Novel Branch Prediction Strategy based on Adaptive History Length for High-Performance Microprocessor

SANG HOON LEE[1], JONGSU PARK[2*]
School of Electrical and Electronic Engineering
Yonsei University
Shinchon-dong, Seodaemun-gu, 03722, Seoul
SOUTH KOREA
sanghoon0.lee@lge.com[1], jspark@yonsei.ac.kr[2*]

Abstract: - The demands for high-performance microprocessors have recently increased. Accurate branch prediction is one of the most important factors for high-performance processors. In order to predict branch outcomes, instruction program counter bits and the history of recently executed branch outcomes are used. Among the executed branch outcomes, some histories are useful while others are useless. In addition, these useful/useless histories vary among branch instructions. Numerous studies have shown a method that identifies optimal history. However, little research has been done regarding the treatment of useless history. In this paper, a new method called Instruction Address alloyed History Length Modification branch predictor is proposed to handle the useless history bits. When PHT entries are 4,096, IAaHLM has a prediction accuracy of 93.22% and Gshare has a prediction accuracy of 91.84%.

Key-Words: - branch prediction, branch predictor, dynamic branch history control, microprocessor

1 Introduction

From smartphones to smart pads, the importance of high-performance microprocessors, which have a significant relationship with device performances, has increased. In order to implement high-performance microprocessors, instructions per clock (IPC) should be increased. There are two ways to increase the IPC. The first method is to use a superscalar structure, which fetches and executes multiple instructions concurrently based on instruction-level parallelism (ILP). However, inadequate instruction parallelism in a basic block requires faithful branch prediction for accurate code scheduling which will eventually increase ILP. The second method for increasing ILP is the use of a deep pipeline. A deep pipeline increases the stages of the pipeline and decreases the number of logic gate levels in each pipeline stage. The major advantage of a deep pipeline is the ability to decrease machine cycle time and hence increase clocking frequency. As a pipeline is deeper, the penalty of pipeline stall from incorrect branch predictions is increased. Therefore, an accurate branch prediction technique is highly important and is required for high-performance microprocessors that use not only a superscalar structure but also a deep pipeline.

The first branch predictor, proposed by Smith [1], was the Smith predictor, which is the basic of modern branch predictors. The predictor consists of a saturating two-bit counter, and prediction is determined by the most significant bit (MSB) of the counter. As the predictor has only one counter, totally different branch instructions use and update the same counter for branch prediction. In addition, the counter is not able to apply previously executed branch instruction outcomes to new branch predictions. Therefore, the predictor has low prediction accuracy. In order to overcome these shortcomings, Yeh et al. [2] proposed a typical two-level branch predictor. This predictor contains a table, called a pattern history table (PHT), whose entries are filled with two-bit saturating counters, and a branch history register (BHR) to store certain numbers of previously executed branch outcomes. In order to predict branch outcomes, n bits from the BHR and same n bits from the instruction's program counter are used as an input to hash function and the outcome is used to index PHT. The two- bit saturating counter, contained in the indexed entry of the PHT, predicts whether the branch is taken. When the branch outcome is available, the counter is updated based on the prediction and the outcome. In addition, the BHR is also updated to contain the outcome. There are two different ways to contain the recent branch outcomes[3, 4]. While the first method causes branch instructions to save their outcomes to separate registers, the second method

does not separate branch instructions, leading to the outcomes being stored in same register. The second method predicts branch instructions more accurately than the first method for the following reason: since most branches are related to previous executed branch outcomes, accurate branch prediction is available when this relationship is used. Therefore, the technique storing the executed branches to different registers is not able to use the relationships between branch instructions, preventing the predictor from making accurate branch predictions. In contrast, saving different branch instruction outcomes into the same register makes the predictor use the relationship and have high accuracy in branch predictions. In summary, to use branch correlation, branch predictors with single register for branch outcomes have a high chance of making accurate branch predictions [5].

Since branch prediction accuracy is related to the BHR, the predictor might increase its accuracy when BHR length is increased. However, increasing BHR length also increases the training duration needed for accurate prediction and the predictor is not able to have the most accurate predictions during the training period. Evers et al. [6] mentioned that the amount of correlation varies between branch instructions. In other words, BHR length should be adjusted for branch instructions that have different correlations with other branches.

Elastic history buffers [7] and regions of limited branch correlation [8] predictors dynamically adjust BHR length after prior program profiling. However, as these predictors dynamically control the BHR with prior profiling, these predictors are not able to be implemented in hardware. In order to overcome this weakness, predictors that dynamically control the BHR without prior profiling are proposed. Dynamic history length fitting [9] dynamically adjusts BHR length for accurate branch prediction. Although this predictor does not require prior profiling, the predictor adjusts BHR length by trial and error during the execution of a program. Branch prediction, grounds for trial and error, hypothesizes that the current and future branch instructions will follow a certain pattern from previous branch instructions. However, there is no guarantee that post-branch instructions will follow the pattern. As these branch predictors have some weaknesses, such as pattern requirements and prior-profiling, it is hard to say that these predictors are effective.

The Dynamic per Branch History Length Adjustment (DpBHLA) [10] branch predictor controls BHR length dynamically based on a logical basis. The branch predictor keeps track of basic block executions that are a straight-line code sequence, such as register update instructions (load, add) or branch instructions. While tracking the block, the predictor records the sequence of the register update instructions and the information is stored in the Branch Register Dependency Table (BRDT), shown in Fig. 1, whose entry amount is same as the number of registers. When predicting branch instructions, the indexed entry is determined based on the source register of the branch instruction. For example, when the branch instruction tests the content of register eight, the indexed entry is the eighth entry. Since a history of register updating instructions is stored in each BRDT entry, data dependency with a prior basic block is detected by indexing the entry. Therefore, using register updating information, we are able to indicate data dependency with other branches, and this knowledge, which helps the branch predictor distinguish useful information between BHR bits, increases prediction accuracy. In order to adjust BHR bits, the DpBHLA predictor resets unnecessary bits (unrelated history bits) to zero while leaving the rest of the bits (necessary bits) unchanged. However, Porter [8] mentioned that setting some BHR bits to zero would decrease PHT usage since the predictor would be biased to some parts of certain PHT regions and ineffective PHT treatment would decrease branch prediction accuracy. Therefore, in order to overcome this weakness, this study proposes a structure called Instruction Address alloyed History Length Modification (IAaHLM), which changes unnecessary bits in the BHR to useful bits, instead of resetting them to zero, to increase prediction accuracy.

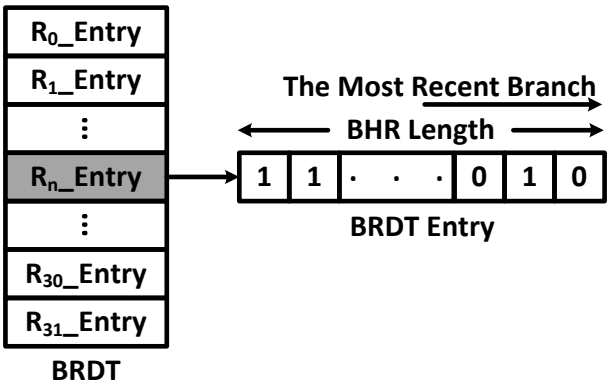

Fig. 1. Branch Register Dependency Table (BRDT)

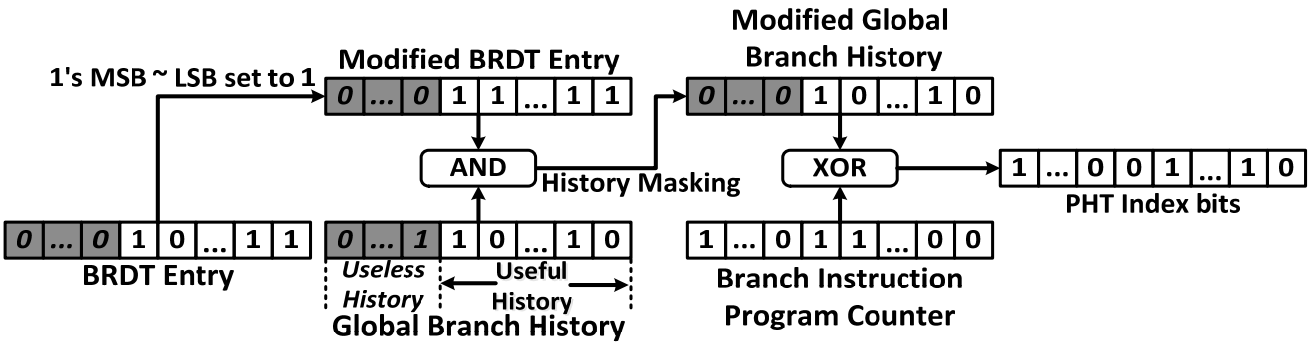

Fig. 2. Exclusive-or arithmetic between modified history and branch instruction program counter

This paper is organized in four parts. Section two introduces limits of dynamic history length control and describes the proposed structure. Section three shows experiments background and analyzes the experimental results by comparing them with previously proposed structures. Section four offers conclusion.

2 Instruction Address Alloyed History Length Modification

In this section, the IAaHLM branch predictor is proposed. The IAaHLM branch predictor is based on the DpBHLA branch predictor. The DpBHLA stores the histories of executed basic block at BRDT as shown in Fig. 1 and the basic blocks are consists of a sequence of code lines, including branch instructions and register update instructions such as load/arithmetic instructions. Therefore, by indexing the BRDT entry that corresponds to the source register of the branch instruction, data dependency in the indexed entry is easily detected. As a branch outcome is decided based on the content of the branch instruction source register, knowing the register update history allows the predictor to make a more accurate prediction.

2.1 Limits of Dynamic branch history control

Based on the content of the BRDT, the basic block, whose instruction updates the register, can be identified. This knowledge determines how many pre-executed branch instructions are related to the current executing branch instruction. While the related previous branch outcomes are useful in the current branch prediction, unrelated previous branch outcomes are unnecessary. Therefore, after searching the useful branch outcomes, the DpBHLA branch predictor selects the useful BHR bits. While selecting the useful BHR bits, the DpBHLA branch predictor resets the non-selected BHR bits to zero. Fig. 2 shows the modification of some BHR bits based on the BRDT content, and the modified BHR bits are exclusive-or with the instruction program counter bits to access the PHT.

The modified BHR bits which reset some BHR bits to zero and the instruction program counter bits are used as an input to exclusive-or arithmetic. Exclusive-or arithmetic has a property in which the output is always same as the other when one of the inputs is zero. For example, when exclusive-or arithmetic is performed between zero and A, the output is always equal to A, regardless of A's condition. As shown in Fig. 3, some of the output bits that correspond to zero bits at the BHR bits are exactly same as that of the instruction program counter bits. The shaded parts of the PHT index bits in Fig. 3 are equal to the some program counter bits, while the rest part is an outcome of exclusive-or arithmetic between the program counter bits and the modified history.

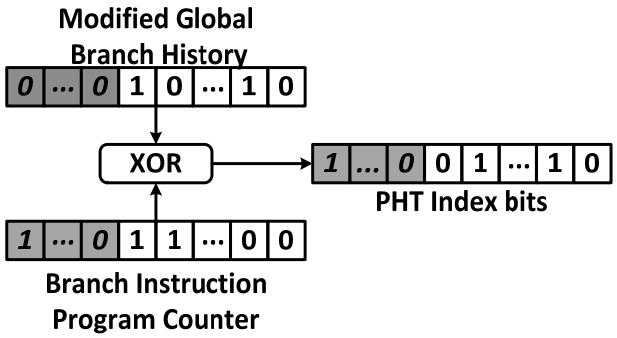

Fig. 3. Branch instruction program counter bits at PHT index bits

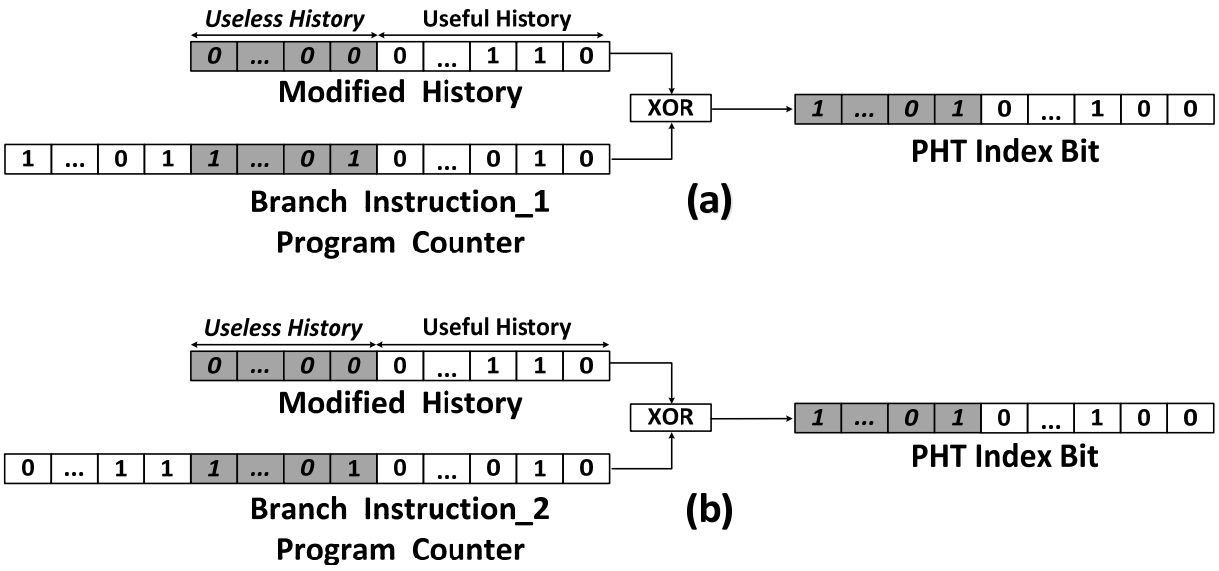

Fig. 4. Same PHT index bit between different branch instructions: (a) Insturction_1, (b) Instruction_2

As mentioned previously, resetting the BHR bits to zero will result in decreased predictor accuracy and inefficient PHT usage since predictors would be biased to certain regions of the PHT. Figs. 4-(a) and (b) show different branches which access to same PHT entry. In Fig. 4, the shaded parts of the PHT index bits are from the corresponding program counter bits, while the rest of the parts are the outcomes of exclusive-or arithmetic between program counter bits and filtered history. In other words, when different branch instructions, whose program counter bits correspond to some BHR bits, which are reset to zero, are the same, these two different branch instructions have a high possibility of accessing the same PHT entry and making a prediction based on the same two-bit counter of the PHT entry. In addition, resetting the useless BHR bits to zero would decrease the number of exclusive-or arithmetic outputs since the input length of exclusive-or arithmetic, which is the program counter bits and useful BHR bits, is decreased. Therefore, decreased exclusive-or arithmetic outputs will use limited entries of the PHT, and therefore, would increase the aliasing of the occurrence rate, which is crucial to accurate branch prediction.

Therefore, in order to overcome these weaknesses and to increase prediction accuracy, the IAaHLM branch predictor, which utilizes the difference of the program counter bits instead of resetting the useless BHR bits to zero, is proposed.

2.2 Methodology

If the branch instructions are said to be different, then their program counter should be different. Even if the program counter bits corresponding to the whole BHR bits are the same, the remaining bits should be different; the proposed structure, the IAaHLM branch predictor, utilizes this aspect. The IAaHLM branch predictor inserts an appropriate length of unused program counter bits to the useless BHR bits, as shown in Fig. 5. Figs. 5-(a) and (b) show different PHT accessing bits by using the same branch instructions but different methods; Fig. 5-(a) uses the DpBHLA method, while Fig. 5-(b) uses the IAaHLM method. While the shaded parts of the PHT index bits in the previous structure (Fig. 5-(a)) are equal to the program counter bits, the shaded part in the proposed structure (Fig. 5-(b)) is different from the program counter bits.

Selecting program counter bits, which will be utilized to substitute the reset BHR bits (useless BHR bits), is very important. This study uses the instruction program counter bits to substitute the useless BHR bits. Starting bits position at program counter corresponds to the MSB of the BHR bits. In addition, the length of the used program counter bits is same as the length of useless BHR bits and these bits normally indicate page numbers as shown in Fig. 6. The main advantage for inserting some program counter bits that stand for page numbers is as follows. The previous structure used the modified BHR bits and instruction program counter bits which are page offset bits as an input to exclusive-or arithmetic. In addition, instructions on same page are separated by the page offset bits while

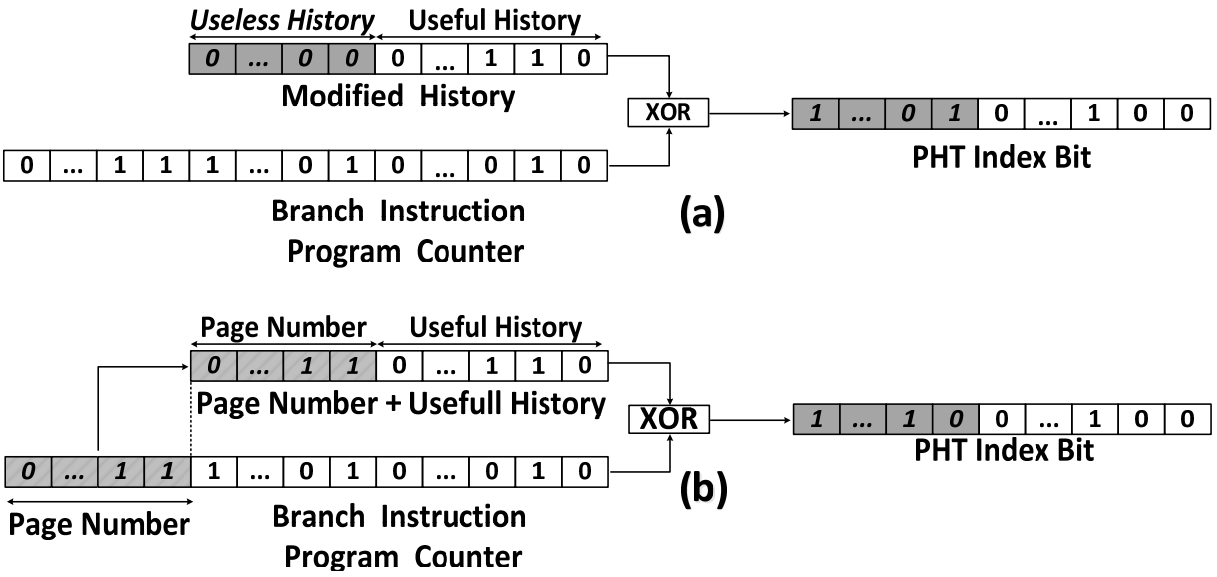

Fig. 5. Different PHT output between DpBHLA and IAaHLM: (a) DpBHLA method (b) IAaHLM method

instructions on different page are separated by page number bits. Since instruction program counter bits which, are used as an input to exclusive-or arithmetic in previous structure, are mostly composed of page offset bits, instructions in different page may have same exclusive-or arithmetic input and this same input will cause aliasing in the PHT. Therefore, the IAaHLM branch predictor is proposed to overcome this weakness. Instead of leaving useless bits in the BHR to zero, the proposed structure change the useless bits in the BHR to page numbers which in result decrease aliasing in the PHT and increase prediction accuracy. Figs. 7-(a) and (b) show different branch instructions accessing different PHT entries using the IAaHLM branch predictor.

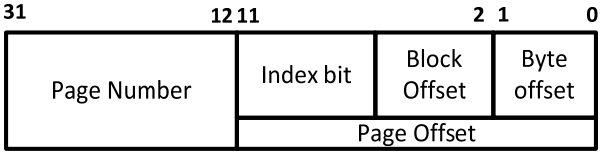

Fig. 6. 32bit program counter description

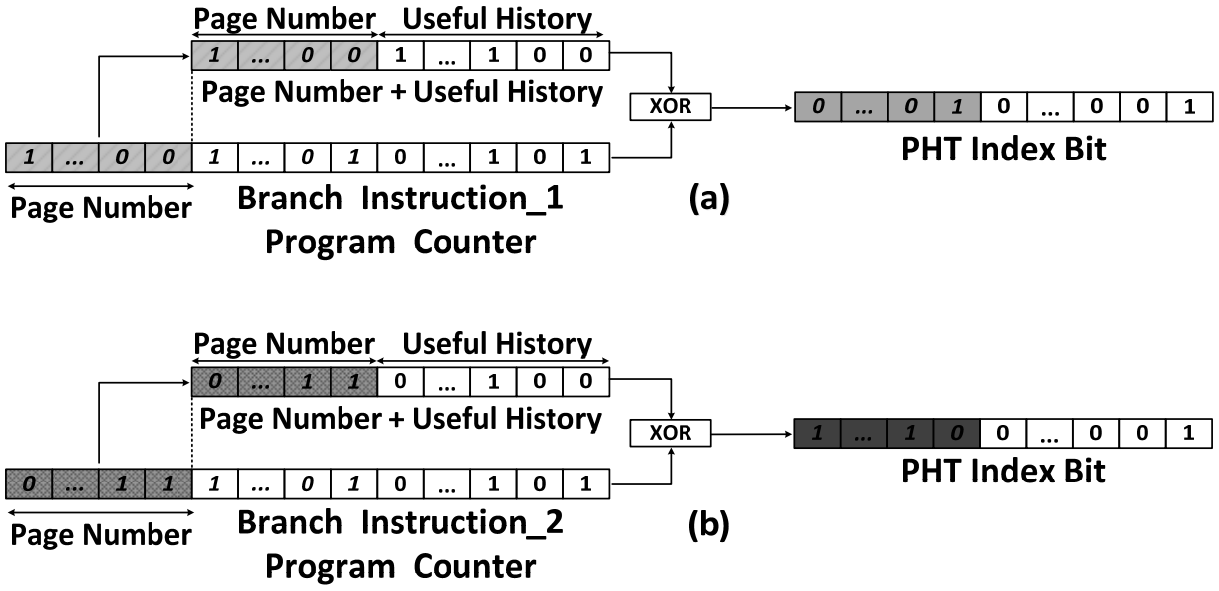

Fig. 7. Different PHT index bits by utilizing Page number bits: (a) Insturction_1 (b) Instruction_2

3 Experimental Results

In order to measure the proposed structure, IAaHLM, SimpleScalar [11], an event-driven simulator, is used. SimpleScalar is an effective simulation tool that allows for not only an instruction level, but also a cycle level. For benchmark programs, the SPEC2000 [12] application suites are used; used benchmarks are shown in Table 1.

Prediction accuracy, instruction per clock (IPC), and aliasing rate is compared among IAaHLM, DpBHLA, and Gshare, which is the standard branch predictor in state of art processors. The aliasing rate is calculated as equation (1).

$$\text{Aliasing rate} = \frac{\text{Total Number of arisedAliasing}}{\text{Sizeof PHT}} \quad (1)$$

The simulation environment is shown in Table 2. The evaluation is performed by varying the PHT entries from 1,024 to 8,196.

Table 1. Benchmark programs description

Benchmark	Lang.	Description
175.vpr	C	FPGA circuit P&R
176.gcc	C	C programming compiler
197.parser	C	Word processing
252.eon	C++	Computer visualization
253.perlbmk	C	Perl programming language
255.vortex	C	Object-oriented database
256.bzip2	C	Compression
300.twolf	C	Place and route simulator

Table 2. Simulation system environment

Benchmark	Description
Fetch queue	4 entry
Fetch, decode width	4 instructions
RUU entries	16 entry
LSQ	8 entry
Funct. (integer)	4 ALUS, 1 Mult/Div
Funct. (floating point)	4 ALUS, 1 Mult/Div
Instruction TLB	64 entry/4k page/30-cyc. .miss
Data TLB	128 entry/4k page/30-cyc. miss
BTB entries	2048 entry
RAS entries	8 entry
Branch misp. penalty	3 cycles
L1 I-Cache	16KB/direct map/32Bline/1cyc,
L1 D-Cache	16KB/ 4-way/32Bline/1 cycle
L2 Cache (unified)	256KB/4-way/64B line/6 cyc.
Memory Latency	First_chunk=18cyc./inter=2cyc.
Memory bus Width	8 byte

While adjusting the BHR bits for accurate branch prediction, the unrelated BHR bits require treatment rather than being reset to zero. If they are reset to zero, aliasing in the PHT will be increased and prediction accuracy will be decreased. Therefore, the IAaHLM branch predictor changes these unrelated BHR bits to some program counter bits which are page number bits. By using page number bits, aliasing will be decreased and therefore branch prediction accuracy will be increased. In addition, increased branch prediction accuracy will finally increase IPC which is the main factor to boost processor performance.

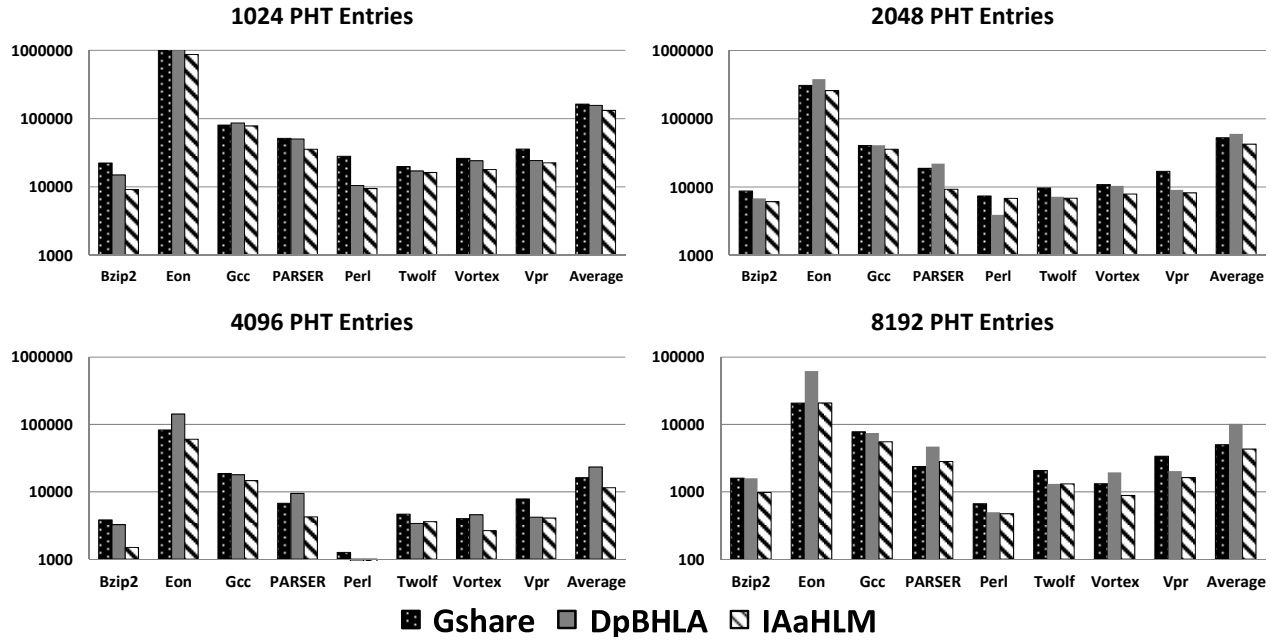

Figure 8. Comparison of aliasing in Gshare, DpBHLA, and IAaHLM

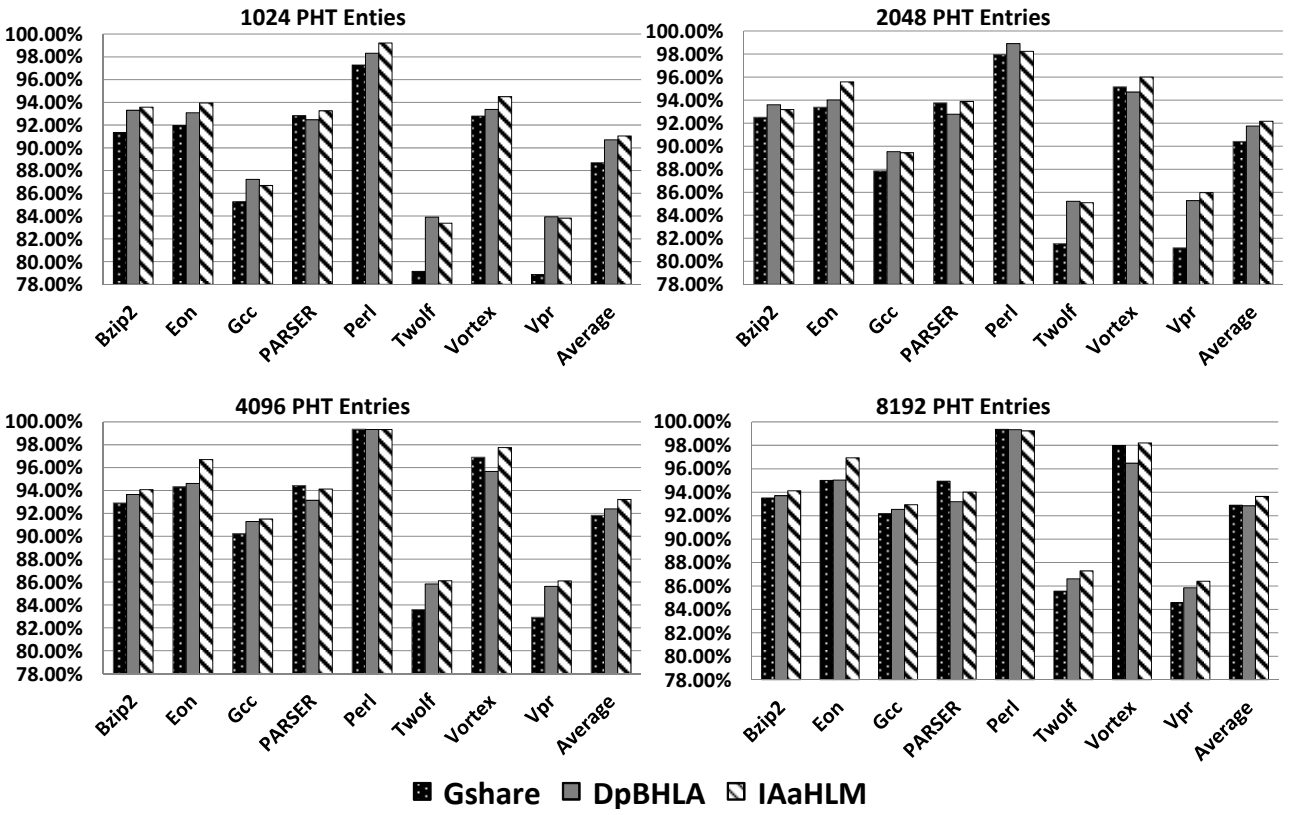

Fig. 9. Branch prediction accuracy in Gshare, DpBHLA, and IAaHLM

The rate of aliasing when varying PHT size is shown in Fig. 8. As in the figure, aliasing happened least in IAaHLM, regardless of the PHT entries. In addition, when the PHT entries were 1,024 and 2,048, aliasing happened most often in Gshare, while DpBHLA has most occurrences of aliasing for the rest the PHT entries. Breen et al. [13] mentioned that the more PHT entries there are, the less aliasing exists, as shown in Fig. 8. This phenomenon is the reason why IAaHLM performance varied by increasing PHT entries.

Fig. 9 shows the prediction accuracy of IAaHLM, DpBHLA, and Gshare. Since IAaHLM had the smallest number of aliasing occurrences, regardless of PHT entries, it predicted the branch outcome more accurately than other branch predictors for most programs, regardless of PHT entries. On average, when the PHT entries were 8,192, IAaHLM prediction accuracy was 93.64%, while the prediction accuracy of DpBHLA was 92.84% and Gshare was 92.89%; further, the smaller the PHT entries were, the more performance differences occurred. This is because, as shown at Fig. 8, aliasing mostly occurred when the PHT was 1,024. In other words, the smaller the PHT entries are, the more aliasing occurs and the more the performance improves.

Fig. 10 shows IPC comparisons of IAaHLM, DpBHLA, and Gshare. Regardless of PHT entries IAaHLM had the greatest IPC, because branch prediction highly affects IPC. On average, when the PHT entries were 8,192, IAaHLM had an IPC of 1.54, DpBHLA had an IPC of 1.52, and Gshare had an IPC of 1.53; the performance difference between IAaHLM and the others increased when PHT entries were smaller than 8,192.

In conclusion, IAaHLM is an efficient branch predictor since it has less aliasing than do DpBHLA and Gshare. In addition, this results in increased branch prediction accuracy and increased IPC.

4 Conclusions

Branch prediction is performed by utilizing branch instruction program counter bits and branch outcome histories. In these histories, some bits are useless in branch prediction, while the others are useful. Using the useful histories is very important. However, the useless histories require careful treatment rather than simply being reset to zero.

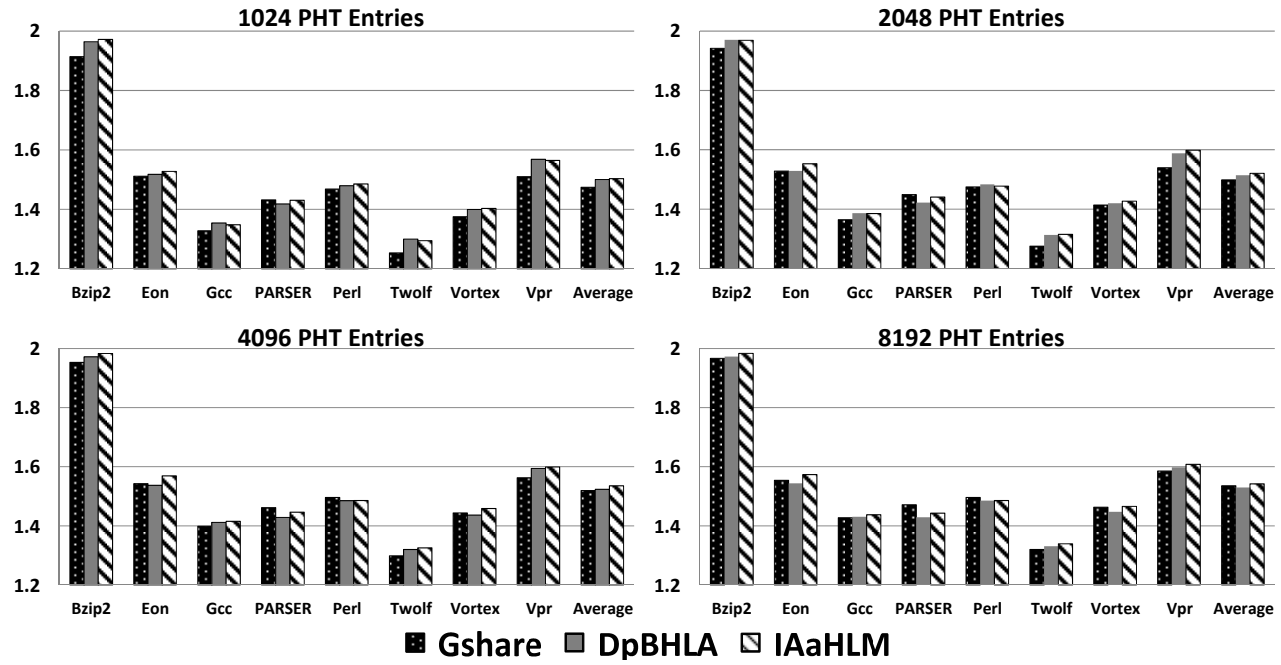

Fig. 10. Comparison of IPC in Gshare, DpBHLA, and IAaHLM

This paper proposes an IAaHLM branch predictor to treat these useless histories. To substitute the useless histories, page numbers at the branch instruction program counter were used. When PHT entries were 8,192, IAaHLM had only 4,302 aliasing occurrences per entry on average, while Gshare had 4,991 and DpBHLA had 10,193. In addition, when the PHT entries were decreased, the aliasing occurrence reduction increased. Due to the reduction of aliasing, branch prediction accuracy was increased. IPC also increased, regardless of PHT entries. When the PHT entries were 4,096, IAaHLM had a branch prediction accuracy of 93.22%, whereas Gshare's was 91.84%. IAaHLM had an IPC of 1.53, while Gshare had an IPC of 1.51 IPC.

References:

[1] J. E. Smith, A study of branch prediction strategies, *in Proceedings of the 8th annual symposium on Computer Architecture*, Minneapolis, Minnesota, USA, pp. 135-148, 1981.

[2] T.-Y. Yeh and Y. N. Patt, Two-level adaptive training branch prediction, *in Proceedings of the 24th annual international symposium on Microarchitecture*, New Mexico, Puerto Rico, pp. 51-61, 1991.

[3] T.-Y. Yeh and Y. N. Patt, A comparison of dynamic branch predictors that use two levels of branch history, *in Proceedings of the 20th annual international symposium on computer architecture*, San Diego, California, USA, pp. 257-266, 1993.

[4] T.-Y. Yeh and Y. N. Patt, Alternative implementations of two-level adaptive branch prediction, *in Proceedings of the 19th annual international symposium on Computer architecture* 1992, Queensland, Australia. p. 124-134, 1992.

[5] S.-T. Pan, K. So and J. T. Rahmeh, Improving the accuracy of dynamic branch prediction using branch correlation, *in Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, Boston, Massachusetts, USA, pp. 76-84, 1992.

[6] M. Evers, S. J. Patel, R. S. Chappell and Y. N. Patt, An analysis of correlation and predictability: what makes two-level branch predictors work, *The 25th Annual International Symposium on Computer Architecture*, Barcelona, Spain, pp. 52-61, 1998.

[7] M. D. Tarlescu, K. B. Theobald and G. R. Gao, Elastic history buffer: a low-cost method to improve branch prediction accuracy. *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Austin, TX, USA, pp. 82-87, 1997.

[8] L. Porter and D. M. Tullsen, Creating artificial global history to improve branch prediction

accuracy, *in Proceedings of the 23rd international conference on Supercomputing*, Yorktown Heights, NY, USA. p. 266-275, 2009.

[9] T. Juan, S. Sanjeevan and J. J. Navarro, Dynamic history-length fitting: a third level of adaptivity for branch prediction. *The 25th Annual International Symposium on Computer Architecture*, Barcelona, Spain, pp. 155-166, 1998.

[10] J. W. Kwak and C. S. Jhon, Dynamic per-branch history length adjustment to improve branch prediction accuracy. *Microprocess. Microsyst.*, vol. 31, pp. 63-76, 2007.

[11] D. Burger, T.M.A. and S. Bennett, Evaluating future micro-processors: the SimpleScalar tool set, 1997.

[12] J. L. Henning, SPEC CPU2000: measuring CPU performance in the New Millennium, *Computer*, vol. 33, pp. 28-35, 2000.

[13] K. C. Breen and D. G. Elliott, Aliasing and anti-aliasing in branch history table prediction SIGARCH Computer Architecture News, vol. 31, pp. 1-4, 2003.