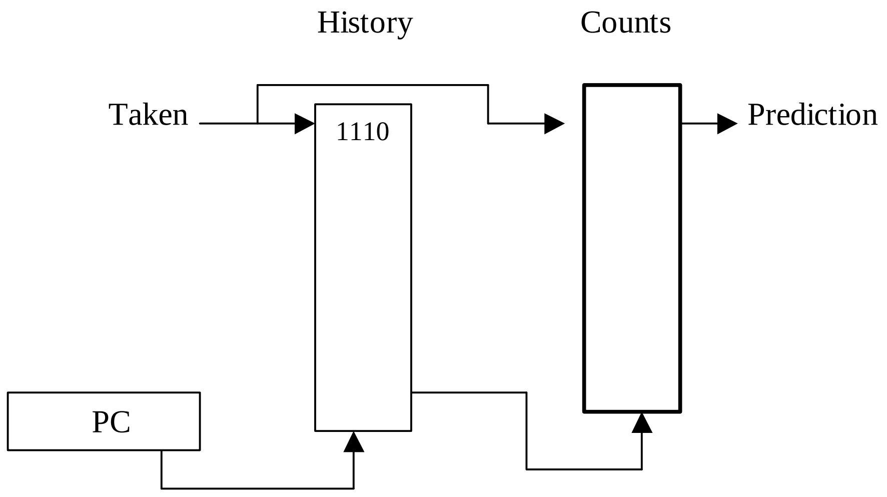

Figure 4. Local History Predictor Structure

➢ Global Branch Prediction Scheme

The global branch prediction scheme takes advantage of other recent branches to make a prediction and make the history table as one single shift register, GR, to record the direction taken by the most recent n conditional branches. Because of the existing of GR, global prediction can be as accuracy as local does. Moreover, single global register substitutes the history table can cut down the overhead of excessive contention for history entries since most of branches are go to the same direction. On the other hand, global register has more difficulty to identify the current branch address than the local predictor does since global register does not keep all records of branch addresses. The structure of global branch prediction scheme illustrates in figure 5.

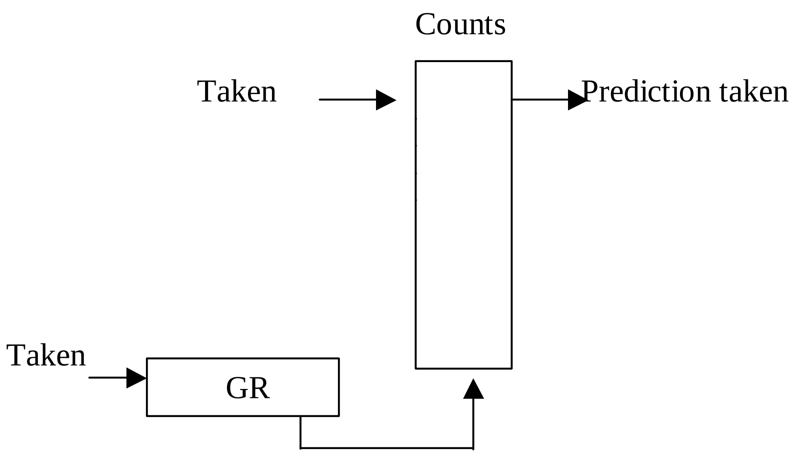

Figure 5. Global Predictor Structure

➢ Global Selection Branch Prediction Scheme

This scheme is an improved version of global scheme by taking advantage of both branch address and global history since global prediction is bad at identifying the current branch than the branch address does. This concept was revealed by Pan, So and Rahmeh [PRS92]. The key idea for the global predictor with index selection scheme is to concatenate the bits of global history and branch address. McFarling [M93] refer to it as gselect. This approach is shown in figure 6 and table 1.

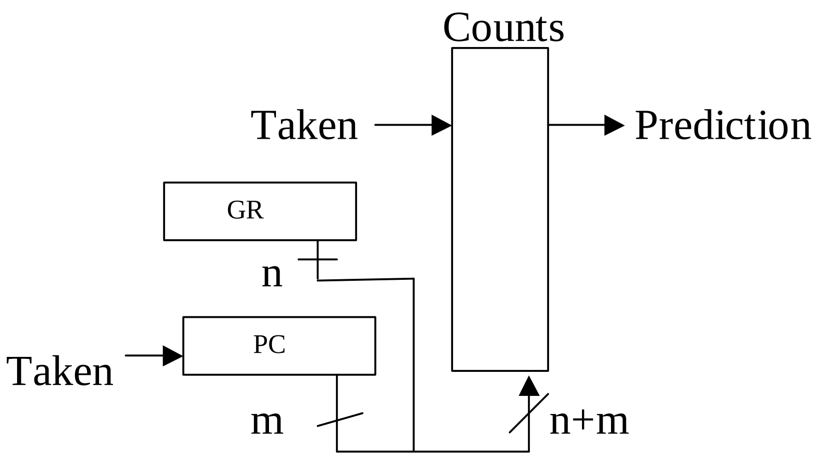

Figure 6. Global History Predictor with Index Selection

Branch Address	Global History	Gselect 4/4 (AND)
0000 0000	0000 0001	0000 0001
0000 0000	0000 0000	0000 0000
1111 1111	0000 0000	1111 0000
1111 1111	1000 0000	1111 0000

Table 1. The Bits Concatenation on Gselect Scheme

➢ Global Sharing Branch Prediction Scheme

Even though gselect predictor can solve the problem, to some extend, that global predictor has hard time to identify branch address well, global selection scheme sometimes still can not distinguish the branches well as the table 1 shows. Global sharing, or gshare as McFarling refereed [M93], predictor extends the strategy of gselect branch prediction scheme. Two parts were modified in gshare scheme from the gselect branch prediction scheme. The first change is the way of combining branch address bits and history bits. Gshare predictor uses exclusive OR (XOR) to combine those two types of bits instead of using concatenation. This change is able to make the index more explicitly as table 2 shows. The second significant change is to allow more branch address bits to be applied in this combination so that gshare predictor can identify the current branch address easily. The structure of gshare predictor is similar to gselect predictor except the combining strategy. The structure of global sharing branch predictor scheme is shown in figure 7.

Branch Address	Global History	Gselect 4/4 (AND)	Gshare 8/8 (XOR)
0000 0000	0000 0001	0000 0001	0000 0001
0000 0000	0000 0000	0000 0000	0000 0000
1111 1111	0000 0000	1111 0000	1111 1111
1111 1111	1000 0000	1111 0000	0111 1111

Table 2. The bits XOR on the Gshare predictor

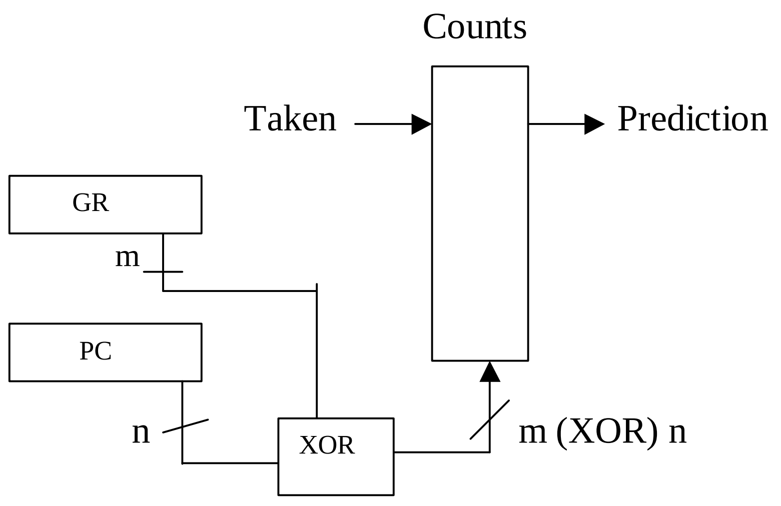

Figure 7. Global History with Index Sharing Scheme

➢ Selective Branch Predictors Scheme

As we mentioned before, one predictor performs excellent under some particular situations, and the others will perform well at other kind of situations. Such as bimodal predictor always performs well when the predictor size is small, whereas local predictor can not see the significant improvement until the counter table grows to some sizes. Based on this observation, McFarling [M93] proposed a combining branch prediction scheme to take advantage of different predictors at vary conditions. The technique is to use two-bit up/down saturating counters to keep track of which predictor is more accurate for the branches that share that counter. The structure is shown in figure 8. The performance and its comparison will be discussed in the later section. On the other hand, the implementation cost of the selective branch predictor is three times of others since two predictors and one selector are used as noted by Su and Zhou [SZ95].

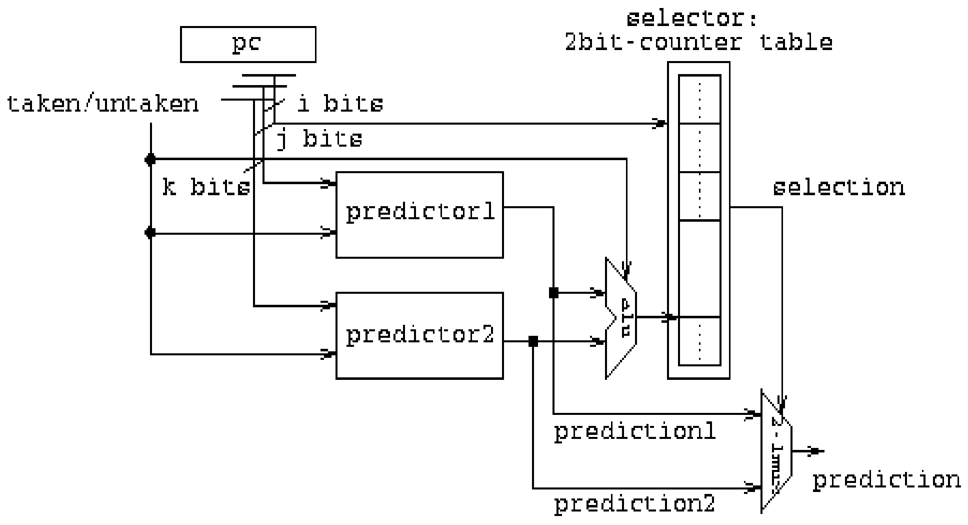

Figure 8. The Structure of Selective Brach Predictor Scheme

The Schemes Performance and Analysis

By the comparison of predictor performance we can analyze their performances explicitly. The measurements for each predictor are based on the SPEC'89 benchmarks and the executions were traced on a DEC station 5000 using the pixie tracing facility. After introduced predictors' individual performance, we will see how the different architecture impacts those branch predictors as experimented by Su and Zhou [SZ95]. The predictors were tested on the SPARC system 10 with two SuperSparc/60 V8 microprocessors and the data were obtained by Shade tracer version 5.15. The measurements were based on the benchmarks of SPECint95-beta and SPECfp92.

❖ The predictors performance on SPEC'89

➢ Bimodal predictor verses Local predictor

As the figure 9 shows, the prediction accuracy always increases with predictors' size since the counters of bimodal predictor are shared by fewer branches as the number of counters increases. However, the accuracy of bimodal predictor can not compete with local predictor once the predictor size greater than 128K bytes and just saturates at 93.5 percent once each branch maps to a unique counter.

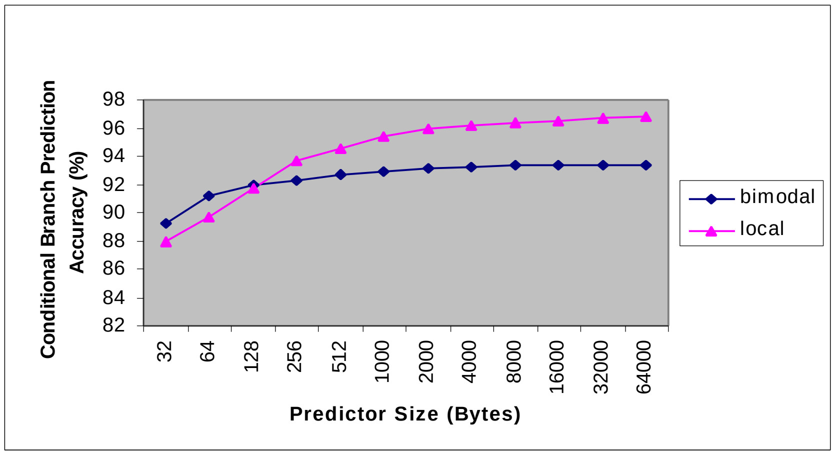

Figure 9. Local Predictor verses Bimodal Predictor

➢ Global predictor verses Bimodal and Local predictors

Here is an interesting finding that global predictor is significantly less effective than the local predictor even though global predictor improves the disadvantage of accessing two tables. This is because global history register can not distinguish different branches as effective as branches address does. However, global history register is able to identify current branch easily once the predictor size is sufficiently large. This is the reason why global predictor performs better than bimodal does when the predictor size is greater than 1K bytes.

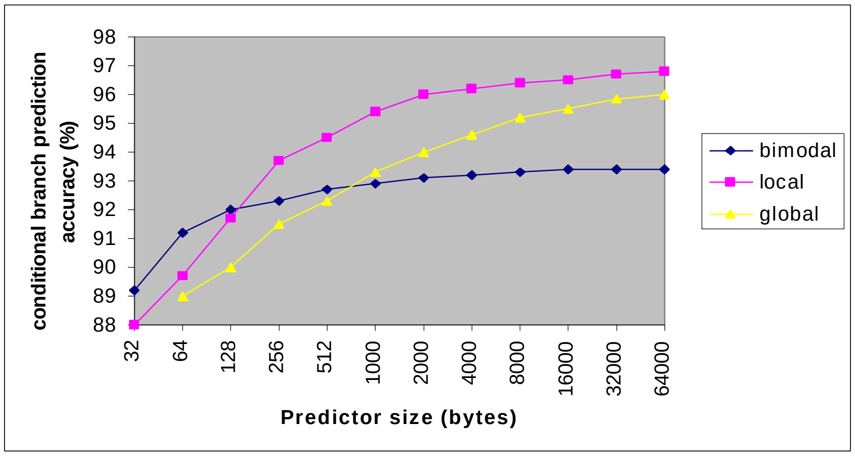

Figure 10. Global Predictor verses Local and Bimodal Predictors

➢ Gselect predictor verses Bimodal, Local and Global predictors

When gselect predictor size sufficiently large to hold enough address bit to identify current branch, the gselect has significant better performance than bimodal predictor does even though they were paralleled when the predictor size is still small. Based on this advantage, gselect predictor can outperforms any other predictors. However, its performance is very close to local predictor since local predictor identifies the current branch solely by the address bits. Nevertheless, gselect still has two advantages to outperform local predictor: the storage space required for global schemes is negligible and gselect only requires single array access whereas local predictor requires two array accesses in sequence.

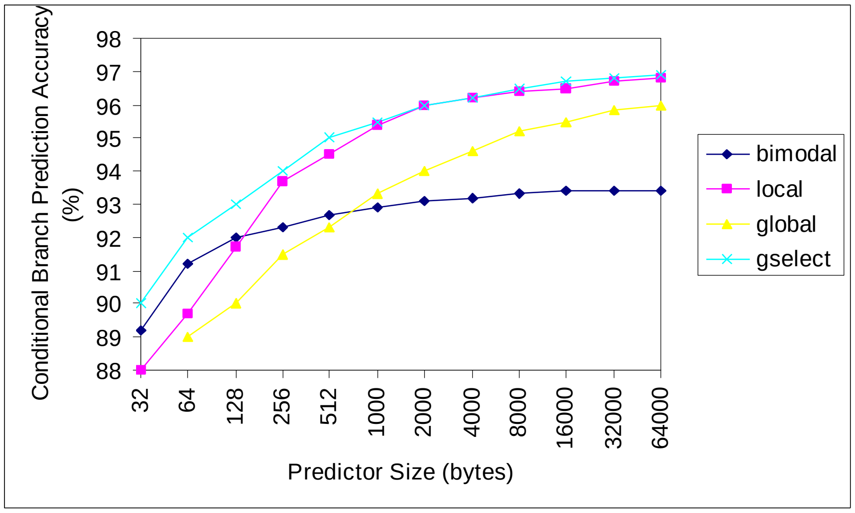

Figure 11. Gselect predictor verses Bimodal, Local and Global Predictors

➢ Gshare predictor verses Global and Gselect predictors

Due to the lack of address bits to identify current branch, the global predictor is much under performs than the other global predictors. On the other hand, gshare can not have better performance until the predictor grows large enough to eliminate the contention for counters between different branches.

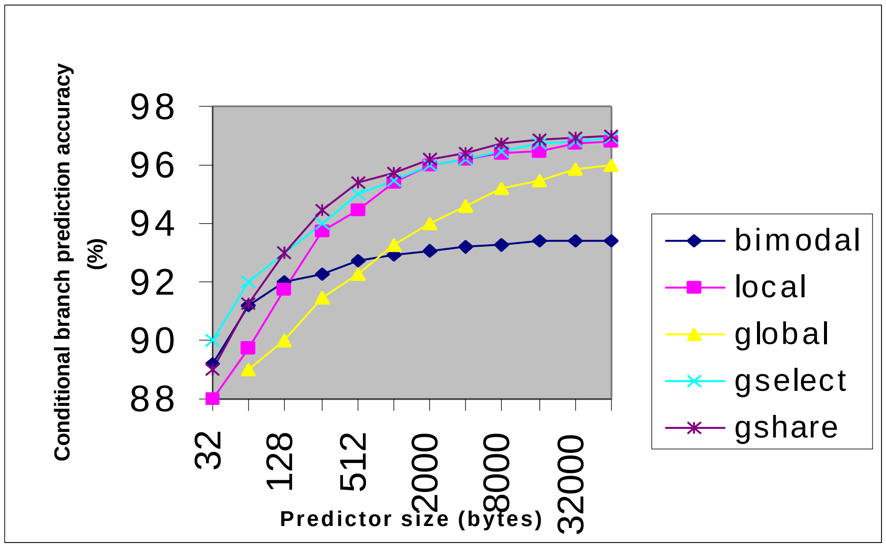

Figure 12. Gshare Predictor verses Other Predictors

➢ Combining predictor verses Bimodal and Gselect predictors

The combining predictor has slightly better performance than gselect predictor does since the combining predictor selection array cost is amortized over more predictor counters. The 1K bytes combining predictor has nearly the same performance as the 16K bytes gselect predictor does.

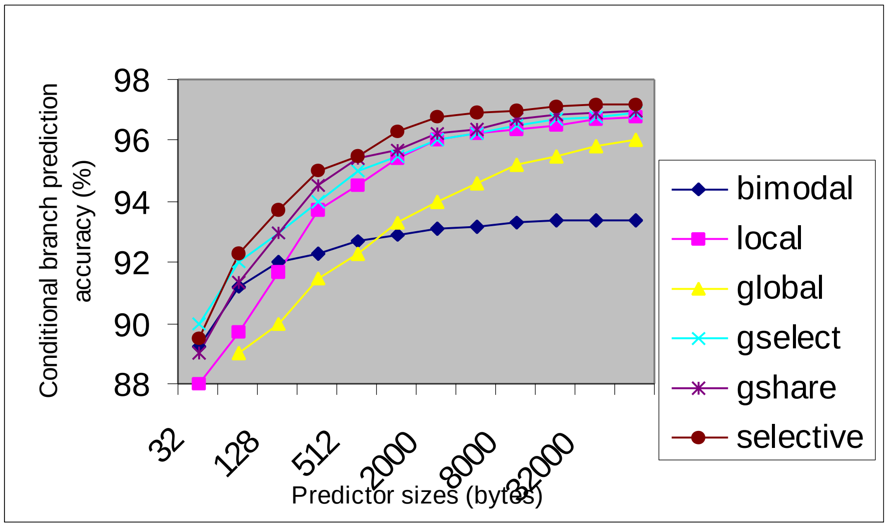

Figure 13. Selective Predictor verses All Other Predictors

❖ The predictors performance on SPECint95-beta and SPECfp92

In this section, we are going to see different aspects of predictors at different analysis. The analysis focuses on evaluating the performances of vary predictors on each program of benchmarks. Except the most of predictors we introduced, the comparison also includes the static and common-correlated branch predictors. The analysis also examines the effect of context switching and the effect of varying the buffer size on branch prediction. The programs of benchmarks are eight integer programs from SPECint95-beta, and thirteen floating-point programs from SPECfp92.

➢ Branches analysis on benchmark programs

Figure 14 shows the frequencies of conditional branches on each program starting with eight integer programs. The blue lines represent integer programs, red lines represent floating point programs, the pink line represents taken instructions, black line and green line represent average branch and taken respectively.

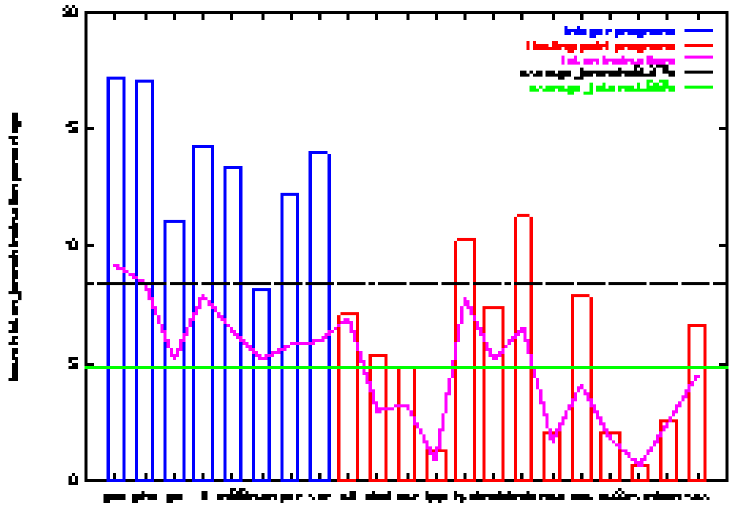

Figure 14. The Frequency of conditional Branches

The floating point programs have lower percentage of conditional branches than integer programs but have higher percentage of taken conditional branches as shows in figure 14 and figure 15. This is because those floating point programs have many long looping structures. On the other hand, an integer program, numi, has lower frequency of conditional branches than the average and highest percentage, almost reaches the average frequency, of conditional branches that are taken among integer programs. This is because numi is the only one written in Fortran, the others are written in C, as well has the highest floating point instruction percentage, 10.2%.

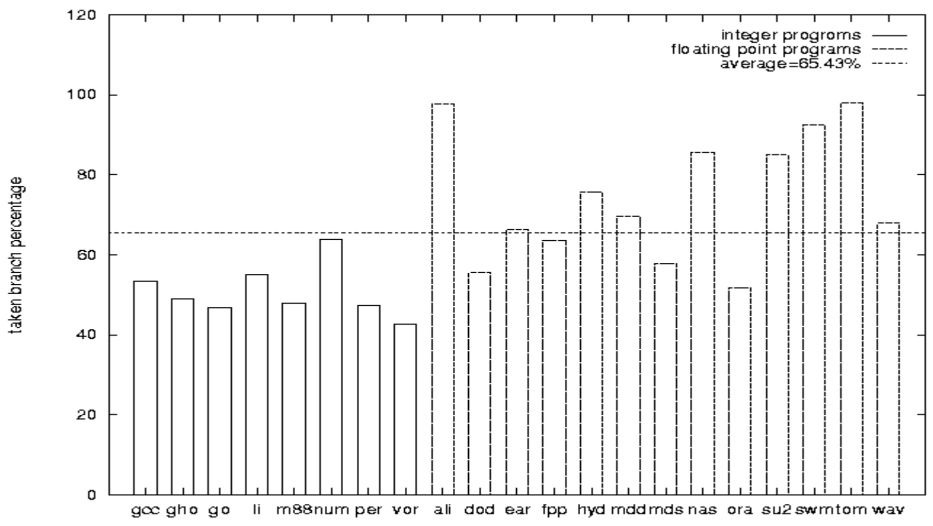

Figure 15. Percentage of conditional branches that are taken

➢ Schemes analysis on benchmark programs

Figure 16 shows all seven predictors' performance over twenty-one benchmark programs. Almost every predictor has the consistent performance on all programs. The differences between their prediction accuracy are somewhat predictable. This suggests that the branch behavior of a program is the most important parameter in determining the prediction accuracy of each scheme.

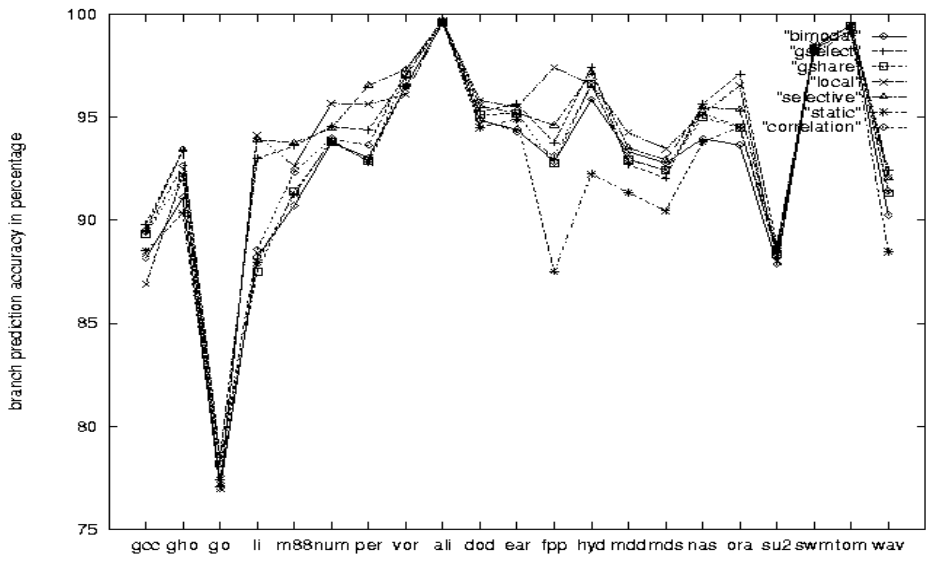

Figure 16. Seven Schemes performance on 21 benchmark programs

We can see the strong correlation between the branches taken and accuracy through the comparison of figure 15 and 16. The interesting finding between these two figures is that the higher the taken frequency the higher the prediction accuracy. This suggest that floating point programs are easier to predict the branch than the integer programs do in terms of the integer programs' low frequency of branches taken. An integer program, go, is the only exception from this correlation since it features many small loops and lots of control flows. This feature causes all the branch predictors that take advantage of long looping structures to perform poorly.

The table 3 summaries the relative performance of the seven schemes. The local predictor has the best performance in floating point benchmark programs since floating point programs have many looping structures that local predictor can take advantage of history register to keep tracking each branch address and reduce the interference among different branches. However, the integer programs contains a lot of if-then-else statements that degrades the performance of local predictor, whereas a scheme with a single array access, gselect, performs better than the local predictor does by taking advantage of the control flow statements.

Benchmarks	**Schemes ordered by performance (from worst to best)**					
INT	***bimodal***	***gshare***	***correlation***	***local***	***gselect***	***selective***
	89.8%	**90.3%**	**90.8%**	**91.3%**	**91.8%**	**92.7%**
FP	***bimodal***	***correlation***	***gshare***	***gselect***	***selective***	***local***
	94.4%	**94.7%**	**94.7%**	**95.3%**	**95.5%**	**95.6%**
ALL	***bimodal***	***gshare***	***correlation***	***local***	***gselect***	***selective***
	92.6%	**93.0%**	**93.2%**	**93.9%**	**93.9%**	**94.4%**

Table 3. Performance Summary of Schemes

Table 3 and figure 16 indicate two differences from other studies. The first differences is that gshare predictor didn't make significant improvement of performance by the feature of reducing aliases. The second difference is stated as follow: with the same size of other branch prediction tables, the combining (selective) predictor can still perform better than other schemes. On the other hand, selective predictor still is the winner over other predictors in this experiment. This indicates combining predictors as another kind of new predictor can get the best accuracy of branch prediction.

Conclusion Remark

- *Identifying current branch plays the key role to make the branch predictor performance more effective.*
  I notice that the local predictor always performs better even though it has the disadvantage of having to access two table arrays in terms of the all global version predictors. Using more branch address bits does increase the predictor performance. None of the correlated predictors can outperform the local predictors by much unless the gselect predictor adapts more branch address bits.

- *Current predictor can perform better by just combining the other predictor.*
  Combining different predictors as a new predictor can impressively improve the accuracy of branch prediction. The combined predictors using local and global branch information reach a prediction accuracy of 98.1% as compared to 97.1% for the previously most accurate known scheme. The experiment

indicates selective scheme achieves the least miss-predict rate with the same size of branch prediction buffer.

- *Tracing less tested programs or a small part of a program may provide misleading data and imprecise conclusion.*
  By importing different benchmarks to run based on a bigger portion of a program and other programs proves selective predictor can still outperform over other schemes using the same size of branch prediction tables.

- *Different structures of branch prediction schemes perform well on different branch structures.*
  Local scheme of branch prediction performs especially well on the branch structure with many loops in a program since the register history table helps local predictor to distinguish the branches. However, its performance can not compete with the gselect predictor when the branch structure in a program contains more control flow statements since accessing the two table arrays degrades performance when the branch structure just consists of if-then-else statements.

References

[BL93] T. Ball and J. R. Larus. “Branch prediction for free. In Proceedings of the ACM SIGPLAN ’93 Conference on Programming Language Design and Implementation,” Albuquerque, NM, 1993.

[FF92] J. A. Fisher and S. M. Freudenberger. “Predicting conditional branch directions from previous runs of a program,” In Proceedings of ASPLOS V, pages 85-95, Boston, MA, October 1992.

[HP96] J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach, 2nd Edition," *Morgan Kaufmann Publishers, Inc.*, 1996.

[JW89] N. P. Jouppi and D. W. Wall. “Available instruction-level parallelism for superscalar and superpipelined machines.” In Proceedings of ASPLOS III, pages 272-282, Boston, MA, April 1989.

[LS84] J. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", *Computer 17:1* Jan. 1984.

[M93] S. McFarling, " Combining Branch Predictors," *TR, Digital Western*

*Research Laboratory,*Jun. 1993

[MH86] S. MaFarling and J. Hennessy "Reducing the Cost of Branches", *Proc. of 13th Annual Intl. Symp. on Computer Architecture,* Jun. 1986.

[PSR92] S. Pan, K. So, and J. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," *Proc. 5th Annual Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems*, Oct. 1992.

[SZ95] Zhendong Su and Min Zhou, “A Comparative Analysis of Branch Prediction Schemes,” University of California at Berkeley, Computer Architecture Project, 1995 .

[YCS96] Gloy C. Young J. B. Chen and M. D. Smith, “An Analysis of Dynamic Branch Prediction Schemes on System Workloads,” Proc. 23rd Annual Intl. Symp. on Computer Architecture, pp. 12--21, May 1996.

[YP91] T. Yeh and Y. Patt, "Two-Level Adaptive Training Branch Prediction," *Proc. 24th Annual ACM/IEEE Intl. Symp. and Workshop on Microarchitecture,* Nov. 1991.

[YS94] C. Young and M. Smith, " Improving the Accuracy of Static Branch Prediction Using Branch Correlation", *Proc. 6th Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems,* October 1994.

[YS95] C. Young and M. Smith, " A Comparative Analysis of Schemes for Correlated Branch Prediction", *Proc. 22nd Annual Intl. Symp. on Computer Architecture,* June 1995.