The Bimode++ Branch Predictor

Kenji Kise[†,‡], Takahiro Katagiri[†], Hiroki Honda[†], and Toshitsugu Yuba[†]
[†] Graduate School of Information Systems
The University of Electro-Communications
[‡] PRESTO, Japan Science and Technology Agency (JST)
{kis, katagiri, honda, yuba}@is.uec.ac.jp

Abstract

Modern wide-issue superscalar processors tend to adopt deeper pipelines in order to attain high clock rates. This trend increases the number of on-the-fly instructions in processors and a mispredicted branch can result in substantial amounts of wasted work. In order to mitigate these wasted works, an accurate branch prediction is required for the high performance processors.

In order to improve the prediction accuracy, we propose the bimode++ branch predictor. It is an enhanced version of the bimode branch predictor. Throughout execution from the start to the end of a program, some branch instructions have the same result at all times. These branches are defined as extremely biased branches. The bimode++ branch predictor is unique in predicting the output of an extremely biased branch with a simple hardware structure. In addition, the bimode++ branch predictor improves the accuracy using the refined indexing and a fusion function.

Our experimental results with benchmarks from SpecFP, SpecINT, multi-media and server area show that the bimode++ branch predictor can reduce the mispredict rate by 13.2% to the bimode and by 32.5% to the gshare.

1 Introduction

Modern wide-issue superscalar processors tend to adopt deeper pipelines[2, 3, 10] in order to attain high clock rates. This trend increases the number of on-the-fly instructions in processors and a mispredicted branch can result in substantial amounts of wasted work. In order to mitigate these wasted works, an accurate branch prediction is vital for the high performance processors.

Since Smith[9] discussed the branch prediction strategies in 1981, many branch prediction schemes have been investigated. McFarling[8] proposed the gshare predictor widely used in commercial microprocessors. The gshare improves the two-level adaptive predictor using the "Exclusive OR" function of the global branch history register (BHR) and the branch address to generate the index into the pattern history table (PHT).

Lee and Mudge[7] introduced the bimode branch predictor. The organization of the bimode predictor is shown in figure 1. It is an attempt to replace destructive aliasing of the gshare predictor with neutral aliasing with three PHTs: choice, taken and untaken[1] PHTs. The taken PHT and the untaken PHT are referred to direction PHTs. The direction PHTs are indexed by the "Exclusive OR" function of branch address and the BHR. The choice PHT selects the one of direction PHTs to be used, and the two-bit saturating counter of the selected PHT makes a prediction.

Recently new approaches like neural predictor[5, 4] have been proposed. These predictors achieves good accuracy. But, their structures are too complex to be implemented in hardware. We propose the bimode++ branch predictor. It is based on the bimode branch predictor because of its hardware simplicity. Throughout execution from the start to the end of a program, some branch instructions have the same result at all times. These branches are defined as extremely biased branches. The bimode++ branch predictor is unique in predicting the output of the extremely biased branch with a simple hardware structure. In addition, the bimode++ branch predictor improves the accuracy using the refined indexing for choice PHT and a fusion function of outputs from three PHTs.

The accuracy of the branch predictor is evaluated using the framework for the branch predictor contest (Championship Branch Prediction[12]) that was held with support of Intel MRL and IEEE TC-uARCH. The framework contains 20 benchmark traces from SpecFP, SpecINT, multi-media and server area. We show that the bimode++ achieves better prediction accuracy than the gshare and the bimode.

The rest of this paper is organized as follows. Section 2 proposes the bimode++ branch predictor. Section 3 reports the evaluation results. Section 4 is a discussion and Section

[1] The word "untaken" is used to indicate that the branch prediction output is "not taken".

Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'05)
1527-1366/05 $20.00 © 2005 IEEE

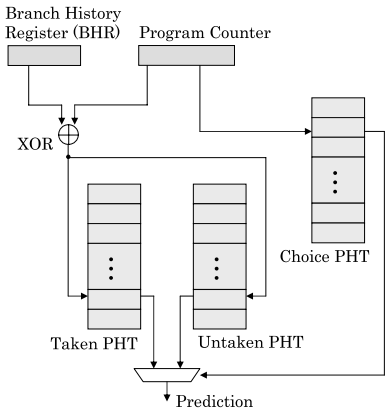

Figure 1. Bimode branch predictor.

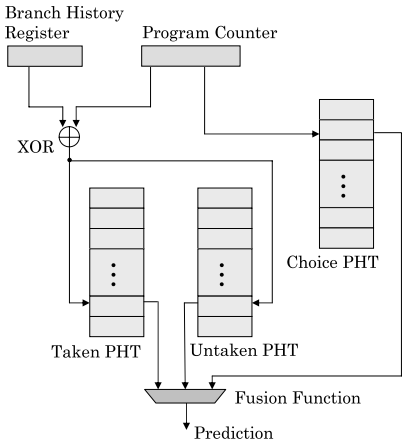

Figure 2. Bimode-fusion branch predictor.

5 contains some concluding remarks. In this paper, a branch predictor may be called a predictor in short.

2 The Bimode++ Branch Predictor

We discuss three techniques which improve the accuracy of the bimode predictor. They are orthogonal and can be applied simultaneously. Then the bimode++ predictor is proposed by combining these techniques.

2.1 Fusion Function

The first technique is to use a fusion function. In the bimode predictor in Figure 1, choice PHT selects the one of direction PHTs to be used. The prediction is made with one counter in the selected direction PHT. Our intuition for proposing to use a fusion function is that a certain amount of information is separately stored in choice, taken, and untaken PHT. Not only using one counter in the selected direction PHT, but also using three counters makes better prediction.

We propose to use a fusion function as shown in Figure 2. This predictor is called the bimode-fusion. The input of the fusion function is six bit from the outputs of choice PHT, taken PHT and untaken PHT. The output is one bit prediction. The question is that what kind of fusion function is suitable for the bimode-fusion predictor? As a promising candidate of a fusion function, we propose to use a majority vote and the property of a transient state of two-bit saturating counter.

2.1.1 Transient State

The two-bit saturating counter is a component used in branch predictors. In the bimode predictor, three PHTs are

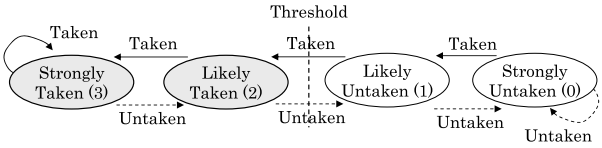

Figure 3. Two-bit saturating up-down counter. The states close to the threshold are defined as transient state.

the table of two-bit saturating counters. The state transition of a two-bit saturating counter is shown in figure 3. The four states of 3, 2, 1 and 0 indicate strongly taken, likely taken, likely untaken and strongly untaken, respectively.

The states close to the threshold (likely taken and likely untaken in a two-bit saturating counter) are defined as the transient state because the prediction in their state may change in a short period. In case of a three-bit saturating counter, the states around the value of three or four will be the transient state. We investigated the accuracy and the value of a saturating counter in the direction PHT and found that the prediction accuracy in the transient state is somewhat lower than the accuracy in the non-transient state.

From this consideration, we propose a fusion function to use a majority vote. If the value of a saturating counter in the selected direction PHT is in the transient state, a majority vote of three two-bit counters from choice, taken and untaken PHTs is used to make the prediction. If the saturating counter is not in the transient state, the prediction is made with a two-bit counter from the selected direction PHT like the bimode predictor.

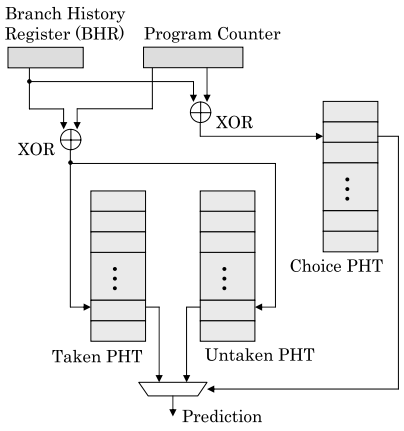

Figure 4. Bimode-indx branch predictor.

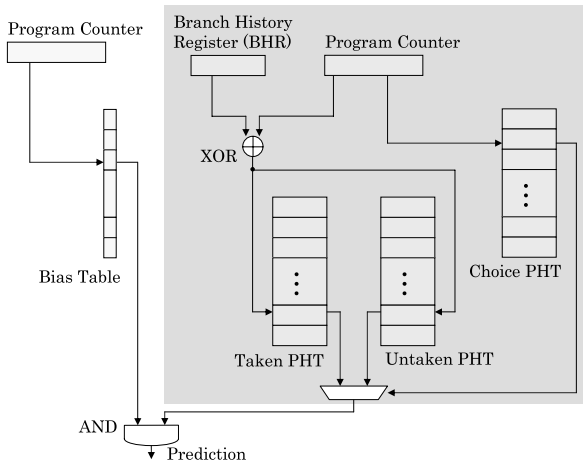

Figure 5. Bimode-plus branch predictor.

2.2 Refined Indexing

The second technique is a simple modification of indexing. In the bimode predictor, the index of the choice PHT is generated using the program counter. Note that the index of the direction PHTs is generated using the "Exclusive OR" function of the branch history register and the program counter.

It was incomprehensible why "Exclusive OR" function is not used for the indexing into choice PHT. We tested and found that the use of the "Exclusive OR" function of the branch history register and the program counter to generate the index into choice PHT improves the prediction accuracy.

A predictor modified to use the refined indexing of the choice PHT is shown in figure 4. This predictor is called the bimode-indx. Since the main purpose of choice PHT is to catch the behavior of each static branches, the use of only a few BHR bits (two or three bit in the 8KB hardware budget) for the choice PHT index is sufficient to improve the accuracy.

2.3 Bias Table

The third technique is a bias table for the extremely biased branches. In execution of a program from start to finish, some branch instructions have the same result at all times. The branch instruction to detect a program error is one of such example. In most cases, no errors occur and the branch result is always untaken.

We define branches that has the same result from start to the prediction as extremely biased branches. We have proposed the bimode-plus predictor[6], which uses the property of extremely biased branches. In Figure 5, the block diagram of the bimode-plus predictor is shown. The shaded area is a part of bimode predictor. We use the taken and untaken tables (each entry is a one-bit flag) to record whether the branch is an extremely biased branch. The taken and untaken tables are referred to the bias table.

First, all entries of the bias table are initialized to zero. Then, when the branch outcome is known to be taken, the entry of the untaken table is updated with the value of one. When the branch outcome is known to be untaken, the entry of the taken table is updated with the value of one. Note that once a flag is set to one, it does not return to zero.

If an entry of untaken table has the value of zero, it indicates that the all history of the branch was untaken. Then the branch may be an extremely biased untaken branch and the predictor makes the prediction of untaken. If an entry of taken table has the value of zero, then the branch may be an extremely biased taken branch and the predictor makes the prediction of taken; otherwise, the outcome of the bimode predictor is used as a prediction.

The choice, taken, and untaken PHT are updated only when the prediction is not made with the bias table. By eliminating the entry for the extremely biased branches from these PHTs, it is possible to mitigate the table interference.

If the rich hardware resource is available for a branch predictor, the bimode-plus with the taken and untaken tables brings good accuracy. In case of the restricted hardware budget, the configuration with only the untaken table may achieve better accuracy.

2.4 Bimode++ Branch Predictor

We propose the bimode++ predictor. Three techniques discussed in sections 2.1, 2.2, and 2.3 can be applied to the bimode predictor simultaneously. Basically the bimode++ predictor is the combination of these techniques.

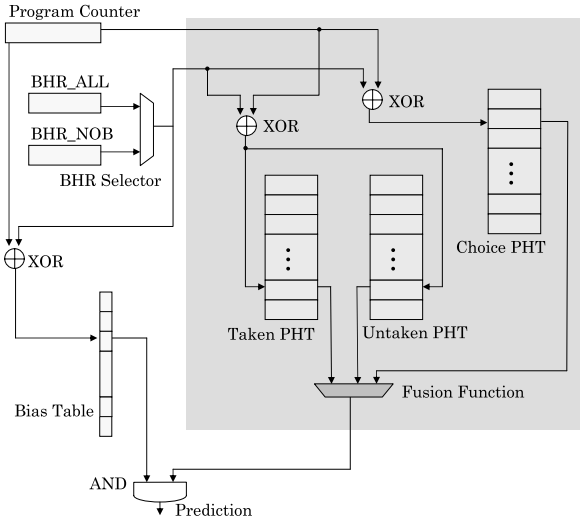

Figure 6. Bimode++ branch predictor.

A block diagram of the bimode++ predictor is shown in Figure 6. The bimode++ predictor uses the bias table for the extremely biased branches. All entries of the bias table are initialized to zero. When the branch outcome is known to be taken(untaken), the entry of the untaken(taken) table is updated with the value of one. If an entry of the taken(untaken) table has the value of zero, the predictor makes the prediction of taken(untaken). The bias table is indexed with the "Exclusive OR" function of the program counter and the branch history register.

The shaded area is the combination of the bimode-fusion and the bimode-indx. The index of three PHTs is generated using the "Exclusive OR" function of the branch history register and the program counter. The prediction is made by a fusion function whose input is six bit from the outputs of choice, taken and untaken PHTs. The choice, taken, and untaken PHT are updated with the same manner of the bimode only when the prediction is not made by the bias table.

Three PHTs are updated only when the prediction is not made by the bias table. In the same way, should the branch history register be updated only when the prediction is not made by the bias table? Should the branch history register be updated by all branch outcomes? We found that the decision is not simple and our solution is a dynamic adaptation among two branch history registers.

2.4.1 Dynamic Branch History Register Selection

We propose to use two BHRs updated using different policies and select an appropriate BHR from among them. One BHR, denoted as BHR_ALL, is updated with the results of all conditional branch instructions. This policy is similar to that used in the traditional bimode predictor. The other BHR, denoted BHR_NOB, is updated using the results of non-biased conditional branches. The BHR selector controls the selection of either BHR_ALL or BHR_NOB. These registers and the selector are shown at the upper-left of figure 6.

The appropriate algorithm and implementation of the BHR selector is another issue. By the preliminary evaluation, we found that BHR_ALL is suitable for a benchmark having many extremely biased branches (such as server benchmarks). In order to detect the number of executed extremely biased branches, we use the saturating counter denoted as bias_mod_cnt, which is incremented when the content of the bias table entry is modified. If the counter has the maximum value and saturation, the BHR_ALL is selected.

3 Evaluation

The prediction accuracy is evaluated using the common evaluation framework version 3 for the championship branch prediction sponsored by Intel MRL and IEEE TC-uARCH. The framework contains 20 benchmark traces classified into 4 categories. The categories are SPECfp (FP), SPECint (INT), Multimedia (MM) and Server (SERV). The predictor parameters are optimized for the 8KB hardware budget.

Each branch predictor has many configuration parameters. From the results of parameter tuning on 8 KB hardware budget, the following parameters are used. For the 16 KB hardware budget configuration, the number of entries in each table is set to double that for 8 KB.

For the bimode, the bimode-fusion, and the bimode-indx, the number of entries for choice PHT is set to double of direction PHTs. An 8 KB predictor, for example, has 4 KB choice PHT, 2 KB taken PHT, and 2 KB untaken PHT.

For the bimode-plus, the same number of entries is used for choice PHT and direction PHT. We use only the untaken table as the bias table. The number of entries for the bias table is double of PHTs. An 8 KB bimode-plus predictor has 2 KB choice PHT, 2 KB taken PHT, 2 KB untaken PHT, and 2 KB the bias table.

For the bimode++, the number of entries for taken PHT is set to four times that for choice PHT. The number of entries for untaken PHT is set to double that for choice PHT. The number of entries for the bias table double of choice PHT. An 8 KB bimode++ predictor has 1 KB choice PHT, 4 KB taken PHT, 2 KB untaken PHT, and 1 KB bias table.

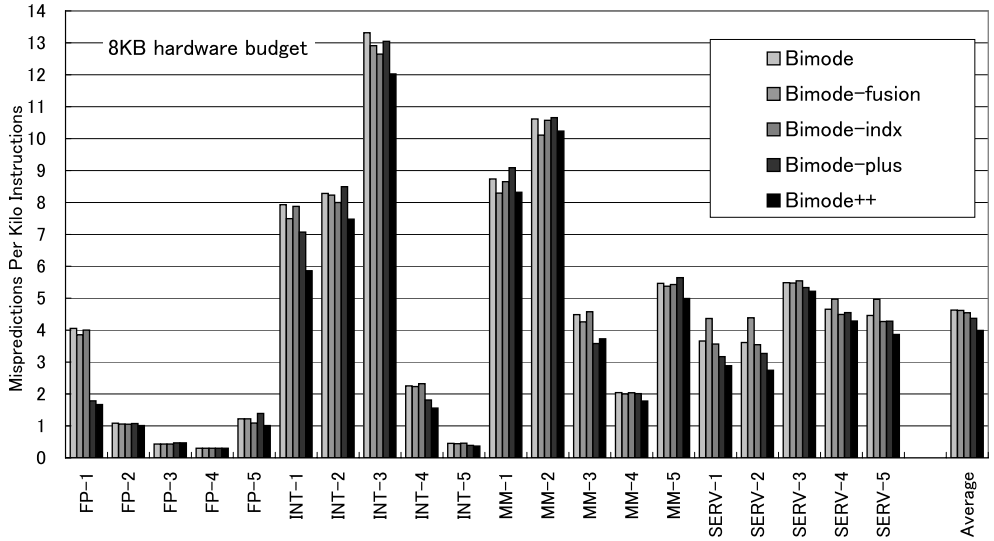

Figure 7. Comparing the prediction accuracy of branch predictors in 8 KB hardware budget. The data is obtained with the common evaluation framework version 3 for the championship branch prediction.

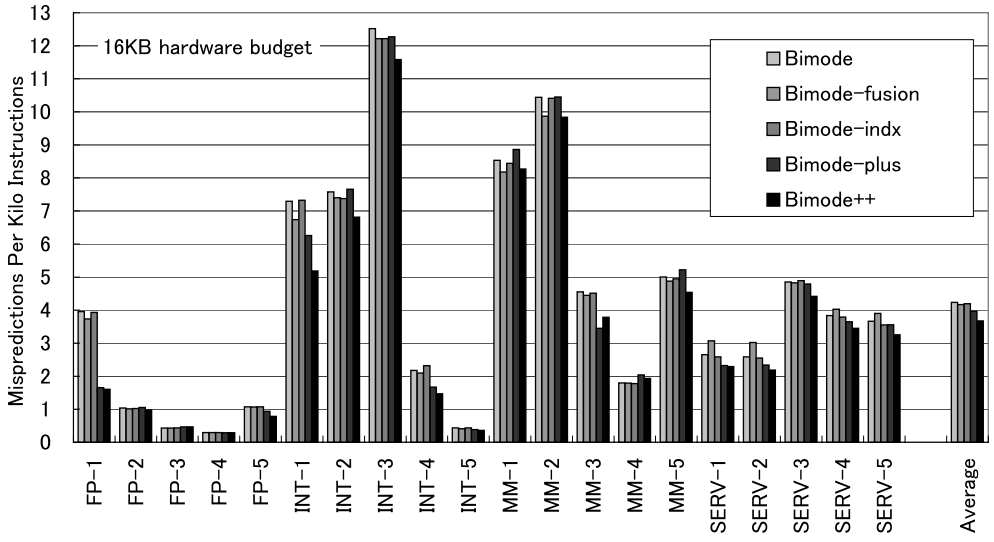

Figure 8. Comparing the prediction accuracy of branch predictors in 16 KB hardware budget.

3.1 Prediction Accuracy

The prediction accuracy of 8 KB and 16 KB predictors is summarized in Figures 7 and 8. The graphs show the data of the five branch predictors called the bimode, the bimode-fusion, the bimode-indx, the bimode-plus, and the bimode++.

Figure 7 gives the results of the predictions when the hardware budget is 8 KB. The number of mispredictions per 1000 instructions is 4.62 for the base predictor of bimode. By comparing the bimode and the bimode-fusion, the influences of the mechanism using a fusion function are understood. The bimode-fusion improves the production accuracy in several benchmarks of SpecINT (INT-1, INT-3) and multimedia (MM-1, MM-2) but lowers the accuracy in the server benchmarks. Compared with the bimode, the bimode-fusion reduces the prediction errors by 0.17% on average. By comparing the bimode and bimode-indx, the influences of the refined indexing of choice PHT are understood. The bimode-indx improves the prediction accuracy in INT-2, INT-3, and SERV-5. Compared with the bimode, the bimode-indx reduces the prediction errors by 1.81% on average. By comparing the bimode and bimode-plus, the influences of the bias table are understood. The bimode-plus improves the prediction accuracy in FP-1 and MM-3 significantly. Compared with the bimode, the bimode-plus reduces the prediction errors by 5.54% on average. The bimode++ branch predictor achieves the highest prediction accuracy. Compared with the bimode, the bimode++ reduces the prediction errors by 13.7%. In summary, the prediction error reduction rates from the bimode are 0.17% for the bimode-fusion, 1.81% for the bimode-indx, 5.54% for the bimode-plus, and 13.7% for Bimode++ on 8KB configuration.

Figure 8 gives the results of the predictions when the hardware budget is 16 KB. The number of prediction errors per 1000 instructions is 4.23 for the bimode. The prediction error reduction rates from the bimode are 1.52% for the bimode-fusion, 0.96% for the bimode-indx, 6.53% for the bimode-plus, and 13.2% for the bimode++.

We confirmed that the bimode++ predictor reduces the prediction errors by more than 13% compared with the bimode on both 8 KB and 16 KB configurations.

4 Discussion

4.1 Hardware Complexity and Related Works

To realize the proposed bimode++ predictor, it is necessary to augment a bias table and several circuits to the bimode predictor. These are not huge modifications. Compared with the bimode branch predictor, the bimode++ branch predictor does not require significant latency increase for prediction. Compared with the bimode branch predictor, the perceptron branch predictor[5] proposed in 2001 reduces the prediction errors by 8.2%. The perceptron predictor, however, has such disadvantages as complicated hardware and long prediction latency. According to the reference[4], the prediction latency of the 8 KB perceptron predictor is estimated to be four cycles even if the latency reduction techniques are used. Compared with the perceptron predictors, the structures of the bimode++ predictor are very simple. While keeping the simple hardware configuration, the bimode++ predictor reduces prediction errors by more than 13% compared with the bimode branch predictor.

The proposed predictor using extremely biased branches has some common features with the agree predictor[11] shown in Figure 9. By regarding a branch direction stored in a bias bit of the BTB (branch target buffer), the agree predictor makes a prediction based on an agreement or

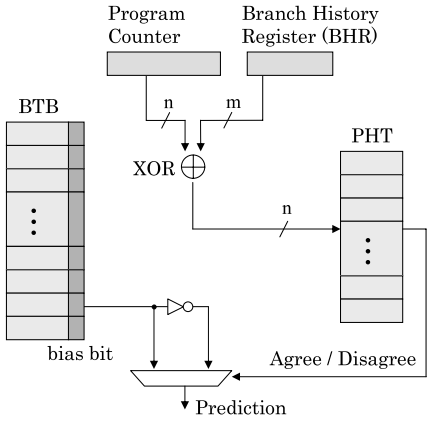

Figure 9. Agree predictor.

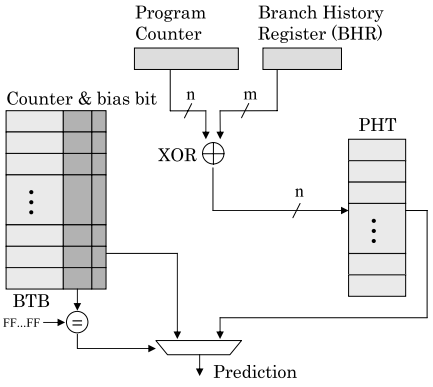

Figure 10. Filter mechanism.

disagreement with the bias bit. A big difference of the agree predictor is that the bimode++ predictor can detect extremely biased branches by considering the long execution history of a branch. In addition the bimode++ prediction has advantage not to change the structure of the BTB.

The bimode++ predictor using the property of extremely biased branches also has some common features with the filter mechanism[1] shown in Figure 10. By using a counter added to the BTB, the filter mechanism predicts the biased branches with a history of certain period intervals. The filter mechanism is similar to our approach in theory but greatly differs in that the mechanism uses a counter. In the learning period until a counter is saturated, the filter mechanism uses a PHT for prediction. In addition, the filter mechanism requires the appropriate setting of a threshold to determine when the counter is saturated. In terms of hardware budget, our approach to use a flag has an advantage because it does not use a counter. In addition, the bimode++ predictor generates a bias table index using the program counter and the branch history register. This is difficult for the filter mechanism which changes the structure of the BTB. The bimode++ predictor does not require the BTB modification and does not refer the BTB to obtain a prediction.

4.2 Parameter Tuning

In this paper, we used a predictor where its configuration parameters were optimized to achieve better accuracy on average under the 8 KB hardware budget. The optimum parameters may be different for each benchmark. If the resources can be allocated dynamically, the prediction accuracy may improve dramatically. The parameter tuning, including the dynamic resource allocation, is a future research subject.

4.3 Processor Performance

By measuring and comparing the prediction accuracy of the bimode++ predictor with that of the conventional predictor, we verified great accuracy improvement.

It is necessary to study the influences of the processor performance and the predictor accuracy. The detailed performance evaluations with clock-level processor simulators are required. In an SMT (simultaneous multi-threading) environment, sharing of branch prediction tables may cause the significant performance degradation. Discussions and evaluations not only in a uni-processor and single-thread environment but also in various processor models are required as a future research subject.

5 Summary

Modern wide-issue superscalar processors tend to adopt deeper pipelines to attain high clock rates. This trend increases the number of on-the-fly instructions in processors and a mispredicted branch can result in substantial amounts of wasted work. In order to mitigate these wasted works, an accurate branch prediction is required for the high performance processors.

In order to improve the prediction accuracy, we proposed the bimode++ branch predictor. Throughout execution from the start to the end of a program, some branch instructions have the same result at all times. These branches are defined as extremely biased branches. The bimode++ branch predictor is unique in predicting the output of an extremely biased branch with a simple hardware structure. In addition, the bimode++ branch predictor improves the accuracy using the refined indexing and the fusion function.

The accuracy of the branch predictor was evaluated using the framework for the branch predictor contest. The framework contains 20 benchmark traces from SpecFP, SpecINT, multi-media and server area. Our experimental results showed that the bimode++ branch predictor reduces prediction errors by 13.7% when the hardware budget is 8 KB and by 13.2% when the hardware budget is 16 KB compared with the conventional bimode branch predictor.

Acknowledgements

This study was partially supported by PRESTO, Japan Science and Technology Agency (JST).

References

[1] Po-Yung Chang, Marius Evers, and Yale N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *International Conference on Parallel Architectures and Compilation Techniques*, 1996.

[2] A. Hartstein and Thomas R. Puzak. The optimum pipeline depth for a microprocessor. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 7–13. IEEE Computer Society, 2002.

[3] M. S. Hrishikesh, Doug Burger, Norman P. Jouppi, Stephen W. Keckler, Keith I. Farkas, and Premkishore Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 fo4 inverter delays. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 14–24. IEEE Computer Society, 2002.

[4] Daniel A. Jimenez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 243–252, 2003.

[5] Daniel A. Jimenez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High-Performance Computer Architectuer*, pages 197–206, 2001.

[6] Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba. The Bimode-Plus Branch Predictor. In *IEICE Technical Report CPSY-2003-10 (in Japanese)*, pages 25–30, 2003.

[7] Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 4–13, 1997.

[8] S. McFarling. Combining branch predictors. Technical report, Digital Equipment Corporation WRL TN-36, 1993.

[9] James E. Smith. A study of branch prediction strategies. In *Conference proceedings of the eighth annual symposium on Computer Architecture*, pages 135–148, 1981.

[10] Eric Sprangle and Doug Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 25–34. IEEE Computer Society, 2002.

[11] Eric Sprangle, Robert S. Chappell, Mitch Alsup, and Yale N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th International Symposium on Computer architecture*, pages 284–291, 1997.

[12] The Journal of Instruction-Level Parallelism. Championship branch prediction, http://www.jilp.org/cbp.