International Journal of Energy, Information and Communications
Vol. 2, Issue 1, February 2011

A Branch Predictor with New Recovery Mechanism

Young-Il Cho
Department of Computer Science, University of Suwon
yicho@suwon.ac.kr

Abstract

To improve the performance of wide-issue superscalar processors, it is essential to increase the instruction fetch and issue rate. Removal of control hazard has been put forward as a significant new source of instruction level parallelism for superscalar processors and the conditional branch prediction is an important technique for improving processor performance. Branch mispredictions waste a large number of cycles, inhibit out-of-order execution, and waste power on mis-speculated instructions. Hence, the branch predictor with higher accuracy is necessary for good processor performance. In global-history-based predictors such as gshare and GAg, many mispredictions come from commit-time update of the branch history. Some works on this subject have discussed the need for speculative update of the history and the recovery mechanism for branch misprediction. In this paper, we present a new mechanism for recovering the branch history after mispredictions. The proposed mechanism adds age_counter to the original predictor and doubles the size of BHR (branch history register). The age_counter counts the number of outstanding branches and is used to recover BHR. Simulation results on SimpleScalar tool set and SPEC2000 benchmarks show that gshare and GAg with the proposed recovery mechanism improve the average prediction accuracy by 2.14% and 9.21%, respectively and the average IPC by 8.75% and 18.08%, respectively over original predictors.

Keywords**:** *branch prediction, misprediction, speculative update, branch history, recovery mechanism, outstanding branch, prediction accuracy, IPC*

1. Introduction

Recently, the effort to improve the performance of processors that are used in all areas is one of the main points which make it possible to do high-capacity data in the shortest time. Achieving the highest possible branch prediction accuracies is critical to good processor performance. In wide-issue and deeply-pipelined processors, a single misprediction creates a pipeline bubble that can waste the opportunity to execute more instructions. For this reason, research continues to explore new techniques, especially for predicting branch outcomes [1-3].

Superscalar processors that perform lots of instructions per cycle need an accurate branch predictor so that it may provide useful instructions. If branch mispredictions happen, they can waste a lot of cycles and processor resources because it performs instructions in a wrong path before it does instructions in the correct path. Accordingly, the research on dynamic branch prediction technique, which has higher prediction accuracy, has been continued. Dynamic branch prediction method is a method that improves the accuracy of prediction using the information that is obtained during execution. Among the dynamic branch prediction methods,

two-level adaptive branch predictor which uses the correlation between preceding branch instructions and a predictive target branch is proved to be efficient [1].

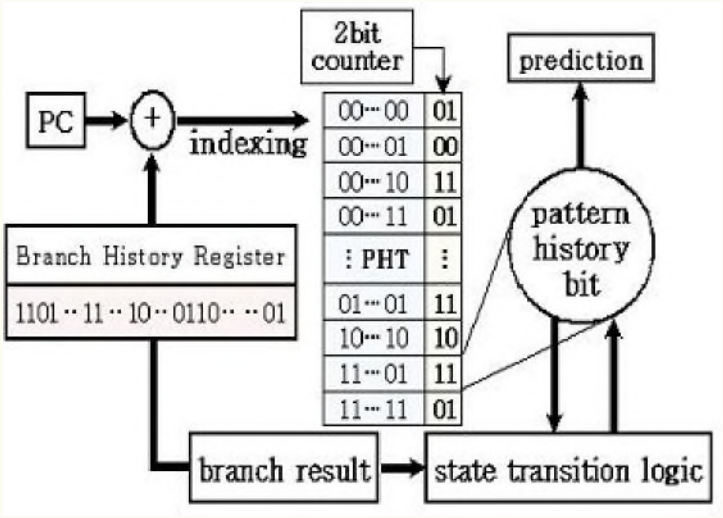


Figure 1 Two-level adaptive branch predictor

At present, two-level branch predictor and its variants are used in various kinds of commercial processors. Current commercial processors use a branch predictor that combines GAs with Agree mechanism, Alpha processor uses a branch predictor that combines the two kinds of two-level branch predictors [4,5]. Recently, hybrid branch predictors which use the local branch history and the global branch history have been researched [6,7].

In pipeline processors, a branch instruction is predicted by the fetch stage and the resolved outcome is determined by the execute stage. In case of a branch misprediction, instructions fetched in wrong path should be invalid and instructions in the correct path should be fetched and executed. There are several cycles between the branch prediction and the resolved outcome. During the cycles, quite a few of branch instructions are fetched and predicted in the processor which has larger pipeline depth and issue. If the branch history has been updated at commit stage, predicted results of in-flight branch instructions shows that it had been predicted by stale branch history. It can especially cause problems with the two-level adaptive branch predictor because these methods are based on the exact branch history of preceding branches [8-10].

The problem can be solved if it is possible for the branch predictor to update the branch history speculatively by the predicted outcome instead of the resolved outcome. If the prediction is correct, the speculative update causes no problem. But, in case of branch misprediction, the polluted branch history must be recovered to the state of just before that prediction because the speculative update inserted wrong information into the branch history. Existing recovery mechanisms used a complicated mechanism in which they store the branch history at queue or reorder buffer. And in case of branch misprediction, it recovers the polluted history into the stored history [12-14].

In this paper, a new mechanism was introduced in order to recover the branch history when branch prediction failed. It adds age_counter which stores the number of unresolved branch instructions and adds SHR which expands the size of BHR (Branch History Register) by two times. When it occurs a branch misprediction, it tries to recover BHR using age_counter.

We can show improvement of prediction accuracy and performance through results of simulation of the SimpleScalar tool set in case of applying the suggested recovery mechanism to existing global-history based branch predictors.

2. Suggested Recovery Mechanism

When a branch misprediction is occurred, existing recovery mechanisms mostly used the QUEUE and required a complicated hardware. The recovery mechanism suggested by this paper is implemented by a simple hardware. In existing mechanisms such as History-based methods and Future-based methods, the size of OBQ (Outstanding Branch Queue) entry has 21 bits (13 bits for storing the content of GHR and 8 bits for tag). Therefore, they are necessary for 420 bits and logic to drive the Queue. But, the suggested mechanism only needs 18 bits (13 bits for SHR and 5 bits for age_counter).

2.1. Branch Prediction and Speculative Update Method

The predictor suggested by this paper has been further equipped with age_counter in order to recover the branch history at a branch misprediction. It also uses SHR (Speculative History Register) instead of GHR (Global History Register).

The age_counter stores the number of outstanding branches which have been fetched and predicted but have not yet updated the branch history as resolved outcomes. If new branch instruction has been fetched, age_counter is incremented because of increasing outstanding branches. In commit stage, if the resolved outcome is same as the predicted result, age_counter is decremented because of decreasing outstanding branches.

In existing predictors, GHR has the branch history which has been updated by the resolved branch outcome at commit stage in order of being fetched. But the SHR includes history which is updated at commit stage and speculative history which is updated at fetch stage.

The Figure 2 shows SHR when age_counter is $\boldsymbol{C}$ which means that the number of outstanding branches is $\boldsymbol{C}$. The range from $\boldsymbol{B_0}$ to $\boldsymbol{B_{C-1}}$ points out the speculative update history of branch instructions which would have been currently executing. The range from $\boldsymbol{B_C}$ to $\boldsymbol{B_{n+m-1}}$ points out the updated history at commit stage. SHR adds m bit for the speculative update which is different from the GHR. At the time of simulation, it has been arranged for the value of m to have the same as one of n.

In the modified gshare branch predictor, which is allowable of the speculative update of GHR, the prediction of conditional branch instructions as shown in the Figure 3 does an 'exclusive-OR' program counter of current branch with the outcome of last n branch instructions in SHR(in case the number of PHT is 2^n), and indexes PHT using that result. If

the value of selected PHT entry (saturating counter) is more than ***2(two)***, the branch is predicted by 'taken'. If the value is less than ***1(one),*** the branch is predicted by 'not taken'. As a conditional branch instructions are predicted, the value of age_counter is increased by ***1(one)*** and this points out that the number of unresolved branch instructions has been increased by ***1(one)***. The speculative update is updated by the outcome of branch prediction. That is, SHR is shifted by ***1(one)*** bit to the left and the prediction outcome would be stored in B_0.

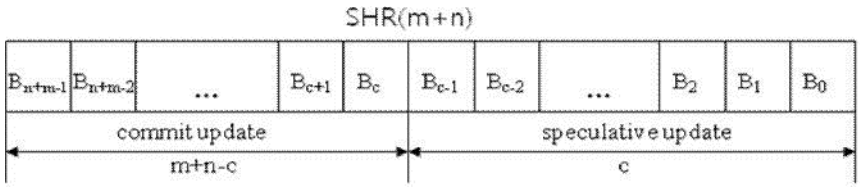


Figure 2 SHR(Speculative History Register)

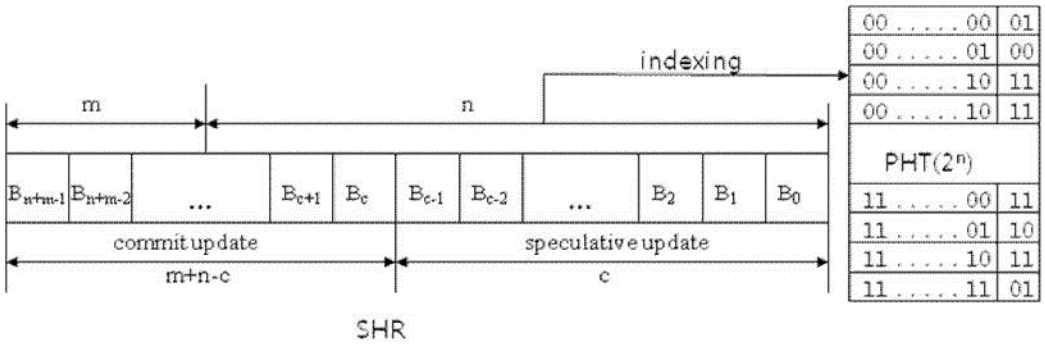


Figure 3 Recovery Mechanism (gshare)

2.2. Recovery Mechanism for Branch Misprediction

If the prediction value is same as the value of resolved outcome, the processor would normally operate. If the prediction value is not same as the value of resolved outcome, SHR must be recovered. If it is not recovered, the polluted SHR would be used continuously to predict following branches and this leads to low prediction accuracy eventually. Accordingly, it is necessary that the polluted SHR should be recovered to the history just prior to prediction of the mispredicted branch.

The recovery mechanism suggested in this paper uses age counter and recovers SHR to the history just prior to prediction of the mispredicted branch. In the Figure 2, B_C points to the resolved outcome of a branch instruction which was committed lastly. When the resolved outcome of the branch instruction has been obtained, it should be compared to $_{BC-1}$ which is the speculative updated outcome of the branch instruction. If two outcomes are same, the processor would proceed normally because the prediction is correct. At this time, age_counter is decreased by ***1(one)***. It points out that the number of unresolved branch instructions in branch history is decreased. However, if two outcomes are different, it has been turned out to

be branch misprediction and branch histories from B_{C-2} to B_0 in SHR were polluted as predicted results of a wrong path. Accordingly, SHR should be shifted to the right by ***C-1*** bits and recovered to the state just prior to prediction of the branch instruction which has been mispredicted. Subsequent histories which has been fetched and updated speculatively after the mispredicted branch instruction should be nullified and the resolved outcome of mispredicted branch should be stored. At this time, the value of age_counter was reset to *0(zero)* and this points out that there remain no unresolved branch instructions.

3. Performance Measurement and Analysis

3.1. Experiment Environment

For the performance measurement we have modified bpred and sim-outorder in order to apply the suggested branch misprediction recovery to gshare and GAg predictor, which is allowable for the speculative update in SimpleScalar tool set [13] that is the cycle level simulator of superscalar processors. Table 1 shows a machine parameter to the structure of a simulated processor. The outcome was based on 8-issue processor with gshare and GAg branch predictors with 1K~8K entry PHT. BTB (Branch Target Buffer) of all branch predictors has been fixed in 512 sets 4-way. Also, it has been used of 128KB L1 Data Cache, 512KB L1 instruction Cache and 1MB unified L2 Cache. Table 2 describes SPEC2000 benchmarks [15] used in simulation and input data. The input data used ref input. The number of executed instructions of benchmarks in order to save time for simulation is limited to 200 million instructions.

3.2. Performance Measurement

We measured the prediction accuracy and performance of original gshare, original GAg, gshare+recovery which applies the proposed recovery mechanism to gshare and GAg+recovery which applies to the proposed recovery mechanism to GAg for 1K, 2K, 4K and 8K entry PHT.

The proposed recovery mechanism has a simple hardware in comparison with existing recovery mechanisms. But the prediction accuracy and the performance of predictors with the proposed recovery mechanism are proved to improve in comparison with those of existing recovery mechanisms.

Figure 4 shows the prediction accuracy of GAg, gshare, GAg+recovery and gshare+recovery for PHTs with 1K, 2K, 4K and 8K entry. GAg+recovery is GAg branch predictor with the proposed recovery mechanism. Also, gshare+recovery is gshare branch predictor with the proposed recovery mechanism.

The prediction accuracy of Gag+recovery is improved by 8.89% (1K entry PHT) in minimum, 9.63% (8K entry PHT) in maximum and 9.21% in average in comparison with that of Gag. As the number of PHT entry increases, it turns out to improve the prediction accuracy more.

Table 1. Machine parameter of simulation processor

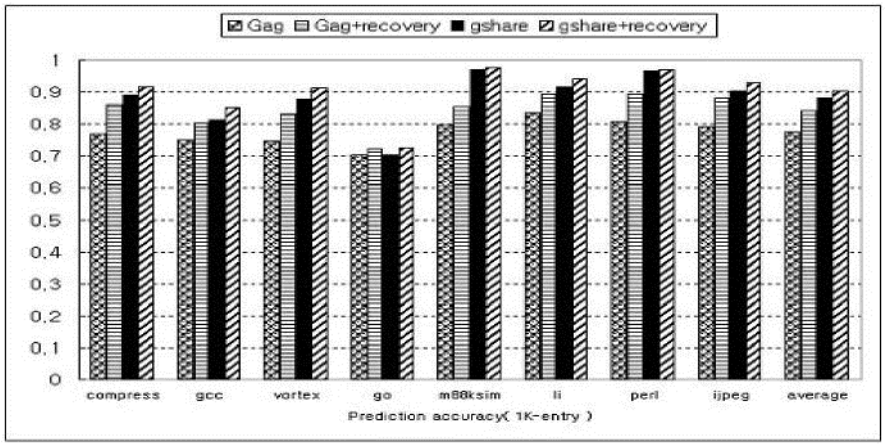
classification	Factor	value	others
Processor Core	RUU size LSQ size Fetch width Decode width Issue width Commit width Functional units	128 entries 64 entries 8 instructions/cycle 8 instructions/cycle 8 instructions/cycle	Instruction window Load store queue In order In order Out of order In order ( ) is latency
Memory	Memory ports Memory access latency	2 ports, 8byte bus first_chunk(18), inter_chunk(2)	( ) is latency
Branch predictor	gshar e GAg	(1K/2K/4K/8K) entries (1K/2K/4K/8K) entries 512 sets, 4 way	( ) is PHT entry size
Cache	L1 data cache L1 instruction cache L2 unified cache	128K, 32B block, 4 way, LRU, 1 cycle latency 512K, 32B block, d-map, LRU, 1 cycle latency 1M, 64B block, 4 way, LRU, 6 cycle latency	

Table 2. Benchmarks & input data

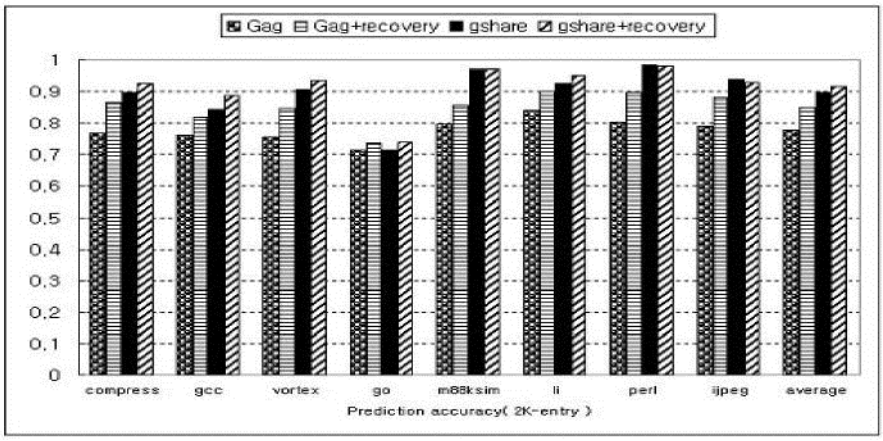
benchmark	Input data	instructions (million)	remark
gcc	cccp.i	200	The GNU C compiler version 2.5.3.
li	train.lsp	183	Xlisp interpreter.
vortex	persons.250	200	An object oriented database.
go	50 9 2stone9.in	200	An internationally ranked go playing program.
m88ksim	dcrand.lit	200	A chip simulator for the Motorola 88100 microprocessor.
compress	100000 e 2231	200	An in-memory version of the common UNIX utility.
perl	primes.pl	200	An interpreter for the Perl language.
ijpeg	penguin.ppm	200	Image compression/decompression on in memory images.

The prediction accuracy of gshare+recovery is improved by 1.9% (8K entry PHT) in minimum and 2.65% (1K entry PHT) in maximum and 9.21% in average in comparison with that of gshare. As the number of PHT entry decreases, it turns out to improve the prediction accuracy more.

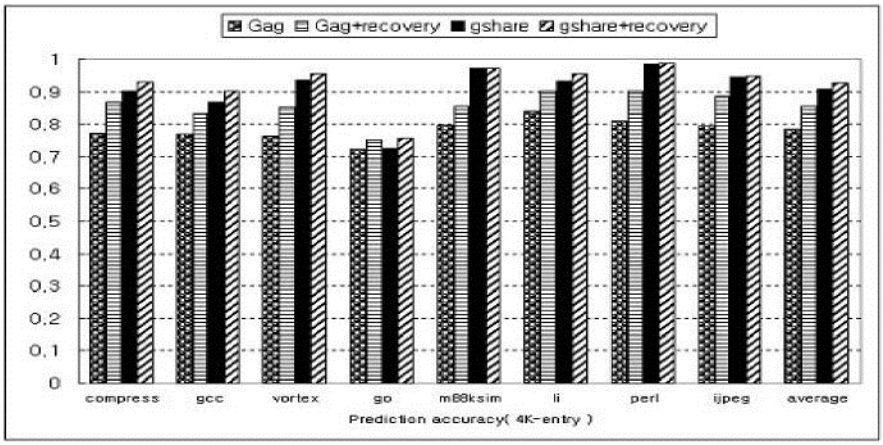
The prediction accuracy of GAg+recovery has been improved more than that of gshare+recovery because GAg uses only GHR when it indexes PHT but gshare uses GHR and the address of branch instruction. It is considered to be a reason that gshare has a less effects on the GHR than GAg. Furthermore, as the change of prediction accuracy according to the change of the number of PHT entries has been measured in Figure 4-e, there had been no great change even when the number of PHT entries increase. Therefore, the recovery mechanism turned out that it has no effect on the number of PHT entry.



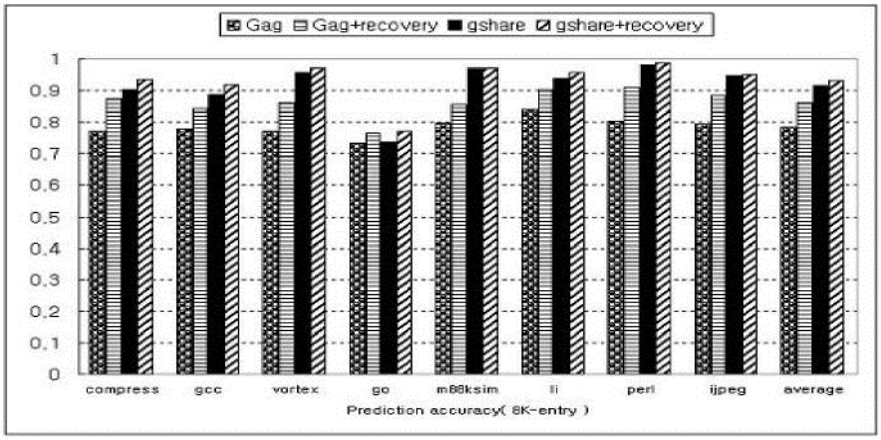
(a) Prediction accuracy (1K entry PHT)



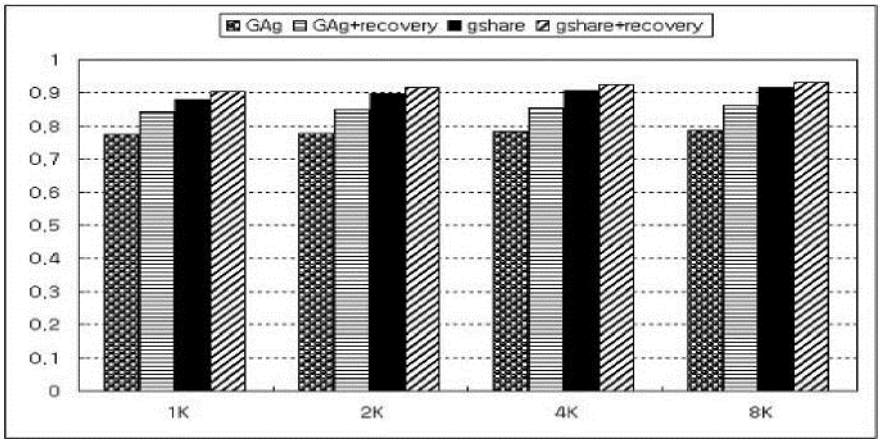
(b) Prediction accuracy (2K entry PHT)



(c) Prediction accuracy (4K entry PHT)



(d) Prediction accuracy (8K entry PHT)



(e) Prediction accuracy according to the change of the number of PHT entries

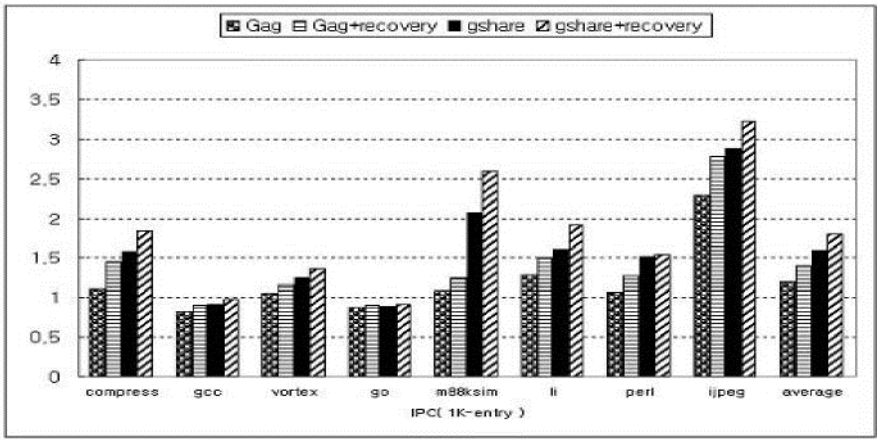
Figure 4. Prediction accuracy of GAg, gshare, GAg+recovery and gshare+recovery

Figure 5 shows the performance of GAg, gshare, GAg+recovery and gshare+recovery for PHTs with 1K, 2K, 4K and 8K entry through IPC (Instruction Per Cycle).

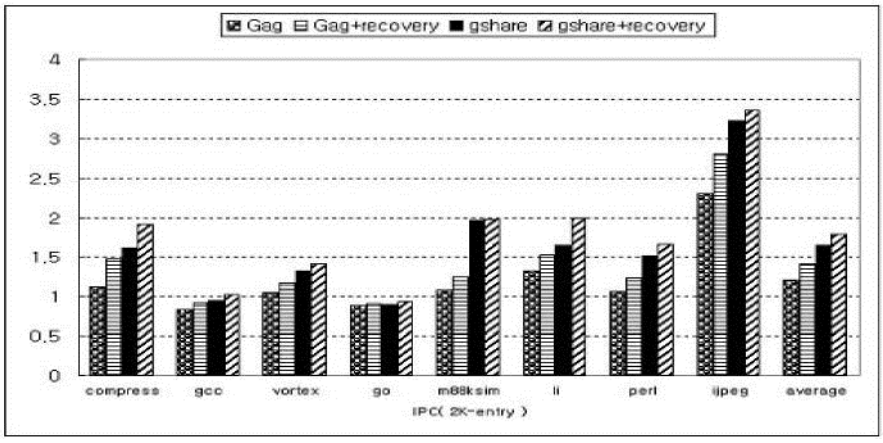
The performance of Gag+recovery is improved by 17% (2K entry PHT) in minimum, 20.2% (8K entry PHT) in maximum and 18.8% in average in comparison with that of Gag. As the number of PHT entry increases, it also turns out to improve the performance more.

The performance of gshare+recovery is improved by 6.3% (8K entry PHT) in minimum and 13.09% (1K entry PHT) in maximum and 8.75% in average in comparison with that of gshare. It is thought that GAg+recovery improves the prediction accuracy better than gshare+recovery and it has caused the performance of processor to be improved more better.

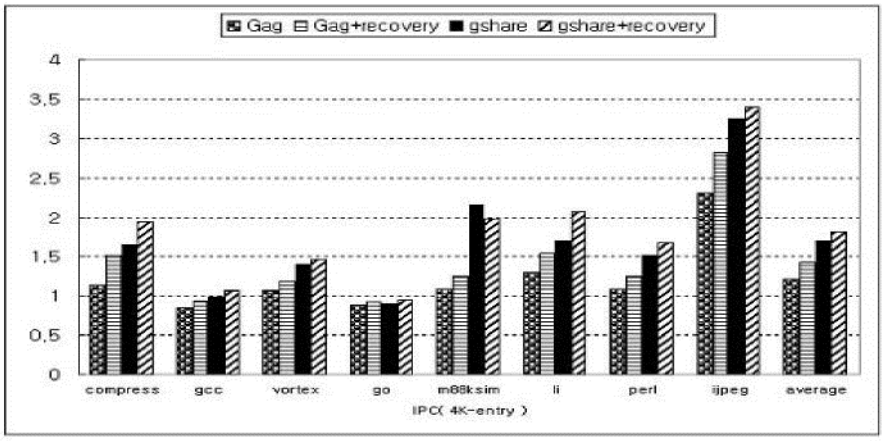
Furthermore, as the change of performance according to the change of the number of PHT entries has been measured in Figure 5-e, there had been no great change even when the number of PHT entries increase. Therefore, the recovery mechanism also turned out that it has no effect on the number of PHT entry.



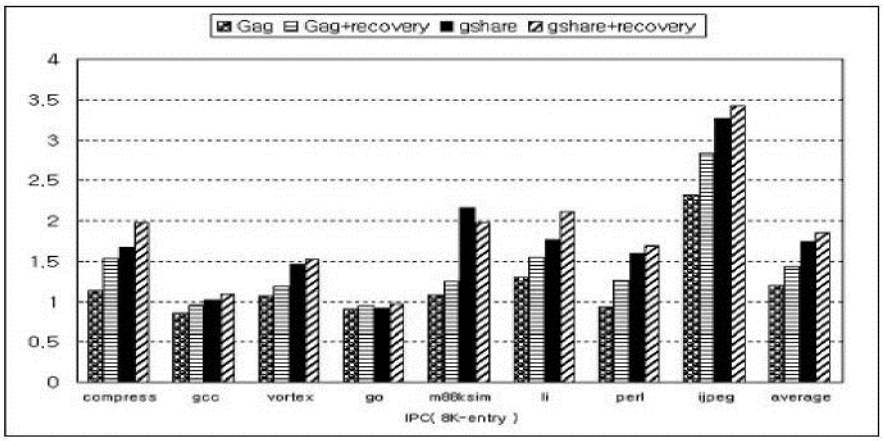
(a) IPC (1K entry PHT)



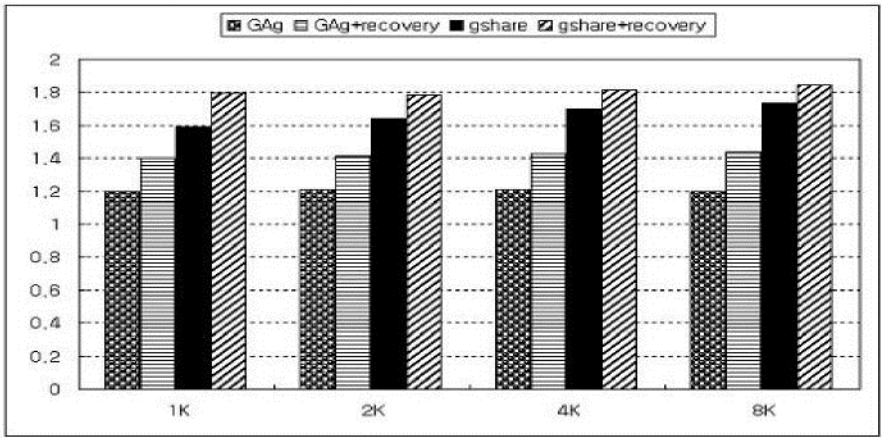
(b) IPC (2K entry PHT)



(c) IPC (4K entry PHT)



(d) IPC (8K entry PHT)



(c) Performace according to the change of the number of PHT entries

Figure 5. Performace of GAg, gshare, GAg+recovery and gshare+recovery

4. Conclusion

In this paper, we presented a simple mechanism through which GHR is recovered to the history just prior to branch prediction after a branch misprediction. Existing recovery mechanisms required a complicated hardware such as QUEUE. But the proposed mechanism could get same effect without almost additional hardware.

We could recognize effects of improvement of the processor performance as well as improvement in the prediction accuracy from the result of simulation in SimpleScalar tool set when the proposed recovery mechanism was applied to GAg and gshare. As the proposed recovery mechanism had been applied to GAg predictor, it has turned out that the prediction accuracy was improved by 9.21% in average and 18.08% of IPC in average. Also, as it had been applied to gshare predictor the prediction accuracy has been improved by 8.75% of IPC in average.

References

[1] 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, May 1992, pp. 124-34.
[2] Hyesoon Kim, J A Joao, O Mutlu, et al. "Virtual Program Counter (VPC) Prediction: Very Low Cost Indirect Branch Prediction Using Conditional Branch Prediction Hardware", IEEE Transactions on Computers, Sep. 2009 pp 1153-1170
[3] Nadav Levison, Shlomo Weiss "Low Power Branch Prediction for Embedded Application Processors", ISLPED 10 Volume: 1, Pages: 67-72 Aug. 2010
[4] K. Diefendorff(1998) "K7 challenges Intel", Microprocessor Report, Oct. 1998, pp. 1, 6-11.
[5] R. E. Kessler, E. J. McLellan, and D. A. Webb "The Alpha 21264 microprocessor architecture", in Proceedings of the 1998 International Conference on Computer Design, Oct. 1998, pp. 90-95.
[6] G.H. Loh, D.S. Henry "Predicting Conditional Branches with Fusion-based Hybrid Predictors", PACT2002, Sep. 2002 pp 395-405.
[7] Z. Lu, J. Lach, M. Stan, and K. Skadron "Alloyed Branch History: Combining Global and Local Branch History for Robust Performance", International Journal of Parallel Programming, Kluwer, volume 31, number 2, Apr. 2003.
[8] A. R. Talcott, W. Yamamoto, M. J. Serrano, R. C. Wood, and M. Nemirovsky "The impact of unresolved branches on branch prediction scheme performance", in Proceedings of the 21st Annual International Symposium on Computer Architecture, Apr. 1994, pp. 12-21.
[9] E. Hao, P.-Y. Chang, and Y. Patt "The effect of speculatively updating branch history on branch prediction accuracy, revisited", in Proceedings of the 27th Annual International Symposium on Microarchitecture, Nov. 1994, pp. 228-32.
[10] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt(1998) "An analysis of correlation and predictability: What makes two-level branch predictors work", in Proceedings of the 25th Annual International Symposium on Computer Architecture, June 1998, pp. 52-61.
[11] K. Skadron, and M. Martonosi "Speculative Updates of Local and Global Branch History : A Quantitative Analysis", JILP Vol. 2, Jan. 2000.
[12] S. Jourdan, J. Stark, T.-H. Hsing, and Y. N. Patt(1997) "Recovery requirements of branch prediction storage structures in the presence of mispredicted-path execution", International Journal of Parallel Programming, vol. 25, Oct 1997, pp. 363-83.
[13] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark(1998) "Improving prediction for procedure returns with return-address-stack repair mechanisms," in Proceedings of the 31st Annual ACM/IEEE International

Symposium on Microarchitecture, pp. 259-71, Dec.
[14] D. Burger, T. M. Austin, and S. Bennett "Evaluating future microprocessors: the SimpleScalar tool set", Tech. Report TR-1308, University of Wisconsin-Madison Computer Sciences Department, July 2000.
[15] The Standard Performance Evaluation Corporation "SPEC2000 Benchmarks", WWW site: http://www.specbench.org/, Dec. 2000.

Authors

Young-Il Cho received B.S., M.S. and Ph.D. in Electronical Engneering from Hanyang University, Seoul Korea in 1980, 1982 and 1985. Currently, he is a professor at Department of Computer Science, University of Suwon. His current research interests include High Performance Microarchitecture, Sensor Network, Optimized Compiler and RFID.