Improving Branch Prediction Performance with A Generalized Design for Dynamic Branch Predictors

Wei-Ming Lin, Ramu Madhavaram and An-Yi Yang
Department of Electrical Engineering
University of Texas at San Antonio
San Antonio, TX 78249-0669, USA
WeiMing.Lin@utsa.edu

Keywords: Branch Prediction, Two-Level Predictor, Gshare, Generalized Branch Predictor

Received: January 1, 2004

Pipeling delays from conditional branches are major obstacles to achieving a high performance CPU. Precise branch prediction is required to overcome this performance limitation imposed on high performance architecture and is the key to many techniques for enhancing and exploiting Instruction-Level Parallelism (ILP). A generalized branch predictor is proposed in this paper. This predictor is a general case of most of the predictors used nowadays, including One-Level Predictor, Two-level predictor, Gshare, and all their close and distant variations. Exact pros and cons of different predictors are clearly analyzed under the same general format. The concept in the traditional Gshare predictor is then extended to form a more flexible predictor under the same construct. By following this generalized design scheme, we are able to fine-tune various composing parameters to reach an optimal predictor and even allow the predictor to adjust according to various types of applications. From our simulation results, it is evident that significant improvement over traditional predictors is achieved without incurring any additional hardware.

Povzetek: Članek obravnava delovanje CPU in napovedovanje razmejitev.

1 Introduction

In the past decade, by taking advantage of RISC architecture and advanced VLSI technology, computer designers were able to exploit more Instruction-Level Parallelism (ILP) by using deeper pipelines, wider issue rates and superscalar techniques. However, these techniques suffer from disruption caused by branches during the issue of instructions to functional units. How to appease such a performance-degrading effect from branch instructions, which typically make up twenty or more percentage of an instruction stream, has to be paid with more attention.

Branch prediction is a common technique used to overcome this performance limitation imposed on high performance architectures and is the key to many techniques for enhancing ILP. Branch prediction essentially involves a guess on the likely stream direction that is to take place after a branch instruction; whenever such a guess is correct, penalty in pipeline delay is either reduced or completely avoided. There have been various branch prediction schemes proposed in this area [1, 2, 6, 7, 8, 10, 13, 15, 21]. They are usually classified as static or dynamic according to how prediction is made. Static prediction schemes always assume same outcome for any given branch, whereas a dynamic scheme uses run-time behavior of branches to adjust the database for later predictions. Focus of this paper is on the dynamic ones which usually show far better prediction accuracy than the static ones.

A typical dynamic prediction mechanism relies on a prediction table, or the so-called Pattern History Table (PHT) to record the behavior of past branches. One of the very early predictors used was the One-Level Predictor which has a one-dimensional PHT and uses only program counter (PC) as the index to retrieve past branch behavior and record new branch behavior. Usually one-bit or two-bit saturating up-down counters are used in the PHT to record the behavior of branches. When these branches are encountered again they are predicted based on their entries in the table, i.e., based on their previous behavior. It was followed by the more widely used Two-Level Adaptive Predictor [17], which has the PHT but organized into a two-dimensional table. Such a PHT is addressed using both the PC index and a history register (HR) index. The HR is used to record behaviors of either the most recent per-address branch or the most recent global branches. The two-level predictor easily outperforms the one-level one by exploiting potential correlation between branches at run-time.

Another very popular predictor design, Gshare, was proposed in [9]. Unlike the previously proposed designs, Gshare addresses the PHT with a blended index between global history and the PC. Based on Gshare, some other designs including LGshare [4] have also been proposed. Although these Gshare-based designs usually yields better prediction results than the traditional two-level predictors in most cases, the exact reason behind their design and their relationship with the two-level predictors has never been discussed, nor has a complete analysis on their benefits ever been presented.

After carefully analyzing the structure of all the aforementioned predictors, we propose a general prediction scheme which encompasses all these popular designs. This generalized scheme displays a standard organization for the PHT and a flexible selection for HR index. This proposed scheme provides a platform with which one can easily understand the similarities and/or differences among different predictors proposed so far. In addition, based on this, we can provide a more systematic explanation for the benefits a predictor provides and the drawbacks it may come with.

The remainder of the paper is organized as follows. A brief overview of the well-known counter-based dynamic branch prediction schemes is presented in section 2. The proposed generalized predictor is then described in the following section along with its variations. In section 4, our simulation and performance comparison results are presented. Concluding remarks are given in the last section.

2 Dynamic Branch Prediction

There have been many dynamic branch prediction schemes proposed in the past decade. A few representative ones are described in the following for the sake of completeness.

2.1 One-Level Predictor

The prediction table is usually indexed by the lower-order address bits in the program counter (PC), although other portions of the PC have been used as well. Figure 1 illustrates the design of such a scheme. Each entry in the prediction table (PHT) is used to provide prediction information for the branch instruction mapped to it, and is implemented by a counter which goes up or down according to the actual outcome of the corresponding branch instruction. Each branch is predicted based on its most recent outcome. Instead of the simple one-bit counter, a well-known two-bit up-down counter has been extensively used in this scheme so as to render a damping effect which enhances prediction accuracy for typical reentrant loop constructs. Damage caused by alternating occurrences between two aliasing branches can also be alleviated using the two-bit counters. Such an observation prompts most later advanced designs to use such a two-bit counter prediction table as a design base.

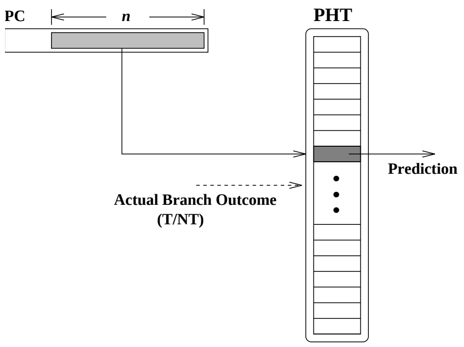


Figure 1: Implementation of Simple One-Level Branch Prediction

2.2 Correlation-based or Two-Level Adaptive Predictor

Outcome of a branch is usually affected by some previously executed branches. Such a correlation could exist among different branch instructions executed temporally close to one and other, or simply refers to the effect on a branch from its own recent execution behavior. The latter one has been partially considered in the simple one-level two-bit counter design. Such an approach requires a separate table, the so-called history table, to record the necessary history information. A general design block diagram is shown in Figure 2 in which the PHT organized as a two-dimensional table is addressed by two separate indices, the PC index and the history index. History information established in a history table can be either in per-address (per-branch) format as shown in Figure 2 or in global format as shown in Figure 3. In a per-address case, a per-address history table is needed which also is addressed by the PC index. A shift register, so-called History Register (HR), is usually used to implement each such entry. On the other hand, for the global format, only one HR is needed, as shown in Figure 3. This aims at exploiting the correlation in behavior existing in most programs between recurring identical branches (as in the per-address case) or between distinct branches adjacent in time (as in the global case). HR index and PC index combined are then used to locate the counter in the PHT for prediction. It is shown that [19] global history schemes perform well with integer programs while per-address history schemes are better for floating point programs. Also, note that the PC index for history table does not have to come from the same least significant portion of PC that the PC index for PHT normally uses. The so-called "per-address" refers to the one that uses the least significant bits of PC for such an index, while the "per-set" refers to the selections otherwise. In general, such

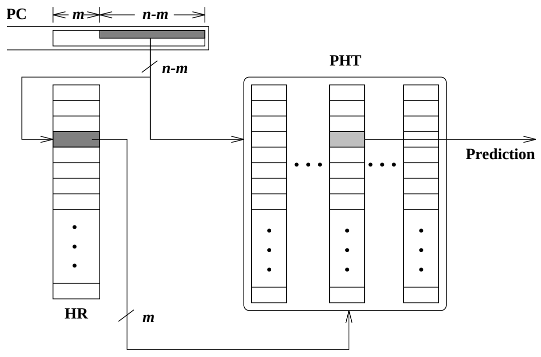


Figure 2: Implementation of Per-Address Correlation-based Branch Prediction

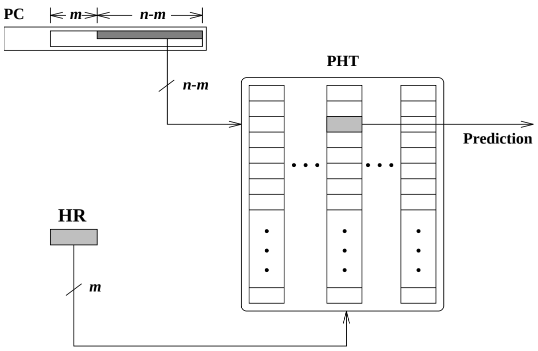


Figure 3: Implementation of Gloabl Correlation-based Branch Prediction

a selection does not lead to any significant discrepancy in performance.

2.3 Gshare Predictor

In the Gshare scheme [9], as shown in Figure 4, the prediction table is addressed by an index established by XORing a global history and part of the PC index. Gshare scheme does lead to improvement in most cases compared to a simple two-level predictor; however, the exact cause for such an improvement has never been clearly analyzed. (Note that, in one of the original Gshare designs, the XORing function is performed over the entire PC index; that is, m is set to be equal to n, which in general leads to worse performance than a simple two-level predictor.)

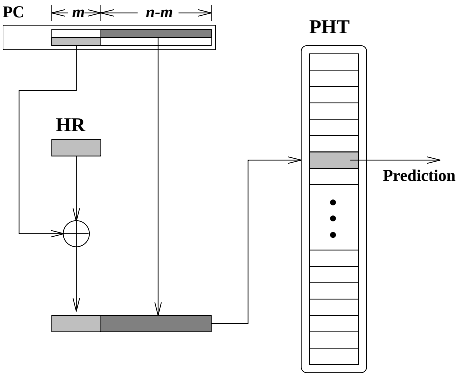


Figure 4: Implementation of Sharing Index Branch (*Gshare*) Prediction

2.4 Others Well-Known Predictors

The possibility of combining different branch predictors is exploited by McFarling in [9]. It comes from the observation that some schemes work well on one type of programs while not so on another. The selective scheme is implemented with two different predictors, with each making prediction separately. A third table is then used to make decision between the two prediction outcomes based on various program scenarios. Such a scheme is claimed to perform well on different circumstances, yet it has a hardware cost roughly three times of what a non-selective one would cost. A predictor called LGshare has also been proposed [4] to further improve on Gshare by using both global as well as per-address history of a branch to predict its behavior. Among many more others in this field, a new predictor discussed in [6] is based on Simultaneous Subordinate MicroThreading (SSMT), which provides a new means to improve branch prediction accuracy. SSMT machines run multiple concurrent microthreads in support of the primary thread to dynamically construct microthreads that can speculatively and accurately pre-compute branch outcomes along frequently mispredicted paths. Another technique is introduced in [5] to reduce the pattern history table interference by dynamically identifying some easily predictable branches and inhibiting the pattern history table update for these branches.

In general, there are a few types of well-known potential problems that would lead to a misprediction result due to the nature of the predictor employed:

- Initialization –
  Every branch instruction that has a predictable behavior needs to have its behavior history properly established in the prediction table before a meaningful prediction can be made.

- Alias –
  This problem occurs when different branches are mapped to the same entry in the PHT. Such a problem is unavoidable unless a sufficiently large number of entries to cover all potential program sizes are provided.

- Undetected Correlation –
  Due to the limited size of history register, correlation among branches far apart in time/trace may not be detected.

- “Random” (Unpredictable) Branch Behavior –
  A branch’s behavior, either at times or throughout the life of the program, may be simply run-time data-dependent which is either completely “random” or unpredictable based on any of the known branch prediction schemes.

Some of the above problems may further intertwine with each other. For example, if the overall size of the predictor table is to remain the same, by increasing the history depth (the history register size) to allow more potential correlation to be detected, alias problem between different branches would worsen. It is part of our goals in our proposed generalized predictor to determine the pros and cons among the various predictors and to incorporate an additional flexibility in our predictor to accommodate for various programs that may call for different prediction techniques.

3 Proposed Generalized Predictor

A generalized predictor is proposed in this section to show that most of the predictors mentioned above fall under this category. With the introduction of this design scheme, potential performance-influencing factors can be more clearly analyzed. Additional design flexibility is also incorporated in this design to allow adjustment of certain design parameters to accommodate for various program behaviors. In order to have a fair comparison among all predictors, all are assumed to be of unified cost, i.e. predictors with similar hardware are compared.

3.1 Generalized Predictor

Figure 5 illustrates the proposed generalized predictor design. In this design, the PHT is organized as a two-dimensional table similar to the traditional two-level predictor. The primary hardware cost resides in the PHT, the size of which is thus fixed at 2^n. The lower portion of PC from bit 0 to bit $n - m - 1$ is used to address the PHT as the row index, while the column index is composed by XORing the m-bit HR and a “floating” portion of PC. This portion of PC is called as the “PC XOR mask” throughout this paper.

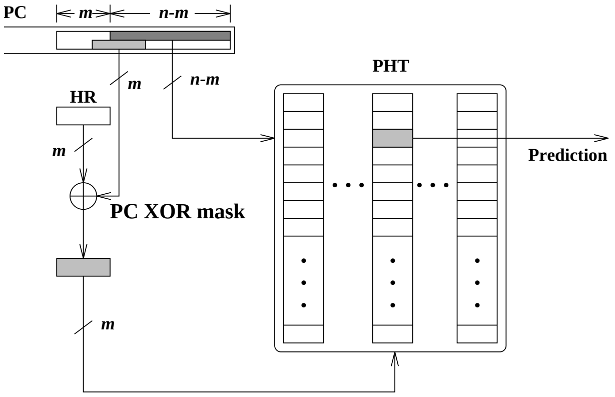


Figure 5: The Proposed Generalized Predictor

3.2 Special Case #1: One-Level Predictor

The one-level predictor as shown in Figure 1 can be reorganized as a special case of the proposed generalized predictor. Figure 6 illustrates such an arrangement. By having the PC XOR mask fixed at the highest position of the n-bit index, and XOR-ing it with the non-existing HR (0’s throughout the HR content), the original one-dimensional PHT is then re-arranged as a two-dimensional PHT. The sequence of addressing in the original one-dimensional PHT is then mapped to a column-major-order sequence in this new two-dimensional PHT, i.e. one column followed by the next one. Such a rearrangement presents us a platform for a direct comparison between any advancement from this technique and the original one.

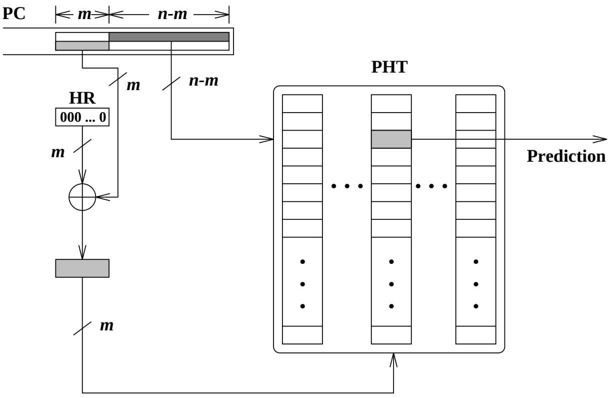


Figure 6: One-Level Predictor as A Special Case

3.3 Special Case #2: Two-Level Predictor

Comparing the two-level predictor with the one-level Predictor, one can obviously see that the former intends to improve prediction accuracy by using more history to exploit correlation information among different branches’ behavior. However, with the cost fixed, the two-level predictor introduces potentially more alias problem into the picture by having a smaller PC index (row index) used. That is, for every additional bit of HR employed (m being increased by 1), the number of row entries of the PHT is reduced by half. It has been shown that such a tradeoff usually is worthwhile to a certain extent of m due to the following reasons:

– Benefit from exploiting correlation among branches usually outweighs the potential performance loss from the alias problems thus incurred.

– Alias problem between two branches thus incurred can sometimes be relieved if they have a different global history pattern in HR even though they are “aliased” into the same row entry. In this case, the alias problem is removed since their prediction entries are mapped into different columns albeit in the same row.

The two-level predictor based on a global history HR as shown in Figure 3 can be also reorganized as a special case of the proposed generalized predictor. Figure 7 illustrates such an arrangement. The new arrangement has the XOR mask confined to within the row index, i.e. the first $n - m$ bits of PC. This results in no change of prediction accuracy because the mapping of branches is merely swapped around among the columns i.e. branches mapped to one column are now mapped to a different column and this takes place symmetrically for all the branches. This comes from a simple understanding of how XOR function applies to a given bit pattern. For example, in Figure 8, two-bit column indices from a two-bit HR are one-to-one mapped to different set of indices when XORed with a different XOR-mask values from PC. Obviously, if this mask portion of PC is within the row index portion of PC as indicated in this special case for two-level predictor, then each branch

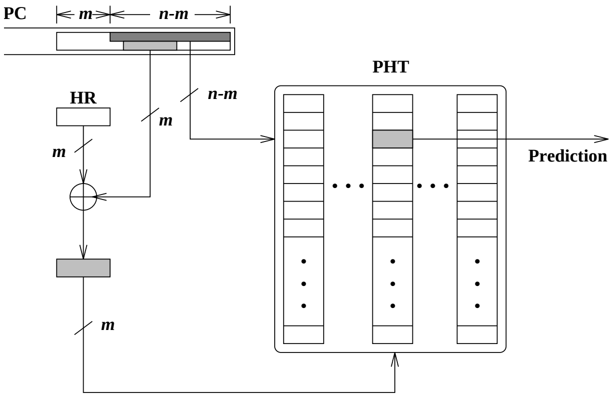


Figure 7: Two-Level Predictor as A Special Case

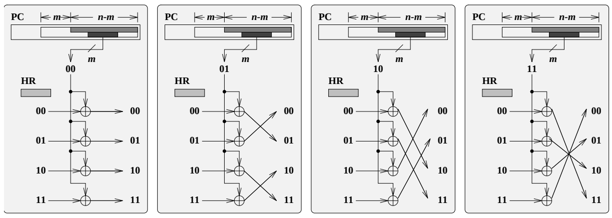


Figure 8: Index Swapping Effect from XOR Function

will be still mapped to the same row except that it may be using a different column index mapping according to its run-time HR content. Thus, the prediction result would remain the same comparing the original arrangement and the new arrangement pertaining to a non-alias case. For an alias case between two different branches that are mapped to the same row, the alias effect still remains the same since both instructions would have the same XOR-mask content from within their identical row index content. An example showing such a swapping between two instructions is given in Figure 9. It happens so because no extra PC information is used and thus two aliased branches cannot be differentiated with the same PC index. As shown in Figure 9 two aliased branches A and B with different histories are mapped to different columns along the same row before XORing with XOR-mask from the PC. Consequently, after XORing as shown in the second figure, both A and B are mapped to different columns but are just swapped around which does not lead to different prediction result from the two-level predictor. So it can be concluded that two-level predictor is also a special case of this generalized predictor.

3.4 Special Case #3: Gshare

Gshare is one of the global two-level predictors, which exploits correlation by basing the prediction on the outcome of the recently executed branches. The XORing is done to incorporate history information into the PC index thereby differentiating interfering branches with the help of history bits. Similarly Gshare can be organized as a special case of our proposed predictor, through which we can easily analyze the benefits brought by this technique. Figure 10 demonstrates this new arrangement. In the following, a thorough analysis on how Gshare compares to the two special cases thus far introduced is given.

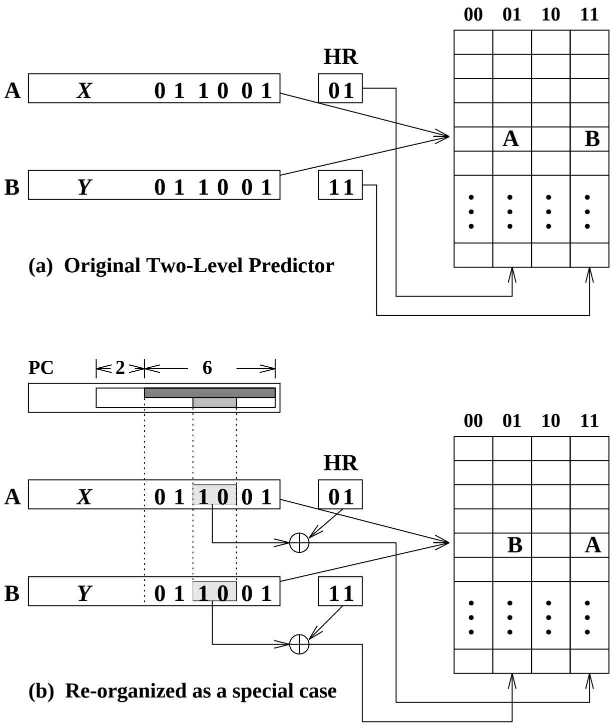


Figure 9: Index Swapping between Two Aliased Instructions

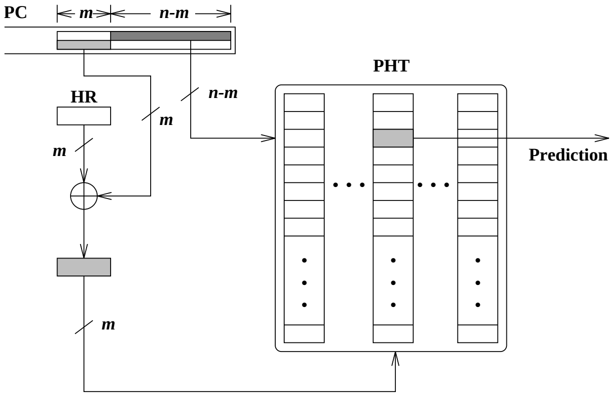


Figure 10: Gshare Predictor as A Special Case

3.4.1 Gshare vs. One-Level Predictor

As shown earlier, the original Gshare predictor is similar to one-level predictor in the way its PHT is organized with the difference that Gshare additionally uses history information and so produces much better prediction accuracies than the one-level predictor. Similar to two-level predictor, Gshare benefits from exploiting correlation information from the use of HR, but it addresses the tradeoff between alias problem and loss of correlation in a more delicate manner, which is to be described in the following section.

3.4.2 Gshare vs. Two-Level Predictor

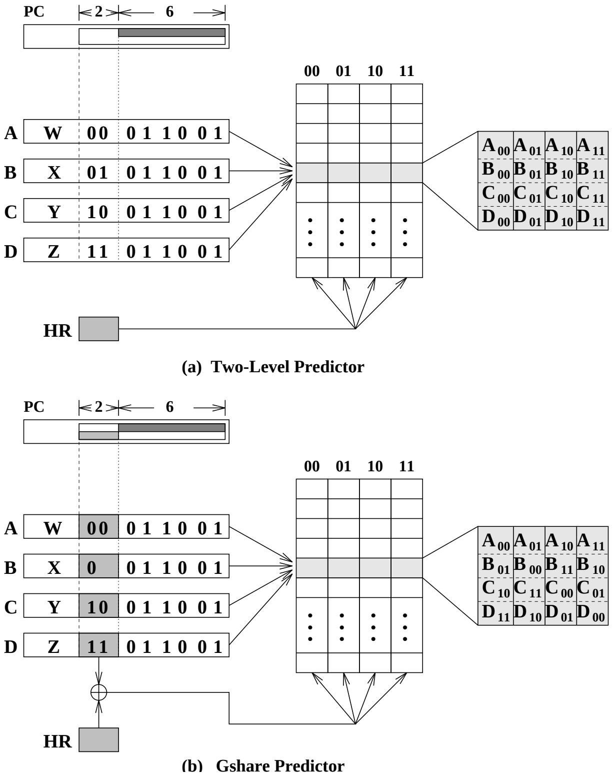
Gshare, when compared to the two-level predictor, is said to be advantageous due to the XORing effect and can be explained as shown in Figure 11. The benefit comes from that fact that, in most programs, global history (HR) content tends to exhibit one dominant pattern, either $00\ldots00$ or $11\ldots11$, due to loop constructs especially when the history depth used (m) is small. This claim has also been confirmed by our simulation results. Consider four aliased branches A, B, C and D that are aliased into the same row in the PHT. The mapping of these branches with different values of HR is shown for both two-level predictor in (a) and Gshare predictor in (b). A_{00}, B_{00}, C_{00} and D_{00}, where the subscripts denote the HR contents, are mapped to different columns of the same row in Gshare as compared to same column in two-level. Same scenario applies to the case when HR has a content of 11. This is one of the advantages of Gshare because aliased branches with most dominant history patterns are now mapped to different locations thereby reducing destructive overlapping. That is, assuming that they have the same history, all the four branches are mapped to the same column in a two-level predictor and so cannot be distinguished. In Gshare on the other hand, these aliased branches are "dispersed" to different columns and so the problem is resolved.

The mapping difference described above is the only distinction between the Gshare and the two-level predictor, and such a distinction may not always favor the Gshare due to different program behavior. A very important conclusion that can be drawn from this is that the Gshare is essentially identical to the two-level predictor if the $(n-m)$-bit row index does not lead to any alias problem. That is, the dispersion of dominating entries among aliasing branches no long exists, thus leading to no more difference in prediction result.

Size of the XOR mask also plays a very important role in Gshare's potential performance. Similar to the two-level predictor, the larger the mask is (larger m), the more history correlation among branches can be exploited. On the other hand, a larger mask leads to more alias problems, although the column mapping of Gshare may provide a better alias-differentiating support than the two-level one from our discovery. As one of the most extreme case, in one of the original Gshare designs, the XORing function is performed over the entire PC index; that is, m is set to be equal to n. This in general leads to an undesirable performance due to its excessive alias problems.

3.5 The Proposed Generalized Predictor with Extensions

A generalized predictor as proposed can be specified with the position of the m-bit XOR mask in PC. Let the least significant bit of this mask be starting at bit W_s; that is, the mask ranges from bit W_s to $W_s + m - 1$. Each of the special cases is then defined as in the following:



(a) Two-Level Predictor

(b) Gshare Predictor

Figure 11: Difference between Gshare Predictor and Two-Level Predictor

Case	Range of W_s	HR value
(a)	$0 \leq W_s \leq n-2m$	run-time
(b)	$n-2m+1 \leq W_s \leq n-m-1$	run-time
(c)	$W_s = n-m$	$00\ldots00$
(d)	$W_s = n-m$	run-time
(e)	$n-m+1 \leq W_s$	run-time

Clearly, case (a) corresponds to the two-level predictor, case (c) to the one-level predictor and case (d) corresponds to the Gshare predictor, while cases (b) and (e) have never been addressed.

Case (b) is essentially a combination of two-level and Gshare predictors. Such a hybrid design displays a various degree of dispersion on dominating entries of aliasing branches. Figure 12 shows an example similar to the one presented in Figure 11. As a compromising point in between the two-level one that shows zero dispersing effect and the Gshare that has a 100% dispersing effect, this special case exhibits a 50% dispersing capability. Branch A

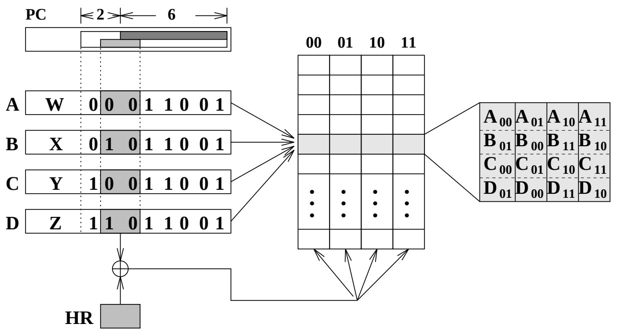


Figure 12: A Hybrid Design of Two-Level and Gshare Predictors

and C remain aligned and so do B and D among the four aliasing branches. Performance from such a hybrid predictor is usually unpredictable due to its nature.

Case (e) is an extension of Gshare aimed at programs with larger address space and/or with a small PHT. The example in Figure 13 shows that, between the two aliasing instructions A and B, the dispersing effect does not take place in the traditional Gshare since both instructions have the same XOR mask value. That is, under such a circumstance, Gshare performs exactly like the two-level predictor. Moving the XOR mask to the higher portion of PC allows the dispersion effect to re-emerge.

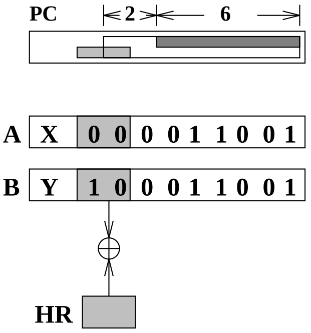


Figure 13: An Extended Design from Gshare

4 Simulation

Our trace analysis and simulation are performed on a SPARC 20 system. Data are obtained using Shade version 5.25 analyzing program. Shade is a dynamic code tracer, which combines instruction set simulations, trace generation and custom trace analysis in a process. Our test programs include a benchmark program from the Stanford Body Benchmark Suite, sfloat, and seven standard Unix utility programs. A brief description of these programs on various aspects of their branch instructions is given in Table 1. A trace analysis has been performed on these eight test programs to show the percentage improvement in misprediction rate.

program	# instr	# taken	# not-taken
sfloat	7730261	111170	125382
ls	1207004	112180	91466
gcc	677017	61258	58096
cc	543900	50519	46536
chmod	492158	45812	42507
grep	491016	45672	42382
awk	490911	45524	42365
pack	486776	45159	42173

Table 1: Description of Test Programs

4.1 Simulation Results

A series of simulation runs are performed by varying the following three parameters on one-bit and two-bit prediction schemes: (1) Number of index bits (n), (2) Number of history bits (m), and (3) Shift in the window (W_s). Performance comparison results are plotted after taking the average of improvement in miss prediction rate for all the above eight programs. The "miss rate improvement percentage" is defined as:

$$\frac{M_{\text{two-level}} - M_{\text{generalized}}}{M_{\text{two-level}}} \times 100$$

where $M_{\text{two-level}}$ and $M_{\text{generalized}}$ denote the miss rate of two-level prediction scheme and that of the generalized one, respectively. Each point in these results corresponds to the average of results from the eight test programs. Figure 14, Figure 15 and Figure 16 show the results for both one-bit and two-bit counters prediction schemes.

We can see that, from all these figures, performance of the generalized predictor is identical to that of the two-level one (i.e. improvement equals to 0) when

$$0 \leq W_s \leq n - 2m$$

Gshare's results (when $W_s = n-m$), in general, are among the better ones for $n = 11$, but are outperformed by most of cases with larger W_s values (the Extended Gshare scheme) for $n = 10$ and $n = 9$. This finding verifies our analysis that the Extended Gshare scheme allows the dispersion effect to reappear when a small PHT is used. The Hybrid scheme (when $n - 2m + 1 \leq W_s \leq n - m - 1$) does not distinguish itself clearly from the two-level one.

5 Conclusion

In this paper, we propose a generalized branch predictor and show that most of the commonly used predictors are actually special cases of this generalized predictor. Similarities and differences among predictors are clearly identified. Based on this construct, we are able to easily analyze and compare the benefits and drawbacks among different predictor designs. We also show that a simple extension of the Gshare design, a direct notion from the generalized design, can outperform Gshare in many cases. This is an improvement at no additional cost on hardware. A dynamic selection of the XOR-mask position according to the nature of the program may bring additional improvement. Also, one potential direction that is worthwhile looking into is the understanding and classification of different kinds of conditional branches, which may help predict the otherwise declared "random" branches that have not been addressed by the prediction methods investigated so far.

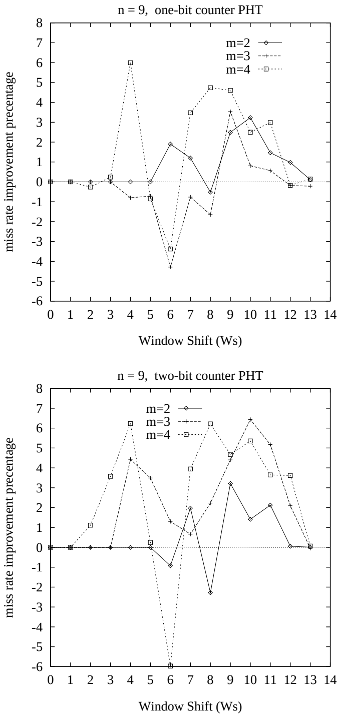


Figure 14: Improvement Results versus Window Shift for $n = 9$

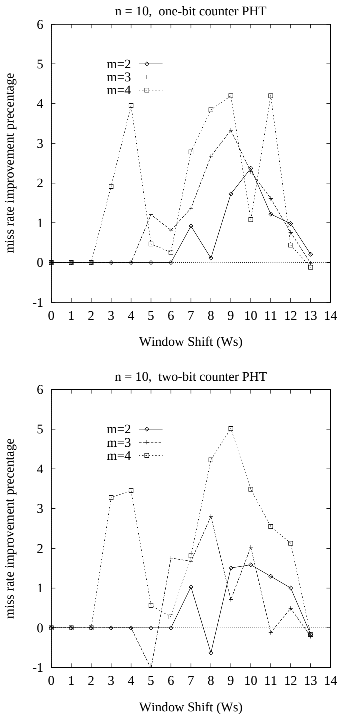


Figure 15: Improvement Results versus Window Shift for $n = 10$

References

[1] T. Ball and J. Larus, "Branch Prediction for Free," *Proc. ACM SIGPLAN 1993 conf. on Prog. Lang. Design and Implementation*, June, 1993.

[2] B. Bray and M. J. Flynn, "Strategies for Branch Target Buffers," *24th Workshop on Microprogramming and Microarchitecture*, 1991, p.42-p.49.

[3] B. Calder and D. Grunwald, "Fast & Accurate Instruction Fetch and Branch Prediction," *Intl. Symp. on Computer Architecture*, April, 1994.

[4] M.-C. Chang and Y.-W. CHou, "Branch Prediction using both Global and Local Branch History information," *Computers and Digital Techniques, IEE Proceedings, Volume: 149 Issue: 2*, March 2002.

[5] P.-Y. Chang, M. Evers and Y.N. Patt, "Improving branch prediction accuracy by reducing pattern history table interference," *Parallel Architectures and Compilation Techniques*, 1996.

[6] R. S. Chappell, F. Tseng, A. Yaoz and Y. N. Patt, "Difficult-path Branch Prediction Using Subordinate Microthreads," *Proc. 29th Annual International Symposium on Computer Architecture*, 2002.

[7] J. Fisher and S. Freudenberger, "Predicting Conditional Branch Direction From Previous Runs of a Program," *Proc. 5th Annual Intl. Conf. on Architectural Support for Prog. Lang. and Operating System*, October, 1992.

[8] J. K. F. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Computer*, January, 1984, p.6-p.22.

[9] S. McFarling, "Combining Branch Predictor," *Technical Report, Digital Western Research Laboratory*, June, 1993.

[10] S. McFarling and J. Hennessy, "Reducing the Cost of Branches," *The 13th Annual Intl. Symposium of Computer Architecture*, 1986, p.396-p.403.

[11] 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 System*, Oct. 1992.

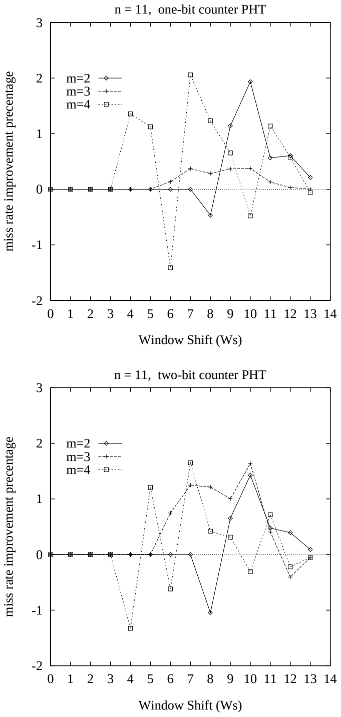


Figure 16: Improvement Results versus Window Shift for $n = 11$

[12] D. Patterson and J. Hennessy, "Computer Architecture: A Quantitative Approach, 2nd Edition," *Morgan Kaufmann Publishers, Inc.*, 1995.

[13] C. Perleberg and A. J. Smith, "Branch Target Buffer Design and Optimization," *IEEE Transactions on Computers*, April, 1993, P396-412.

[14] J. Smith, "A Study of Branch Prediction Strategies," *Proc. 8th Annual Intl. Symp. on Computer Architecture*, May, 1981, p.135-p.147.

[15] Z. Su and M. Zhou, "A Comparative Analysis of Branch Prediction Schemes," *Technical Report, University of California at Berkeley*, 1995.

[16] "Shade Manual," *Sun Microsystems*, 1995.

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

[18] T. Yeh and Y. Patt, "Alternative Implementations of Two-level Adaptive Branch Prediction," *Proc. 19th International Symp. on Computer Architecture*, May. 1992.

[19] T. Yeh and Y. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *Proc. 20th Annual Intl. Symp. on Computer Architecture*, May. 1993.

[20] T. Yeh and Y. Patt, "Two-level Adaptive Branch Prediction and Instruction Fetch Mechanism for High Performance Superscalar Processors," *Computer Science and Engineering Div. Tech. Report CSE-TR-182-93, University of Michigan*, Oct. 1993.

[21] C. Young and M. Smith, "Improving the Accuracy of Static Branch Prediction Using Branch Correlation," *Technical Report 06-95, Center for Research in Computing Technology, Harvard University*, March, 1995.

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