Employing both inter-branch and intra-branch correlation to improve the accuracy of branch prediction for superscalar processors

Meng-chou Chang* Ting-yu Chiu Chih-pei Chang
Department of Electrical Engineering
Chang Gung University, Tao-Yuan, Taiwan
Email: mchang@mail.cgu.edu.tw*

Abstract

Today's superscalar processors use branch prediction to reduce the influence of control hazards. Conventional two-level branch predictors make predictions based on either intra-branch correlation or inter-branch correlation. In the paper, the authors proposed two new branch predictors, called PGXg and PGAg, which make predictions by employing both intra-branch correlation and inter-branch correction. It is shown that the proposed branch predictors can achieve a better cost-performance ratio than conventional two-level branch predictors, such as PAg and gshare.

Keyword: superscalar processor, branch prediction, speculative execution, two-level branch predictor

1. Introduction

A superscalar processor has multiple functional units, allowing more than one instruction to be executed in parallel. However, the performance of a superscalar processor can not be improved unlimitedly by just providing enough functional units because the data and control dependences in a program add many constraints on the execution order of instructions. Most of today's superscalar processors use branch prediction and speculative execution to alleviate the effects of conditional branches. When a superscalar processor fetches a branch instruction, it will predict the outcome of the branch and continue to execute the following instructions along the predicted path. The speculative results of pending instructions will be buffered in a dedicated hardware, such as the reorder buffer [15]. When the outcome of the predicted branch become resolved, the speculative results will be committed if the prediction is proven to be correct; otherwise, the speculative results will be squashed and the processor will restart instruction fetching and execution from the misprediction point.

Branch predictions can be made in static or dynamic way. Static prediction schemes [5,9,12,14] predict the direction of a branch at compile time, while dynamic prediction schemes [1-4, 6-8, 10-11, 13-14, 16-18] predict the direction of a branch at run time. In general, dynamic prediction schemes are able to achieve higher prediction accuracy than static prediction schemes. Smith [14] proposed a dynamic branch prediction scheme that uses a table of 2-bit saturating up-down counters to predict the directions of branches. Each branch instruction is mapped via its address to a counter in the table. Whenever the result of a branch is resolved, its associated counter is updated according to the outcome of the branch. The counter is incremented by 1 if the outcome of the branch is taken, and is decremented by 1 if non-taken. When the processor fetches a branch, it predicts the result of the branch according to the value of the associated counter. If the most significant bit of the value is 1, the branch is predicted as taken; otherwise, the branch is predicted as non-taken.

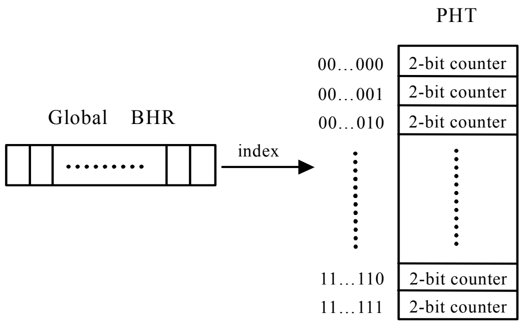

Figure 1. Structure of the branch predictor GAg.

Yeh and Patt [18] proposed the two-level branch prediction scheme. A two-level branch predictor is mainly composed of two components, BHR (branch history register) and PHT (pattern history table). BHR is used to record the outcomes of the most recently executed branches, and PHT is used to keep track of the most likely direction of a branch when a particular pattern is encountered in BHR. Two types of two-level branch predictors, GAg and PAg, are shown in Figure 1 and Figure 2. As shown in Figure 1, the branch predictor GAg has only one BHR, the global BHR, which records the outcomes of all branches. The content of the global BHR represents the global branch history, which is used as an index to access the corresponding saturating counter in PHT. As shown in Figure 2, the branch predictor PAg has a table of local BHRs, and the previous outcomes of a particular branch are recorded in its corresponding local BHR. The content of the local BHR represents the local branch history, which is used as an index to access the corresponding saturating counter in PHT. McFarling [12] proposed the branch predictor gshare, a variation of the global-history branch predictor. As shown in Figure 3, the difference between gshare and GAg is that in gshare the global history is XORed with the branch address to form the index to PHT. It has been shown that gshare can achieve higher prediction accuracy because the addressing scheme of gshare can reduce the possibility of branch interferences in PHT.

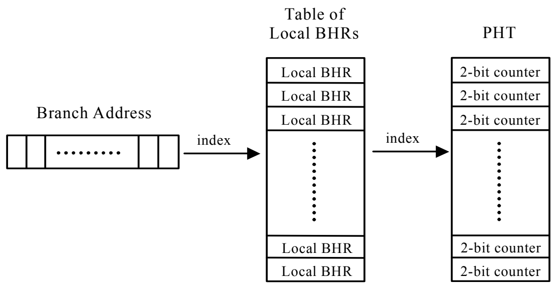

Figure 2. Structure of the branch predictor PAg.

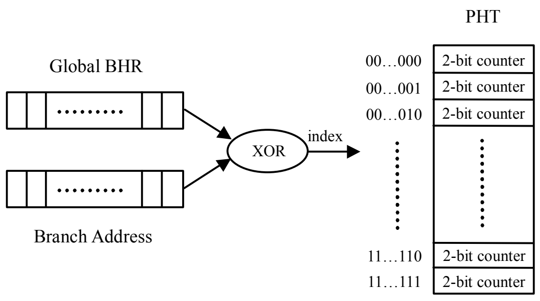

Figure 3. Structure of the branch predictor gshare.

In the paper, we proposed two new branch predictors, PGAg and PGXg, which can achieve higher branch prediction accuracy than conventional branch predictors, such as PAg and gshare.

2. Importance of accurate branch prediction

Branch instructions and their target labels divide a program into basic blocks. A basic block is composed of a straight-line code sequence with no branches in except to the entry and no branches out except at the exit. Let β denote the average dynamic branch frequency, then the average basic block size (i.e., the average number of instructions in a basic block) can be estimated as:

$$\begin{aligned}&\textit{Average basic block size}\\&=\frac{\textit{The number of total dynamic instructions}}{\textit{The number of dynamic branch instructions}}\\&=\frac{\textit{The number of total dynamic instructions}}{(\textit{The number of total dynamic instructions})\times\beta}\\&=\frac{1}{\beta}\end{aligned}$$

For typical MIPS programs the average dynamic branch frequency is often between *15%* and *25%*, so the average block size is between *4* and *6.67*. If a superscalar processor does not support branch prediction, it has to stall to wait for the outcome of the next branch every *4* to *6.67* instructions before it can continue to execute the following instructions in the next block. Since the size of a basic block is very small, the available instruction-level parallelism will be little, leading to underutilization of processor pipelines.

If a superscalar processor supports branch prediction and speculative execution, it can execute instructions across pending branches. Therefore, the processor can find more instructions to fill its pipelines, and a greater amount of instruction-level parallelism can be exploited. Let ρ denote the accuracy of branch prediction, the average code size between two branch mispredictions, denoted by λ, can be estimated as:

$$\begin{aligned}&\textit{Average code size between two branch mispredictions } \lambda\\&=\frac{\textit{The number of total dynamic instructions}}{\textit{The number of mispredicted branches}}\\&=\frac{\textit{The number of total dynamic instructions}}{(\textit{The number of total dynamic instructions})}\\&\times\frac{1}{(\textit{branch frequency})\times(\textit{misprediction rate})}\\&=\frac{1}{\beta\times(1-\rho)}\end{aligned}$$

Let branch frequency β = 20%, then

$\lambda = 83$	if ρ=94%
$\lambda = 100$	if ρ=95%
$\lambda = 125$	if ρ=96%
$\lambda = 167$	if ρ=97%
$\lambda = 250$	if ρ=98%

It is seen that the average code size between two branch mispredictions λ increases rapidly with the increase of branch prediction accuracy ρ. For example, λ has a improvement of 25% when ρ is improved from 95% to 96%. Thus, the code segment in which a superscalar processor can exploit instruction-level parallelism without branch hazards is enlarged significantly even with a small improvement on the branch prediction accuracy. Moreover, the misprediction penalty is proportional to the misprediction rate, so improving the branch prediction accuracy can effectively reduce the misprediction penalty.

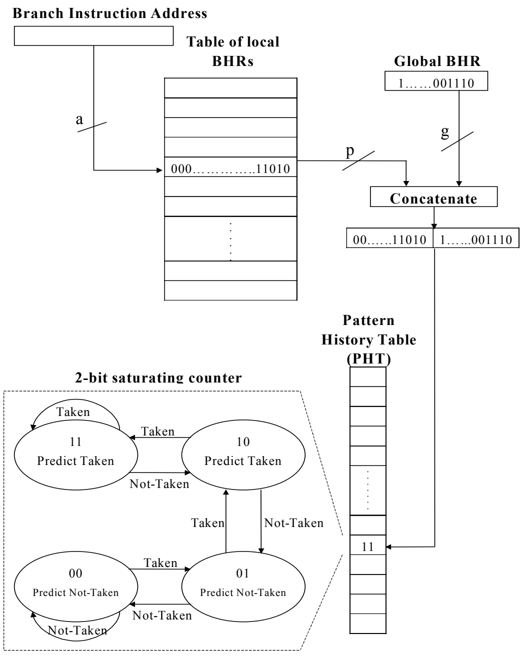

Figure 4. Structure of the branch predictor PGAg.

3. The branch predictors PGAg and PGXg

Conventional two-level branch predictors make predictions either based on inter-branch correlation or based on intra-branch correlation. However, the behavior of some branches in a program can not be well predicted by just employing one type of correlation.

The new branch predictors, PGAg and PGXg, use both inter-branch and intra-branch correlation to make predictions. Figure 4 shows the structure of the branch predictor PGAg. PGAg is composed of a global BHR, a table of local BHRs and a PHT. The global BHR records the global history of all branches, and the local history of a particular branch is recorded in its associated local BHR. The *g*-bit global branch history and the *p*-bit local branch history are concatenated to index the corresponding 2-bit saturating counter in PHT. In the way, the indexed counter records the branch tendency for a particular combination of the global history pattern and the local history pattern.

The structure of the branch predictor PGXg is shown in Figure 5. The structure of PGXg is very similar to that of PGAg. The only difference is that the *g*-bit global branch history is first XORed with the branch address before it is concatenated with the *p*-bit local branch history. The operation of XOR might reduce the possibility of branch interferences in the pattern history table.

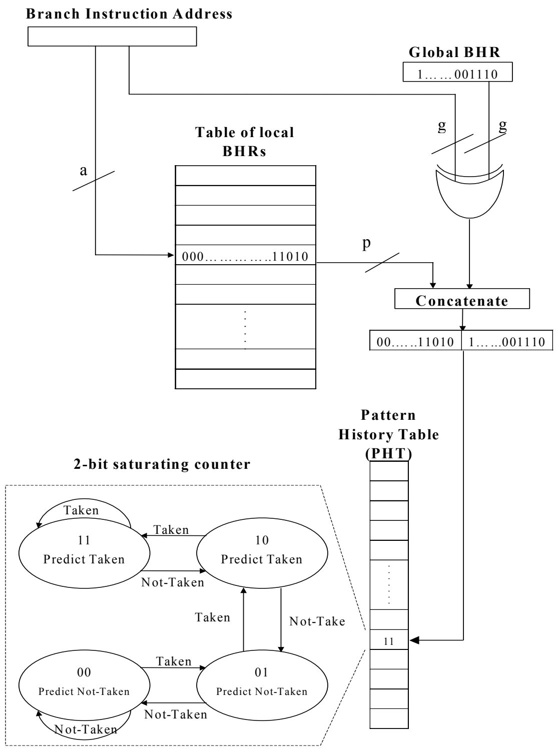

Figure 5. Structure of the branch predictor PGXg.

Both of the branch predictors PGAg and PGXg employ the combination of global and local branch history to access the PHT. Therefore, they predict the outcome of a branch based on both of the inter-branch and intra-branch correlation.

4. Simulation results

In this section, we compare the performance of four two-level branch predictors, PAg, gshare, PGAg, and PGXg. PAg makes predictions based on intra-branch correlation, and gshare make predictions based on inter-branch correlation. PGAg and PGXg make predictions by utilizing both inter-branch and intra-branch correlation.

In the experiment, five benchmarks from SPEC95int benchmark suit are used. The five benchmarks include *gcc, perl, m88ksim, vortex,* and *li*. The brief descriptions of the benchmarks are given in Table 1. All benchmarks are simulated until finish or for over 10000000 conditional branches. The statistics of simulation are summarized in Table 2.

Table 1. Benchmarks used in the experiment

Benchmark	Description
gcc	Based on the GNU C compiler version 2.5.3.
perl	An interpreter for the Perl language.
m88ksim	A chip simulator for the Motorola 88100 microprocessor.
vortex	An object oriented database.
li	Xlisp interpreter.

Table 2. Statistics of simulation in the experiment.

Benchmark	Static instruction count	Dynamic instruction count	Dynamic branch instruction count
gcc	8,474	63,885,066	10,461,104
perl	1,626	524,283,858	77,676,741
m88ksim	935	743,722,373	100,000,000
vortex	4,569	1,742,632,325	200,000,000
li	847	1,427,260,543	200,000,000

Table 3. The formulas of predictor size.

Predictor type	Predictor size (bits)
PAg	$g+2\times2^{g}$
gshare	$l\times e+2\times2^{l}$
LGshare	$l\times e+g+2\times2^{(l+g)}$
PGAg	
PGXg	

Figure 6 shows the performance comparisons of four branch predictors. The x-axis of the figure is the predictor size, which represents the hardware cost to implement the branch predictor. The formulas of predictor size are given in Table 3, where *g* denotes the length of the global BHR, *l* denotes the length of the local BHR, and *e* denotes the number of local BHRs in the predictor. From Figure 6, it is seen that PGXg has the best performance over other predictors.

When the predictor size is fixed at about 4k-byte, the misprediction rates of PGXg, PGAg, gshare, and PAg are 3.3%, 3.45%, 4.0%, and 4.4%, respectively. As the branch misprediction penalty is proportional to branch misprediction rate, about 25% of branch misprediction penalty can be deleted when PGXg replaces the PAg branch prediction scheme, and about 17.5% of branch misprediction penalty can be deleted when PGXg replaces the gshare branch prediction scheme.

When the predictor size is fixed at about 16k-byte, the misprediction rates of PGXg, PGAg, gshare, and PAg are 2.6%, 2.75%, 3.0%, and 3.4%, respectively. About 23.5% of branch misprediction penalty can be deleted when PGXg replaces the PAg branch prediction scheme, and about 13.3% of branch misprediction penalty can be deleted when PGXg replaces the gshare branch prediction scheme.

5. Conclusion

As the pipeline depth and issue rate of superscalar processors increase, the importance of an excellent branch predictor becomes more crucial to delivering the potential performance of a wise-issue, deep pipelined processor. We have proposed two branch predictors, PGAg and PGXg, which employ both the intra-branch correlation and the inter-branch correlation to improve the branch prediction accuracy. These two branch predictors can achieve better performance over conventional branch predictors such as PAg and gshare.

Acknowledgement

This work is supported by the National Science Council, Taiwan, under Grant no. NSC92-2220-E-182-005.

References

[1] M.-C. Chang and Y.-W. Chou, "Branch prediction using both global and local history information," *IEE Proceedings-Computers and Digital Techniques*, Vol. 149, No. 2, March 2002, pp. 33-38.

[2] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y.N. Patt, "Branch classification: a new mechanism for improving branch predictor performance," in *Proceedings of the 27th Annual ACM/IEEE International Symposium on Microarchitecture*, 1994, pp. 22-31

[3] P.-Y. Chang, E. Hao, and Y.N. Patt, "Alternative implementations of hybrid branch predictors," in *Proceedings of the 28th ACM/IEEE Annual International Symposium on Microarchitecture*, 1995, pp. 252-257

[4] P.-Y. Chang, M. Evers, and Y.N. Patt, "Improving branch prediction accuracy by reducing pattern history table interference," in *Proceedings of the 1996 International Conference on Parallel Architecture and*

Compilation Techniques, 1996, pp. 48-57

[5] B.L. Deitrich, B.C. Chen, and W.W. Hwu, "Improving static branch predictor in a compiler," in *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, 1998, pp. 214-221

[6] A.N. Eden and T. Mudge, "The YAGS branch prediction scheme," in *Proceedings of the 31st Annual IEEE/ACM International Symposium Microarchitecture*, 1998, pp. 69-77

[7] C. Egan, *Dynamic Branch Prediction in High Performance Superscalar Processors*, PhD Dissertation, Department of Computer Science, Faculty of Engineering and Information Sciences, University of Hertfordshire, United Kingdom, August 2000

[8] 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," in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998, pp. 52-61

[9] J.A. Fisher and S.M. Freudenberfer, "Predicting conditional branch directions from previous runs of a program," in *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992, Boston, pp. 85-95

[10] C.-C. Lee, I.K. Chen, and T.N. Mudge, "The bi-mode branch predictor," in *Proceedings of the 30th Annual IEEE/ACM International Symposium Microarchitecture*, 1997, pp. 4-13

[11] S. Manne, A. Klauser, and D. Grunwald, "Branch prediction using selective branch inversion," in *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, 1999, pp. 48-56

[12] S. McFarling and J. Hennessy, "Reducing the cost of branches," in *Proceedings of the 13th Annual International Symposium on Computer Architecture,* 1986, pp. 396-403

[13] S. McFarling, *Combining branch predictors*, Technical Report TN-36, Digital Western Research Laboratory, June 1993.

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

[15] J.E. Smith and A.R. Pleszkun, "Implementing precise interrupts in pipelined processors," *IEEE Transactions on Computers*, Vol. 37, No. 5, 1988, pp.562-573

[16] E. Sprangle, R.S. Chappell, M. Alsup, and Y.N. Patt, "The agree predictor: a mechanism for reducing negative branch history interference," in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 284-291

[17] L.N. Vintan and C. Egan, "Extending correlation in branch prediction schemes," in *Proceedings of the 25th EUROMICRO Conference*, 1999, pp. 441-448

[18] 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, pp. 124-134

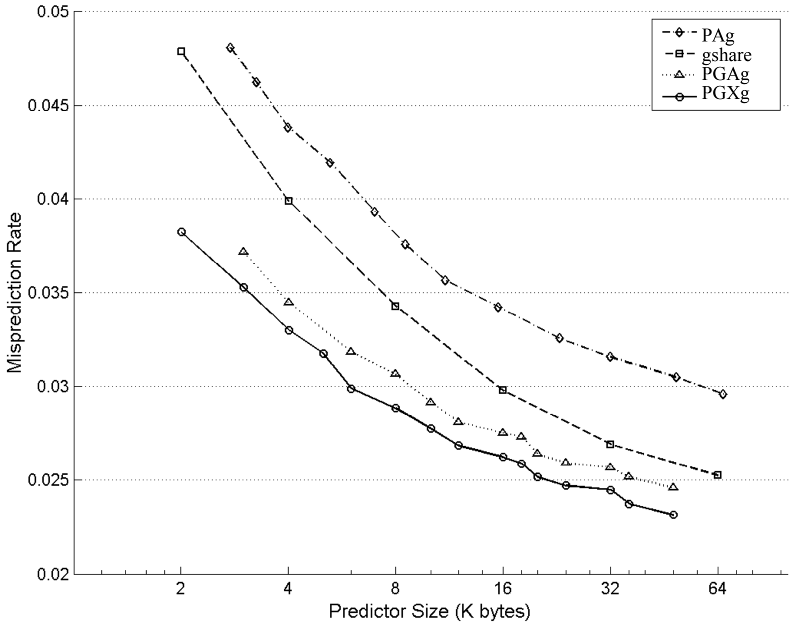

Figure 6. Performance comparison of four branch predictors.