A Novel Meta Predictor Design for Hybrid Branch Prediction

YOUNG JUNG AHN, DAE YON HWANG, YONG SUK LEE, JIN-YOUNG CHOI AND GYUNGHO LEE
The Dept. of Computer Science & Engineering
Korea University
Anam-dong 5-ga-1, Sungbuk-gu, Seoul, 136-701
THE REPUBLIC OF KOREA
yjahn@formal.korea.ac.kr, dyhwang@formal.korea.ac.kr, duchi@korea.ac.kr,
choi@formal.korea.ac.kr, ghlee@korea.ac.kr

Abstract: Recent systems have been paved the way for being high-performance due to the super-pipelining, dynamic scheduling and superscalar processor technologies. The performance of the system is greatly affected by the accuracy of the branch prediction because the overhead of each misprediction has grown due to greater number of instructions per cycle and the deepened pipeline. Hybrid branch prediction is usually used to increase the prediction accuracy on such high-performance systems. Normally hybrid branch prediction uses several branch predictors. A meta-predictor selects which branch predictor should be used corresponding to the program context of the branch instruction instance for the branch prediction. In this paper, we discuss about the saturating counter within meta predictor. The design of the saturating counter which selects a predictor that has high-prediction ratio has brought out the high accuracy of the prediction for the branch predictor.

Key-Words: Branch Prediction, Saturating Counter, Prediction Accuracy, Hybrid Branch Predictor, Meta Predictor.

1 Introduction

There have been many recent studies and increasing efforts to improve the performance of computer system. Exploiting Instruction Level Parallelism (ILP) has been a major means of achieving high-performance computer systems [1, 17, 18]. Deep pipelines, various superscalar methods and many dynamic scheduling algorithms have been utilized for exploiting ILP. In such high performance systems, branch prediction to predict the outcome of a conditional branch has become an increasingly important component in determining overall performance [19, 20]. Without the branch prediction, processor would have to wait until a branch is resolved before the next instruction can enter the fetch stage in the pipeline. The branch predictor attempts to avoid this delay by trying to guess whether the conditional jump is most likely to be "taken" (true branch) or "not-taken" (false branch). The branch that is guessed to be more likely is then fetched and speculatively executed. If it is later detected that the prediction was wrong, then the speculatively executed or partially executed instructions are discarded: Processor starts over with the correct program control flow. Dynamic branch prediction records the history of branch instructions, which means the directions taken by the past instances of a branch instruction, and predicts the direction of a branch instance based on the history. Dynamic branch prediction schemes generally fall into two types. One is self-history-based branch prediction, and the other is correlation-based branch prediction. The self-history-based branch prediction predicts the direction of the current branch instruction with only using the history of past instances of the instruction, i.e. self-history. This scheme may achieve a high prediction accuracy when it's using for the program with lots of loops. The correlation-based branch prediction predicts the direction of a branch using the history of branch instructions in addition to its own history. This can provide better accuracy when a branch direction depends on control flow paths reaching to a particular branch instance.

To take advantage of the both types of branch predictors, one may selects one of the predictors by employing two predictors, one with self-history based prediction and the other with correlation

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. R01-2007-000-20750-0) and by the IT R&D program of MKE/IITE [2009-KI002090].
This paper is an extended version of the paper [28] presented at the WSEAS Int. Conf. on CSECS '09.

based prediction. Such a hybrid prediction has been a choice for high performance computer systems.

A hybrid branch predictor needs a "choice" predictor, also known as a meta predictor to choose one of the branch predictors employed to reflect the current program context of a branch better. The meta predictor predicts which one of the branch predictors employed by utilizing a saturating counter as in a self-history based branch prediction. A saturating counter that is used in a branch predictor for predicting a branch direction decides the direction of branch. While the saturating counter used in the meta predictor decides a branch predictor which is more suitable for predicting the current branch instruction instance per the program context.

A specific saturating counter design adopted for the meta predictor obviously affects branch prediction accuracy, though there have been little research on the design of saturating counter for the meta predictor in a hybrid branch prediction. This paper considers saturating counter design specifically for the meta predictor used for a hybrid branch predictor. Different state transitions from a usual saturating counter may choose a particular branch predictor better and result a higher branch prediction accuracy. The saturating counter design may take a different form per the purpose of its use.

This paper is organized as follows: related works are presented in section 2; we introduce a typical branch prediction technique that could be used mainly by existing systems. In section 3, we introduce the saturating counter of meta predictor of the combining predictor for improving branch prediction accuracy and we analyze our experiment. Finally, we discuss the results of this study and our future work.

2 Related works

A Branch Instruction is predicted by a computer system and has basic a Bimodal and a Correlate feature [21]. For this reason, the Branch Prediction method is mainly used to classify the Bimodal Method, the Two-Level Adaptive Branch Prediction and these modification methods [22]. The definition of the Bimodal of branch prediction determines whether a branch direction is taken or untaken. So, it is easy to predict a pattern of a branch instruction by the previously executed result of the branch instruction. After the main loop statement of high language is compiled, it has a Bimodal tendency. This means that a branch instruction that correlates the branch direction is changed by the mutual relation between the previous branch instructions and the directions, not a pattern. Basically, when a conditional statement of high language is compiled, it has the Correlate tendency.

The Bimodal method of basic branch prediction uses extra Branch History Table for branch prediction. The factors of Branch History Table have n-bit saturating counter of branch instructions after addressing the branch instruction and accessing the Branch History Table that performs branch prediction according to the Saturation Counter of the branch instruction. The Two-Level Adaptive Branch Prediction stores the history of the previous n(counter) branch instruction at a Branch History Register and accesses the Pattern History Table by the value of the Branch History Register at every branch prediction of a branch instruction, so it predicts the branch direction by the value of the Saturating Counter, which stores the Pattern History Table. The Two-Level Adaptive Branch Prediction shows excellent performance of correlate branch instruction. The modification of Two-Level Adaptive Branch Prediction is the Gshare, which solves some Aliasing problems of the branch predictor [22]. The Gshare branch predictor solves some aliasing problems by accessing the Pattern History Table for branch prediction, working an exclusive-or operation with the address of the branch instruction and the value of the Branch History Register. It is a Neural Branch Prediction method for increasing branch accuracy by variable weight according to every branch instruction [23]. Also, it is a hybrid branch predictor that chooses a better appropriate branch predictor between a Bimodal branch predictor and a Correlate branch predictor.

Alpha EV6 and alpha EV8 are microprocessors designed for achieving high-performance. Both two microprocessors use hybrid branch predictor for improving branch accuracy. The following explains alpha EV6 and alpha EV8.

The alpha EV6 implements a sophisticated tournament branch prediction scheme that dynamically chooses between local and global history to predict the direction of a given branch [2]. Attribute of local history is that pattern behavior sometimes correlates with the execution of a single branch at a unique program counter location. And Attribute of global history is that pattern behavior sometimes correlates with the execution of all previous branches. When branch result is an alternating taken/not-taken sequence, the local prediction is very useful. As the branch executes multiple times, it will saturate the prediction counters corresponding to these local history value

and make the prediction correct. When the result of a branch can be inferred from the direction of precious branches, the global prediction is very useful. The global history predictor can learn this pattern with repeated invocations of the two branches.

When a branch instruction retires, the alpha EV6 updates the chooser. The chooser consists of 2-bits saturating counters. If the results of the local and global predictor differ, the alpha EV6 updates the selected meta prediction entry to support the correct predictor.

Branch predictor of 21464 Microprocessor Architecture Global branch history branch predictor tables lead to a phenomenon known as aliasing or interference, in which multiple branch information vectors share the same entry in the predictor table, causing the predictions for two or more branch substreams to intermingle[3, 4, 5]. "De-aliased" global history branch predictors have been recently introduced: the enhanced skewed branch predictor e-gskew, the agree predictor, the bimode predictor and the YAGS predictor [6, 7, 8, 9]. These predictors have been shown to achieve higher prediction accuracy at equivalent hardware complexity than larger "aliased" global history branch predictors such as gshare or GAs [2, 10]. However, hybrid predictors combining a global history predictor and a typical bimodal predictor only indexed with the program counter may deliver higher prediction accuracy than a conventional single branch predictor [11, 2]. Therefore, "de-aliased" branch predictors should be included in hybrid predictors to build efficient branch predictors.

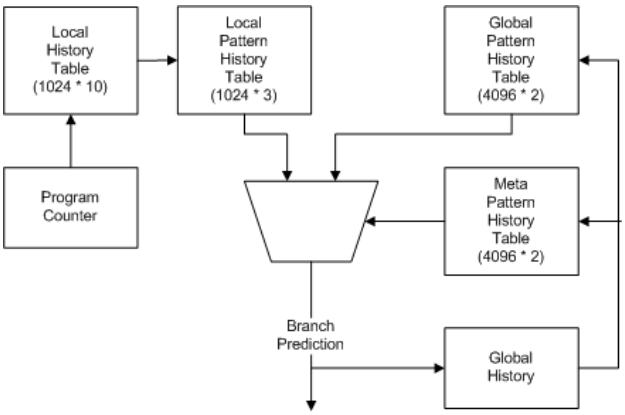

Fig.1 The alpha EV6

Fig.1 is a block diagram of the Branch predictor of 21464 Microprocessor Architecture. The local history branch predictor is on left. The global history branch predictor are meta predictor are on the right.

The EV8 branch predictor is derived from the hybrid skewed branch predictor 2Bc-gskew presented in [12]. 2Bc-gskew combines e-gskew and a bimodal branch predictor [6]. It consists in four identical predictor-table banks, i.e., the three banks from the e-gskew -including a bimodal bank- plus a meta predictor. 2Bc-gskew-pskew combines a bimodal component, a global history register component and a per-address history component [12].

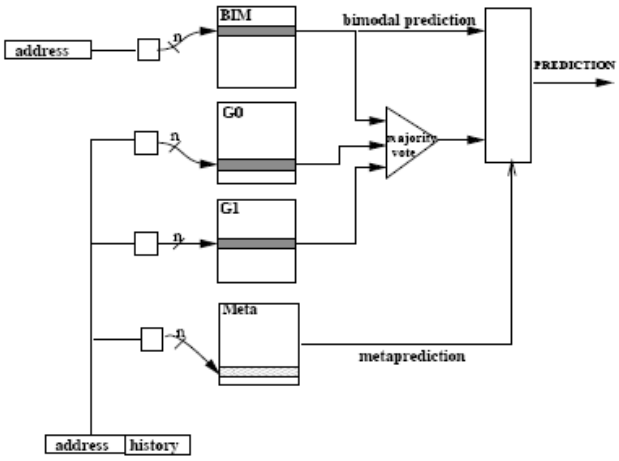

Fig.2 The alpha EV8

Predictor cbp1.5 is a particular instance of a family of predictors which called GPPM, for global-history PPM-like predictors [24]. GPPM predictors feature two tables, a bimodal table and a global table. The bimodal table is indexed with the program counter, and each bimodal entry contains a prediction associated with the branch. The global table consists of several banks. Each bank is indexed with a different global-history length. Each global entry contains a tag for identifying the global-history value owning the entry, and a prediction associated with this global history value. The prediction is given by the longest matching global-history value, or by the bimodal table if there is a tag miss in all the global banks. Predictor cbp1.5 can be viewed as a degraded version of an ideal GPPM predictor which called GPPM-ideal. One can go from GPPM-ideal to cbp1.5 by introducing successive "degradations" corresponding to real-life constraints. They call degradation a modification that increases the number of mispredictions. By quantifying each degradation, one can get insight on the behavior of the application and on potential ways to improve

the predictor.

The TAGE conditional branch predictor stands for Tagged Geometric history length as the O-GEHL predictor [25, 26]. TAGE is derived from Michaud's tagged PPM-like predictor [26]. It relies on a default tagless predictor backed with a plurality of (partially) tagged predictor components indexed using different history lengths for index computation. These history lengths form a geometric series. The prediction is provided either by a tag match on a tagged predictor component or by the default predictor

This allows to efficiently capture correlation on recent branch outcomes as well as on very old branches. The L-TAGE Predictor consisting of a 13-component TAGE predictor combined with a 256-entry loop predictor [27].

Nair has been researching much about the saturating counter for predicting a branch direction decides the direction of branch [13]. This paper proposes one of various 2-bits saturating counter of meta predictor, which provide better prediction accuracy. This saturating counter is used in many branch predictors, but there is no research about the saturating counter that is used in the meta predictor. This saturating counter influenced in the performance of the hybrid branch predictor. Therefore, this paper treats the saturating counter that is used for the meta predictor to choose one of the branch predictors having the highest branch prediction accuracy.

3 Saturating Counter

3.1 Saturating Counter for Predicting

The purpose of a branch predictor is to fetch branch instruction without any delay of the instruction fetch, when it encounters a branch instruction. In order to do this, it predicts the behavior of the branch instruction's next move looking up the past behaviors whether it was taken or not. A common way to design of such predictor employs a saturating counter. The size of the saturating counter is normally 2-bits or 3-bits, even though there is no general size. When the counter is 2-bits, prediction changes the direction if the prediction fails twice. The saturating counter is stored in a table called Pattern History Table and is index by using a part of the branch instruction program counter. The direction of the branch instruction is determined from the value of the saturating counter indexed within the table. Many researches are being done on saturating counters which are used for predicting branch direction. Assume that a saturating counter uses N-bits to represent a state. The branch direction of branch instruction is represented as 'taken' or 'not-taken.' If the size of the saturating counter is N-bits, there can be 0 to 2^N-1 states. If the state is 0 to 2^{N-1}-1, branch predictor estimates the direction as 'not-taken.' The branch predictor will predict the branch direction as 'taken' if the state of saturating counter is 2^{N-1} to 2^N-1. The actual branch direction of a branch instruction can be checked after the instruction executed at the function unit of the microprocessor. The state of saturating counter is transited by the result of the branch instruction. The state of Fig.3 is the history of the last two dynamic instances. If the direction of branch instruction is 'taken', then the state transition is as following. The saturating counter stays at 2^N-1 if its state is 2^N-1. In other words, the saturating counter isn't transited. If the state of Fig.3 is 0, the Fig.3 transit to 2^{N-1}. If the state of Fig.3 is not 0, it would be just increased. When the branch direction of the branch instruction is 'not-taken', the transition is following. If the state of the saturating counter is 0, the counter stays 0. It means saturating counter is not transited. If the state of Fig.3 is 2^N-1, the Fig.3 transit to 2^{N-1}-1. The state of Fig.3 will be decreased if it is not 0. Fig.3 shows that the relation of transition when the saturating counter is 2-bits.

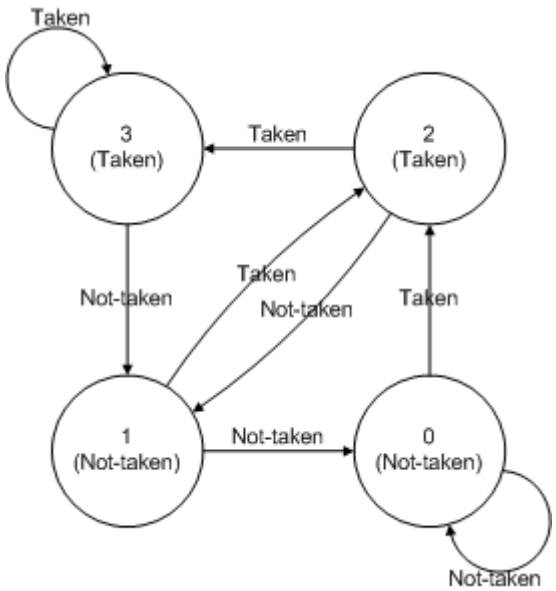

Fig.3 The saturating counter to predict – type 1

When the branch direction of the branch instruction is 'taken', the transition of Fig.4 is following. The state of Fig.4 will be increased if it is not 2^N-1. If

the direction of branch instruction is 'taken', the state of Fig.4 will be decreased if it is not 0. Fig.4 shows that the relation of transition when the saturating counter is 2-bits.

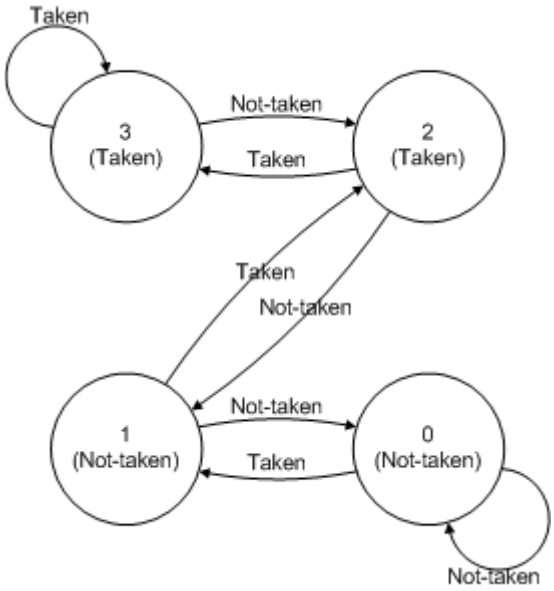

Fig.4 The saturating counter to predict – type 2

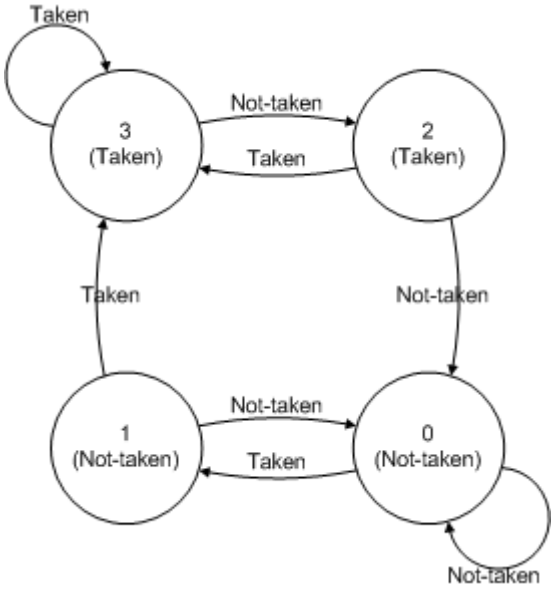

Fig.5 The saturating counter to predict – type 3

When the direction of branch instruction is 'taken', the state transition is of Fig.5 as following. If the state of Fig.5 is 2^{N-1}-1, the Fig.5 transit to 2^{N}-1. If the state of Fig.5 is neither of 2^{N-1}-1 nor 2^{N}-1, it would be just increased. When the branch direction of the branch instruction is 'not-taken', the transition of Fig.5 is following. If the state of Fig.5 is 2^{N-1}, the Fig.5 is transited 0. The state of Fig.5 will be decreased if it is not either 0 or 2^{N-1}. Fig.5 shows that the relation of transition when the saturating counter is 2-bits.

When a branch instruction's direction is determined, there is a high probability that the branching direction of the instruction will be same. Nair proposed a saturating counter based on this idea [13]. To determine the branch direction, therefore, Fig.5 is better than Fig.3 and Fig.4.

3.2 Saturating Counter used in Meta Predictor

Predictors are categorized into four classes depending on the branch prediction scheme. These predictors have been affected the branch prediction accuracy from the characteristics of program and input data. The predictor based on this is called the tournament branch predictor. A tournament branch predictor basically uses more than two predictors. Therefore, which predictor should be used must be determined for each prediction. In order to do this, another predictor exist which called a meta-predictor. A meta-predictor selects the predictor which has the highest accuracy by a saturating counter. For example, assume there are two types of predictors, A and B and we are using a 2-bits saturating counter. Predictor A will be used as long as A predicts correctly for a branch instruction. If A fails twice, then the predictor is changed to B. By doing this, the more accurate predictor will be selected. In other words, which predictor that will be used is depending on the status of the saturating counter.

This paper considers type and size of saturating counter used in the meta predictor. A meta predictor in a hybrid branch predictor should decides a branch predictor which is more suitable for predicting the current branch instruction instance per the program context. In other words, the accuracy of hybrid branch predictor increases only when the meta predictor chooses one of the branch predictors having the highest branch prediction accuracy.

Next is a dynamic method for meta predictor to choose more accurate branch predictor. The two branch predictors for predicting the direction in the hybrid branch predictor will be named A and B. In both A, B, there is a memory to store the weight of each instruction. The weight is increased if the branch predictor for predicting the direction hits, and is decreased if the predictor mispredicts. The role of a meta predictor is to choose the highest weighting branch predictor. However, large amount of memory is needed to use this method.

This paper researches various types of saturating counter used in meta predictor as below. The saturating counter will transit only when the predict direction of A and B are different. If the predict direction of A and B are same, the meta predictor's decision is not changed, so the saturating counter transit when the predict direction of A is different with the predict direction of B. If the direction of the branch predictor which is selected by meta predictor is correct then represent hit, if it's not then represent mis. If a saturating counter's size is N-bits, the counter takes value from 0 to 2^N-1. Meta predictor chooses A when the state of saturating counter is from 0 to 2^{N-1}-1, and chooses B when the state is from 2^{N-1} to 2^N-1. After the branch predictor is executed in the function unit of a microprocessor, the direction predicted by A and B could be checked whether it is a hit or a mis.

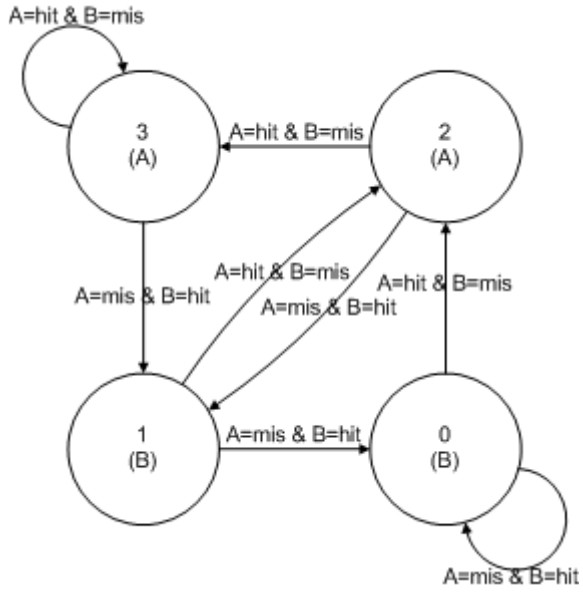

Fig.6 The saturating counter to choose – type 1

If A is a hit, then the state transition of Fig.6 is as following. The saturating counter stays at 2^N-1 if its state is 2^N-1. In other words, the saturating counter isn't transited. If the state of Fig.6 is 0, the Fig.6 transit to 2^{N-1}. If the state of Fig.6 is not 0, it would be just increased. When the branch direction of the branch instruction is mis, the transition is following. If the state of the saturating counter is 0, the counter stays 0. It means saturating counter is not transited. If the state of Fig.6 is 2^N-1, the Fig.6 transit to 2^{N-1}-1. The state of Fig.6 will be decreased if it is not 0. Fig.6 shows that the relation of transition when the saturating counter is 2-bits.

If A is a hit, the Fig.7's state is increased, but if it is a mis, the state would be decreased. Fig.7 illustrates the transition of a 2-bits saturating counter proposed.

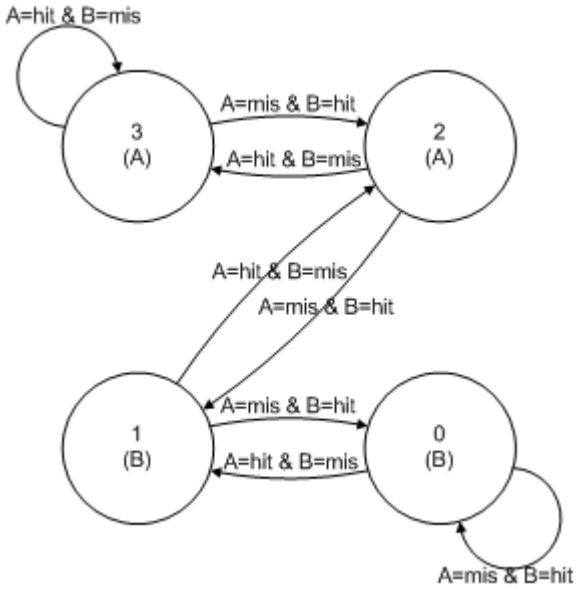

Fig.7 The saturating counter to choose – type 2

When A is a hit, the state transition is of Fig.8 as following. If the state of Fig.8 is 2^{N-1}-1, the Fig.8 transit to 2^N-1. If the state of Fig.8 is neither of 2^{N-1}-1 nor 2^N-1, it would be just increased. When B is a mis, the transition of Fig.8 is following. If the state of Fig.8 is 2^{N-1}, the Fig.8 is transited 0. The state of Fig.8 will be decreased if it is not either 0 or 2^{N-1}. Fig.8 shows that the relation of transition when the saturating counter is 2-bits.

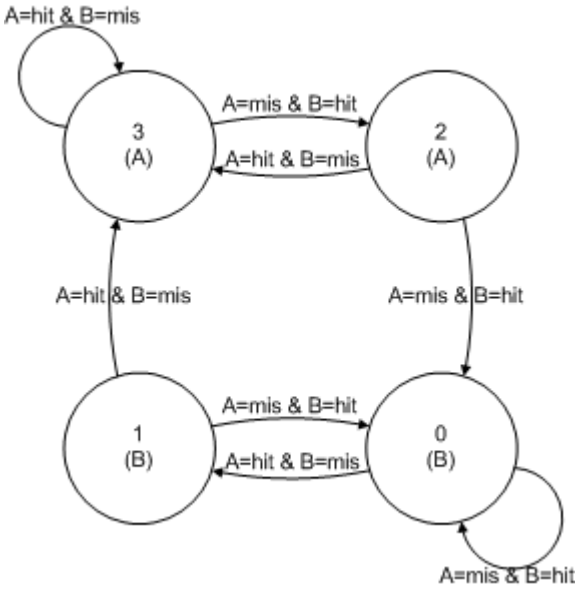

Fig.8 The saturating counter to choose – type 3

The saturating counter showed above transit only when the predict direction of A and B is different. Because if A is hit, B is mispredict, and if A is mispredict, B is hit. Fig.8 shows a conversion of saturating counter to choose the branch predictor.

For experiment, we used a hybrid branch predictor of Alpha ev6 processor [14]. The predictors used in this experiment consist of a local predictor, a global predictor and a meta predictor which selects the predictors to be used.

The local history table holds 10 bits of branch history for up to 1024 branches, indexed by the instruction address. The 21264 uses the 10-bit local history to pick from one of 1024 prediction counters. The local prediction is the most-significant bit of the prediction counter. After branches issue and retire the 21264 inserts the true branch direction in the local history table and updates the referenced counter (using saturating addition) to train the correct prediction.

The global predictor is a 4096 entry table of two-bit saturating counters that is indexed by the global, or path, history of the last twelve branches. The prediction is the most-significant bit of the indexed prediction counter. The 21264 maintains global history with a silo of thirteen branch predictions and the 4096 prediction counters. The silo is backed up and corrected on a mispredict. The 21264 updates the referenced global prediction counter when the branch retires.

The chooser array is 4096 two-bit saturating counters. If the predictions of the local and global predictor differ, the 21264 updates the selected choice prediction entry to support the correct predictor.

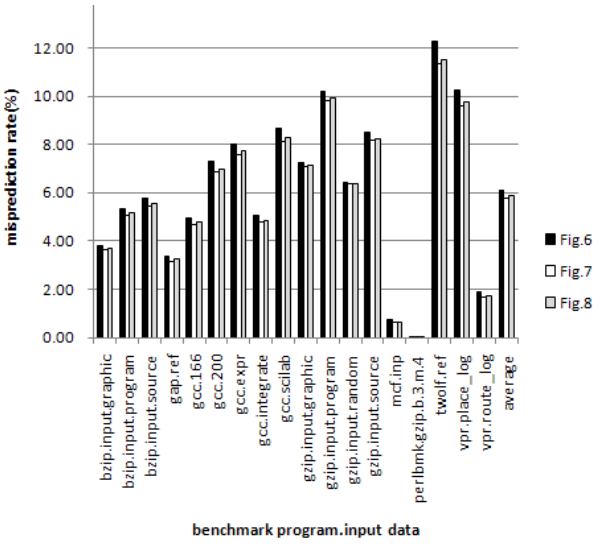

Fig.9 The misprediction rate

The benchmark program of SPEC CINT2000 was used for the experiment [15]. The simulation was performed with 10 billion instructions per reference input data in the SimpleScalar 3.0b [16].

The black bar of Fig.9 shows the misprediction rate when using a saturating counter illustrated in Fig.6 as the meta predictor. The white bar is the misprediction rate when using the proposed saturating counter. The graph shows suitability of the proposed saturating counter as the saturating counter of the meta predictor.

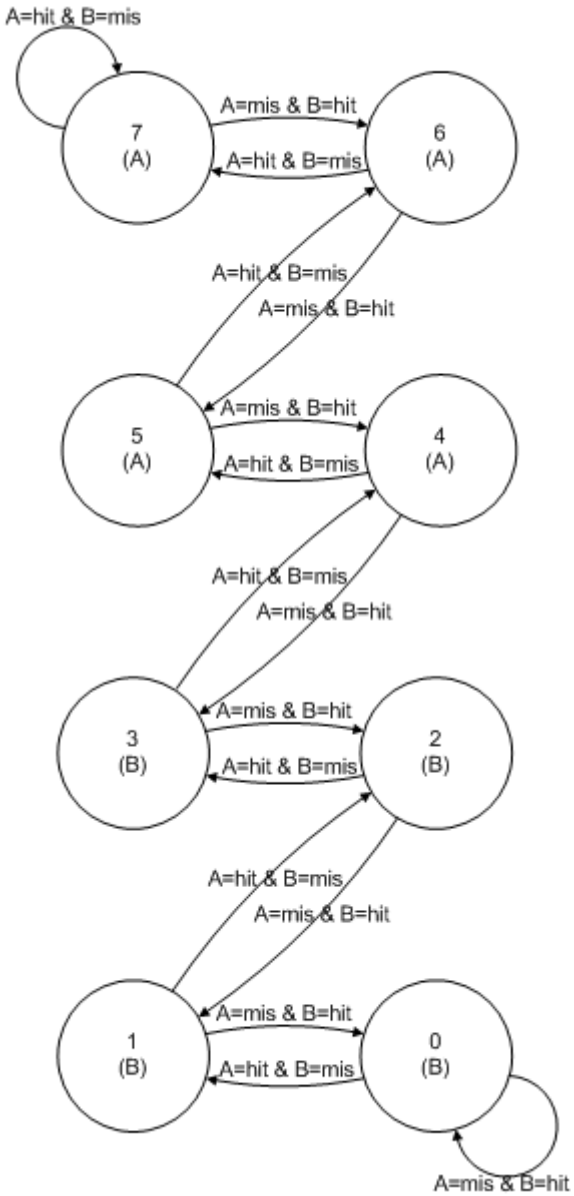

Fig.10 The 3-bits saturating counter to choose

Fig.10 illustrates the transition of a 3-bits saturating counter proposed. If A is a hit, the Fig.10's state is increased, but if it is a mis, the state would be

decreased.

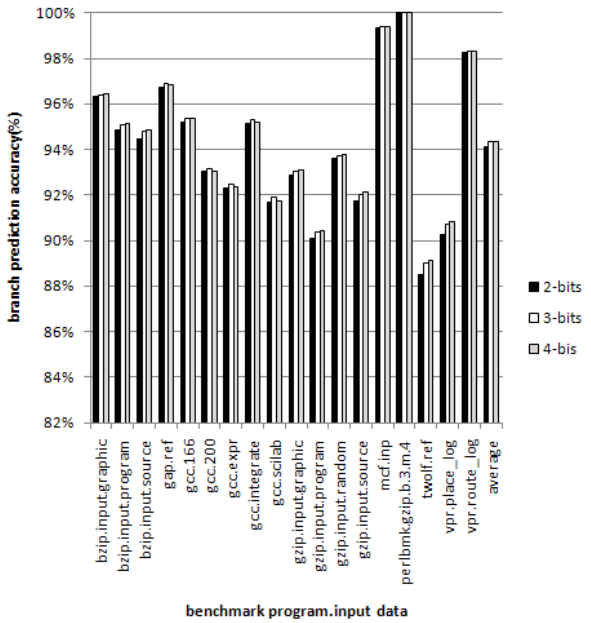

Fig.11 The branch prediction accuracy

This paper suggests size of saturating counter used in meta predictor as below. Fig.11 shows the experimental results of the Alpha EV6 predictor using saturating counters of 2-bits, 3-bits and 4-bits proposes saturating counters. When using a 3-bits saturating counter, the accuracy of branch prediction is 0.2% higher than that with using 2-bits saturating counter. Because it has increased that the state of the saturating counter that stored information of the branch predictor's weight. Branch predictors can choose a more accurate branch predictor because as they can compare with the information about of the branch predictor's weight. However, 3-bits and 4-bits saturating counters have shown similar branch prediction accuracy results, which means that in the benchmark program that was used in this experiment, the saturating counter doesn't need to be bigger than 3-bits.

4 Conclusion

Once the processor fetches a branch instruction, then the branch predictor predicts the branch direction using the past history of the branch instruction. The processor fetches instruction according to program counter, every time. The branch predictor uses the program counter value to map with the entry of the Pattern History Table. There is predictor at the entry of the Pattern History Table. The status affected by past history of the branch instruction observed is in the predictor. According to this status, the branch direction is predicted. The predictor stored within Pattern History Table entry is represented as 2-bits saturating counter of which the status is increased or decreased depending on certain events. An event is the direction of a branch after a branch instruction is resolved. The direction of the branch instruction is determined by the value of the saturating counter indexed. A characteristic of a branch instruction is that it usually maintains the direction once the branch direction is determined. Therefore, it is preferred to change the direction whenever an N-bits saturating predictor fails n times.

A hybrid branch predictor uses two different types of branch predictors to increase the branch predicting accuracy. The hybrid branch predictor consists of two different typed predictor, and meta predictor. A meta predictor is used to determine which direction to follow between the two predicted directions. The performance of the branch predictor can be different depending on the characteristic of program and input-data. Therefore it(?) uses various branch predictors as choice. The branch accuracy of the combining predictor depends on which predictor the meta predictor selects on each branch instance. The meta predictor should select the predictor that has the highest branch accuracy for each branch instance. The meta predictor selects a branch predictor depending on the saturating counter value. In other words, the selection of a branch predictor depends on the saturating counter value of the meta predictor. Saturating counter used for a meta predictor obviously affects branch prediction accuracy. We compare types and sizes of saturating counter used in meta predictor, and shows the which type and sized saturating counter has the best the accuracy by experimental results.

Next is the transition relation of the saturating counter used for a meta predictor which had the highest branch prediction accuracy. The saturating counter should transit only when the directions predicted by the branch predictors are different. When the chosen branch predictor is a hit, the saturating counter will increase one state. And when the branch predictor which was chosen by the meta predictor is a mispredict, the saturating counter will decrease one state. We conclude this paper with suggesting that a saturating counter for meta predictor should be 3-bits, if there is no constraint on memory size.

References

[1] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach, Fourth Edition*, Morgan Kaufmann Publishers, 2007.

[2] S. McFarling, Combining Branch Predictors, Technical Note TN-36, *Digital Western Research Laboratory*, June 1993.

[3] Andre Seznec, Stephen Felix, Venkata Krishnan and Yiannakis Sazeides, Design tradeoffs for the Alpha EV8 conditional branch predictor, *Proceedings of the 29th annual international symposium on Computer architecture (ISCA '02)*, Vol.30, Issue.2, May 2002, pp. 295 – 306.

[4] C. Young, N. Gloy, and M. Smith, A comparative analysis of schemes for correlated branch prediction, *In Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995, pp.276-286.

[5] A. Talcott, M. Nemirovsky, and R. Wood, The influence of branch prediction table interference on branch prediction scheme performance, *In Proceedings of the 3rd Annual International Conference on Parallel Architectures and Compilation Techniques*, 1995, pp.89 – 98.

[6] P. Michaud, A. Seznec, and R. Uhlig, Trading conflict and capacity aliasing in conditional branch predictors, *In Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, June 1997, pp. 292 – 303.

[7] E. Sprangle, R. S. Chappell, M. Alsup, and Y. Patt, The agree predictor: A mechanism for reducing negative branch history interference, *In Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, June 1997, pp 284-291.

[8] C.-C. Lee, I.-C. Chen, and T. Mudge, The bi-mode branch predictor, *In Proceedings of the 30th Annual International Symposium on Microarchitecture*, December 1997, pp.4-13.

[9] A. N. Eden and T. Mudge, The YAGS branch predictor, *In Proceedings of the 31st Annual International Symposium on Microarchitecture*, December 1998, pp.69-77.

[10] T.-Y. Yeh and Y. Patt, Alternative implementations of two-level adaptive branch prediction, *In Proceedings of the 19th Annual International Symposium on Computer Architecture*, Vol.20, Issue.2, May 1992, pp.124-134.

[11] J. Smith, A study of branch prediction strategies, *In Proceedings of the 8th Annual International Symposium on Computer Architecture*, May 1981, pp.135-148.

[12] A. Seznec and P. Michaud, De-aliased hybrid branch predictors, *Technical Report RR-3618*, lnria, February 1999.

[13] R. Nair, Optimal 2-bit Branch Predictors, *IEEE Trans on Computers*, Vol.44, No.5, May 1985, pp.698.702.

[14] R. E. Kessler, The Alpha 21264 Microprocessor, *IEEE Micro*, Vol.19, No.2, March/April 1999, pp.24-36.

[15] SPEC CPU2000 Benchmarks, http://www.specbench.org/

[16] SimpleScalar Tool Suite © 1994-2003 Todd M. Austin, Ph.D. and SimpleScalar, LLC, http://simplescalar.com/

[17] V. Escuder, R. Durán, and R. Rico, Evaluating x86 condition codes impact on superscalar execution, *6th WSEAS International Conference on Systems Theory and Scientific Computation*, August 2006, pp.214-219.

[18] S. Barandagh, H.S.Shahhoseini, and N. Rizvandi, Improving a fixed-point RISC processor by a hybrid adder, *3rd WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA 2004),* February 2004.

[19] H. Vandierendonck, J. Jacquet, B. Nootaert and K. De Bosschere, Formally Modeling Microprocessor Caches and Branch Predictors, *WSEAS Transactions on Computers*, Vol.5, Issue.11, November 2006, pp.2588-2595.

[20] K. Vivekanandarajah, T. Srikanthan, S. Bhattacharya, and P. Kannan, Incorporating Pattern Prediction Technique for Energy Efficient Filter Cache Design, *2002 WSEAS International Conference on Electronics, Control & Signal Processing and 2002 WSEAS International Conference on E-Activities*, December 2002.

[21] Po-Yung Chang, Eric Hao, Tse-Yu Yeh, Yale Patt, Branch Classification: A New Mechanism for Improving Branch Predictor Performance, *In Proceedings of the 27th International Symposium on Microarchitecture*, 1994, pp.22-31.

[22] Tse-Yu Yeh and Yale Patt, Two-level Adaptive Branch Prediction, *Technical Report CSE-TR-117-91, Computer Science and Engineering Division, Department of EECS*, The University of Michigan, November 1991.

[23] Daniel A Jimenez and Calvin Lin, Neural Methods for Dynamic Branch Prediction, *ACM Transitions on Computer Systems*, Vol.20, No.4, November 2002, pp.369-397.

[24] Pierre Michaud. A ppm-like, tag-based predictor. *Journal of Instruction Level*

Parallelism, (http://www.jilp.org/vol7), Vol.7, April 2005.

[25] Andr´e Seznec and Pierre Michaud. A case for (partially)-tagged geometric history length predictors. *Journal of Instruction Level Parallelism* (http://www.jilp.org/vol7), Vol.7, April 2006.

[26] A. Seznec. The o-gehl branch predictor. *In The 1st JILP Championship Branch Prediction Competition (CBP-1), in conjunction with MICRO-37*, 2004.

[27] A. Seznec, A 256 kbits l-tage branch predictor, *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, vol. 9, May 2007.

[28] Young Jung Ahn, Dae Yon Hwang, Yong Suk Lee, Jin-Young Choi and Gyungho Lee, Saturating Counter Design for Meta Predictor in Hybrid Branch Prediction, *Proceedings of the 8th WSEAS Int. Conf. on Circuits, Systems, Electronics, Control & Signal Processing (CSECS '09)*, December 2009, pp.217-221.