BRANCH DIRECTION PREDICTION WITH VOTE PREDICTOR

Hung-Ching Lin Chang-Jiu Chen

Department of Computer Science and Information Engineering

National Chiao Tung University

Email: hclin@csie.nctu.edu.tw Email:cjchen@csie.nctu.edu.tw

ABSTRACT

As modern microprocessors employ deeper pipelines and issue multiple instructions per cycle, they are becoming increasingly dependent on accurate branch prediction. Up to now, various branch prediction strategies have been proposed. However, from the experiment we find that there is no one branch predictor is good for all benchmarks.

With the factor above, we propose the vote predictor, which combine three different branch predictors to make prediction. From the simulation results, it shows that the vote predictor outperforms its component predictors that make prediction alone. With 4K entries of pattern history table, the vote predictor increases the prediction accuracy with one of its most accurate component predictor from 1% to 2%

1. INTRODUCTION

In the search for higher levels of performance, recent machine designs have made use of increasing degrees of instruction level parallelism (ILP). For example, both superscalar and superpipelining techniques are becoming increasingly popular. In modern pipelined superscalar processor, multiple instructions can be fetched and processed concurrently. Out-of-order instruction issue is effective only when instructions can be supplied at a sufficient rate to keep the execution unit busy. But when a branch instruction occurs, it will interrupt the steady flow of instructions stream.

In order to resolve such problems, many branch prediction schemes had been proposed in these years. In traditionally, the branch prediction can be divided into two parts, one is the branch direction prediction, and another is the branch target prediction. This paper will focuses on direction prediction.

Hardware branch prediction strategies have been studied extensively. The most well-known technique, referred to as bimod branch prediction. More recent works have shown that significantly utilizing more branch history can make more accurate predictions, such as local and global branch predictors. In order to get the more accurate branch prediction, the combining branch predictor was proposed [1]. This approach is shown to provide more accurate predictions than any one predictor alone.

In this paper, we will present a new branch predictor called as “vote predictor”. The experimental results show that the vote predictor can obtain more accurate direction prediction than all his component predictors.

The rest of this paper is organized as follows. In chapter 2, we will describe the related work of branch prediction conceptually. In chapter 3, we will describe the structure of the vote predictor in detail, including its performance and hardware cost. In chapter 4, we will present the experimental results and provide some analysis. In chapter 5, the last chapter, we will make conclusions and suggest some future work

2. RELATED WORK

2.1 Bimod Predictor

The bimod predictor was first proposed by Lee and Smith [6]. It use 2-bit saturation counters to record the history outcomes of every branch instruction. If the counter is bigger than or equal to 2, the branch is predicted to jump; otherwise the branch is predicted to fall-through.

2.2 Gshare and Path-based branch predictor

Several variations of the two-level branch predictor have been proposed. The gshare was proposed by McFarling [1]. In order to utilize the PHT more efficiently the n-bit global history register is combined with m lower-order bits of the program counter value using exclusive-OR. The m-bit resultant is then indexed into the patter history table. The advantages are that it can accommodate a longer global history for a given table size, and spread the accesses to the pattern history table more evenly among the entries.

Nair [3] proposed using a "path history" instead of a pattern history to index the PHT. This has the advantage of being able to represent the path but has the disadvantage that information from fewer branches can be captured in the history.

2.3 Skewed Branch Predictor

The skewed branch predictor was used to reduce interference in 2-level branch predictor [5]. It splits the original PHT into several PHT banks, and indexes them by different and independent hashing functions. A prediction is read from each of the banks and a majority vote is used to select a final branch direction.

The ideas of our vote predictor are derived from the skewed predictor. The skewed predictor is focused on reducing interference in 2-level branch predictor. However, our vote predictor is focused on combining multiple branch predictors to increase the prediction accuracy further as the combining branch predictor does.

3. VOTE PREDICTOR

3.1 Concept

Today hardware based branch predictor can provide prediction accuracy grater than 90%. But different branch predictors can work well for different types of branches. For example, the bimod predictor works well for those branches that are bias taken or non-taken; but the gshare works well for those branches that have correlation with nearby branches. With the reason above, this leads different branch predictors to work well for different benchmark programs. This situation is shown in Figure 1.

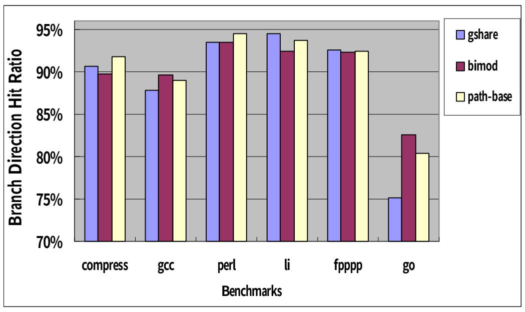


Figure 1. Branch direction hit rate of variant branch predictors with 4K entries of PHT

So if we can combine the advantages of multiple branch predictors into one predictor, we can get more accurate branch prediction. Our basic principle is to combine odd branch predictors into one predictor. This means that we can combine three or more predictors into one. But take cost-effectiveness into account we think three predictors is the realizable choice. The following examples will show how our vote predictor can increase the accuracy of prediction.

Example 1. This example is illustrated on

Figure 2 where we present a gshare, a bimod, and a path-based predictor with 16 entries of PHT. In this example, we suppose there is a destructive conflict with gshare. Under this assumption, obviously this will cause mispredictions of the first branch and the second branch in gshare. But since these two branches do not cause any conflicts in both bimod and path-based predictors, the direction still can be predicted correctly by those two predictors. So if we combine these three predictors into one predictor, then we still can get the correct direction prediction. From this example we can find that one of the advantages is that it can remove some of the conflicts in one predictor. This is because that if two branches are aliased with each other in one predictor, then they are less likely to be aliased in the others.

Example 2. This example is analogous to example 1, but there are no any conflicts with each predictor. Suppose that we are going to predict the first branch, and the gshare predictor makes the wrong prediction. There are several factors which may lead so, such as compulsory aliasing (occur when a branch substream is encountered for the first time), congenital limit (this means that the gshare predictor is no use predicting those branches). By including the other two branch predictors (suppose these two predictors can make the correct prediction), we still can correctly predict the direction.

Form above two examples, we can find that our vote predictor can dynamically select the correct prediction and this is just why our vote predictor can outperform each predictor that works alone.

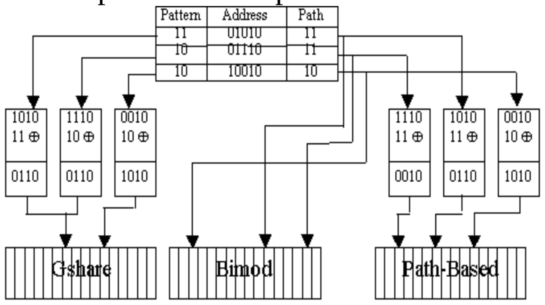


Figure 2. Conflicts can be removed by combining multiple predictions

3.2 Vote Predictor Structure

The hardware configuration of the vote predictor is shown in Figure 3. As we mention in the previous section, we combine three different predictors into our vote predictor (such as predictor 1, predictor 2 and predictor 3 in Figure 3). These three predictors can be any well-known predictors, such as gshare, bimod… etc.

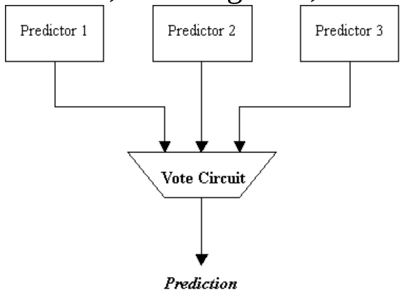


Figure 3. Logical organization of the vote predictor

The arbitrational rules are as follows:

Predict "Taken": if two or more predictors predict that the branch instruction will jump to target address, then the vote circuit will output "Taken" signal.

Predict "Non-Taken": if two or more predictors predict that the branch instruction will jump to fall-through address, then the vote circuit will output " Non-Taken" signal.

3.3 Updating the Vote Predictor

As the previous mention, there are three predictors in our vote predictor, but it is impossible that three predictors can make the correct prediction every time. With the above-motioned, we consider two policies for updating the vote predictor across multiple predictors:

Total Update policy: each of the three predictors is updated as if it were a sole predictor in a traditional prediction scheme.

Partial Update policy: when a predictor gives a wrong prediction, it is not updated when the overall prediction is correct. This wrong predictor is considered to be attached to another branch instruction. When the overall prediction is incorrect, all predictors are updated as dictated by the outcome of the branch instruction.

4. SIMULATION RESULTS

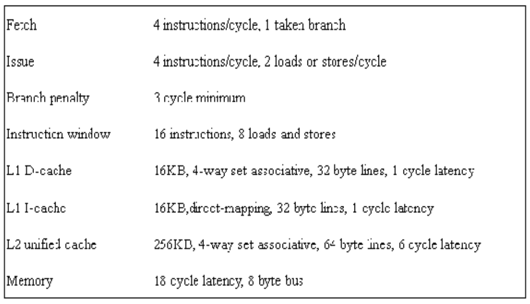
4.1 SimpleScalar Tool Set

The SimpleScalar tool set was written by Todd Austin over about one and a half year, between 1994 and 1996 [4].

4.2 Experimental Model Configurations

Details of the simulated superscalar pipeline are shown in Table 1.

Table 1. Modeled superscalar pipeline parameters



Fetch	4 instructions/cycle, 1 taken branch
Issue	4 instructions/cycle, 2 loads or stores/cycle
Branch penalty	3 cycle minimum
Instruction window	16 instructions, 8 loads and stores
L1 D-cache	16KB, 4-way set associative, 32 byte lines, 1 cycle latency
L1 I-cache	16KB,direct-mapping, 32 byte lines, 1 cycle latency
L2 unified cache	256KD, 4-way set associative, 64 byte lines, 6 cycle latency
Memory	18 cycle latency, 8 byte bus

In our experiment, we will use four well-known predictors and combine three of them arbitrarily. These four predictors are bimod predictor, gshare predictor, PAg predictor, and the path-based branch predictor. Since we use four different predictors, it will produce four different combinations. We identify the four different predictors as the following:

Vote1 Predictor: it combines a bimod, a gshare and a path-based predictor into one predictor.

Vote2 predictor: it combines a bimod, a PAg, and a path-based predictor into one predictor.

Vote3 predictor: it combines a bimod, a PAg, and a gshare into one predictor.

Vote4 predictor: it combines a PAg, a gshare, and a path-based predictor into one predictor.

In order to compare our vote predictors with each signal predictor fairly, we will configure each predictor with the same size of PHT. This means that we don't configure our vote predictor with 3 times the PHT size but with the same size of PHT.

4.3 Simulation Results

4.3.1 Update Policy

In the section 3.4, we proposed the total and the partial policies. To verify the effect of each update policy, we simulate each vote predictor with 4096-enty PHT under total and partial update policies. The comparisons between total and partial update policy are shown in Figure 4, we can find that the partial update is better than the total update. The reason why "partial update" is better than "total update" is intuitive.

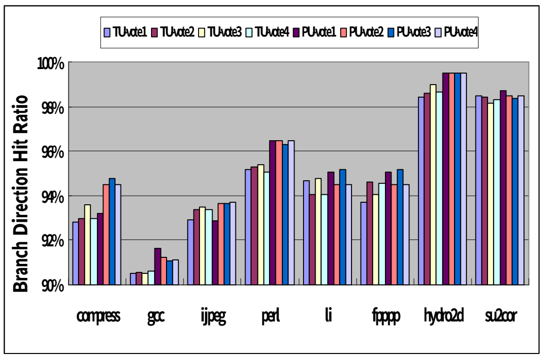


Figure 4. Branch direction hit rate of various vote predictors under total and partial update

4.3.2 Vote1 Predictor Simulation

Figure 5 is the simulation results of vote1 predictors versus its component predictors, including gshare, bimod, and path-based predictor. The PHT entries are vary from 2048 to 163824 entries.

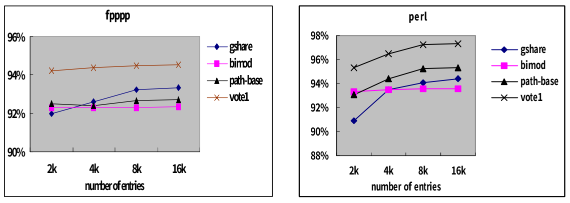


Figure 5. Branch direction hit rate of vote1 predictor vs. its component predictors

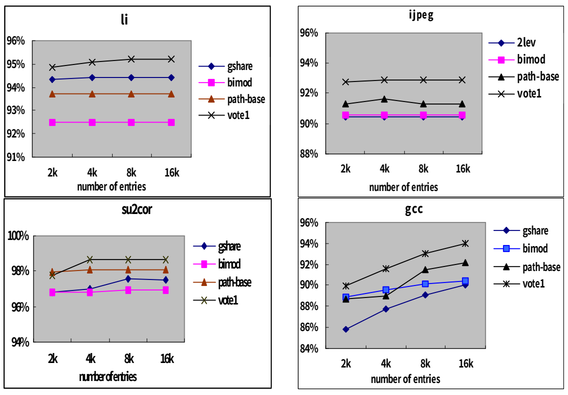


Figure 5. Branch direction hit rate of vote1 predictor vs. its component predictors (cont')

4.3.3 Vote2 Prediction Simulation

Figure 6 is the simulation results of vote2 predictors versus its component predictors, including bimod, PAg, and the path-based predictors. The PHT entries are vary from 2048 to 163824 entries.

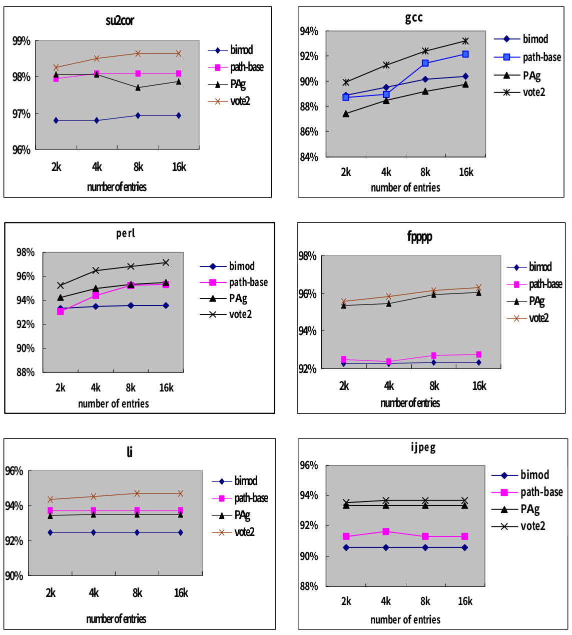


Figure 6. Branch direction hit rate of vote2 predictor vs. its component predictors

4.3.4 Vote3 Predictor Simulation

Figure 7 is the simulation results of vote3 predictor versus its component predictors, including bimod, PAg, and gshare predictors, the pattern history table (PHT) entries are vary from 2048 to 163824 entries.

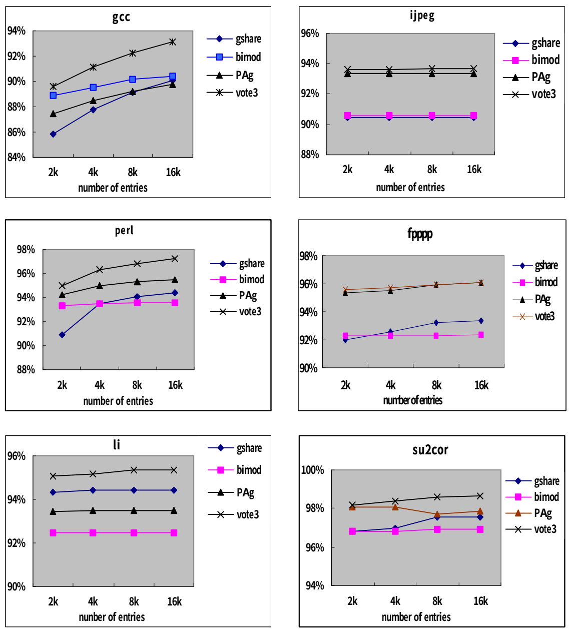


Figure 7. Branch direction hit rate of vote3 predictor vs. its component predictors

4.3.5 Vote4 Predictor Simulation

Figure 8 is the simulation results of vote4 predictor versus its component predictors, including gshare, PAg, and the path-based predictors, the pattern history table (PHT) entries are vary from 2048 to 163824 entries.

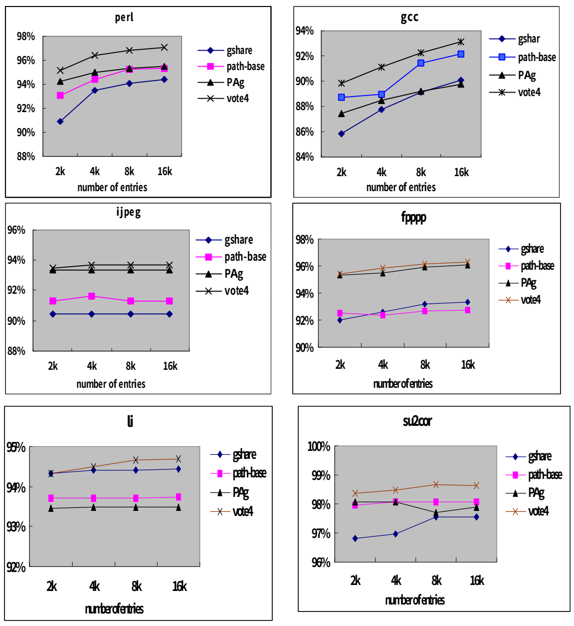


Figure 8. Branch direction hit rate of vote4 predictor vs. its component predictors

4.3.6 Compare Various Vote Predictor Models

According to the previous results, each predictor with 4k-entry PHT has the best cost-effectiveness. So in the following, we will compare various vote predictors under 4k-entry PHT. The results of comparison between our four vote predictors are shown in Figure 9. Table 2 lists the hardware costs of various vote predictor models.

From the results, we can find that the vote3 predictor has the best average prediction accuracy, and the vote1 predictor has the worst average prediction accuracy. However, if we take the hardware cost into account, the hardware cost of vote1 predictor is approximate to 1/3 those in other predictors. So the vote1 predictor model still has the best performance/cost ratio.

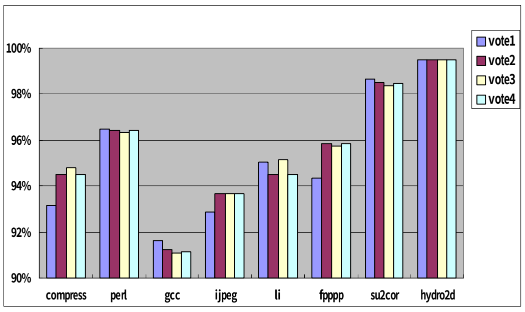


Figure 9. Comparison of various vote predictors models with 4K-entry PHT

Table 2. Hardware costs of various vote predictor models

	gshare	bimod	path-base	PAg	PHT
vote1	8 bits	0 bits	9 bits	N/A	4096*2 bits
vote2	N/A	0 bits	9 bits	2048 * 8 bits	4096*2 bits
vote3	8 bits	0 bits	N/A	2048 * 8 bits	4096*2 bits
vote4	8 bits	N/A	9 bits	2048 * 8 bits	4096*2 bits

4.4 Vote Predictor VS. Combining Branch Predictor

4.4 Vote Predictor VS. Combining Branch Predictor

The combining branch predictor was proposed by McFarling. It combines multiple branch predictors into one, including bimod and gshare predictors, and then uses a meta-table to choose which result to be used. In order to compare our vote1 predictor with combining predictor fairly, we configure the combining branch predictor with one bimod, one gshare that has 8-bit global history register, and one meta-table, and the same size of PHT as the vote1 predictor except the size of meta-table. For the meta-table, we configure it with extra 2048-entry PHT. The results are shown in Figure 10.

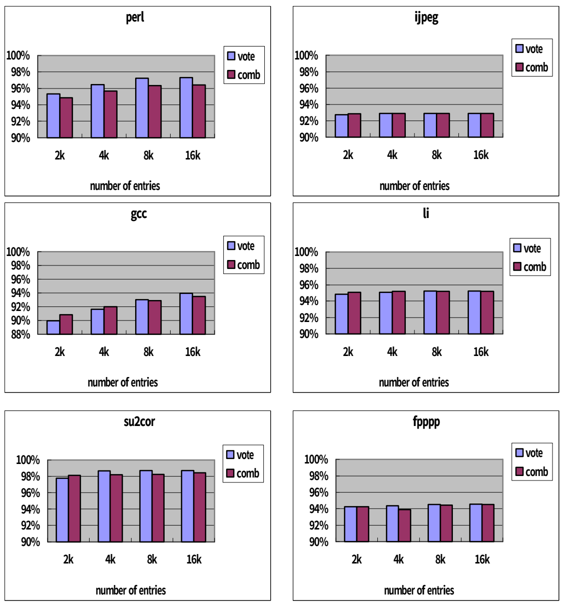


Figure 10: Branch direction hit rate of vote predictor vs. combining branch predictor

4.5 Performance with Context-Switch

In the real world, the context-switches will occur frequently. When a context-switch occurs, the relative machines that use to keep track of branch history need to be flushed. The effects of prediction accuracy with context-switch and without context-switch are shown in Figure 11. In this experiment, we compare the vote1 predictor with its component predictors with 4096-entry PHT and assume a context-switch occurs every 100k and 500k instructions. Here the “gshare-100k” represents the gshare predictor with context-switch occurring per 100k instructions and the “gshare-no cs” represents no context-switch occurring. The value 100k is derived by assuming that a 100 MHz clock is used and a context-switch occurs every 1 ms in a 1 IPC machine. However, the value 500k is derived by assuming that a 500 MHz clock is used and a context-switch occurs every 1 ms in a 1 IPC machine.

According to the results, we can find that the bimod predictor has the lowest average accuracy degradation, the vote predictor is second, the path-based predictor is third, and the gshare is fourth. This is not a bit surprised, since the bimod predictor only uses the simple 2-bit table to keep track of branch history; it needs less warm-up time. However, for our vote predictor, because of the existence of complementary relations, it can reduce the warm-up time. The fact that path-based predictor is better than the gshare predictor is consistent with that in [3].

Although for accuracy degradation, our vote predictor is not the lowest, but for the prediction accuracy, our vote predictor is still better than other predictors.

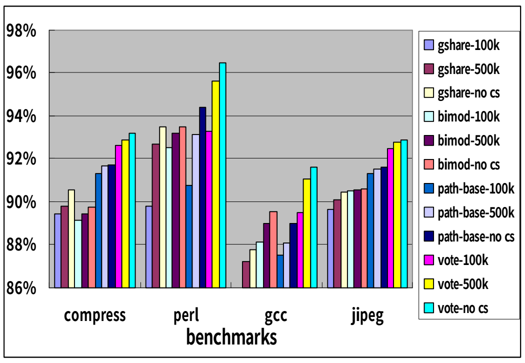


Figure 11. Prediction accuracy with context-switch

5. CONCLUSIONS

In this paper, we experiment on six benchmark programs with several well-known branch predictors to compare their prediction accuracy. With 4K entries of pattern history table, the vote predictor increases the prediction accuracy with one of its most accurate component predictor from 1% to 2%. In this paper, we only use four simple predictors to construct the vote predictor and the average prediction accuracy is 95.5%. In the future maybe we can put other complex and accurate predictors into the vote predictor to obtain more prediction accuracy.

6. REFERENCES

[1] S. McFarling, “Combining Branch Predictors,” Technical Report TN-36, Digital Western Research Laboratory, June 1993.

[2] T. -Y. Yeh and Y. N. Patt, ”A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History,” Proceedings of the 20th Annual International Symposium on Computer Architecture, Pages 257-266, 1993.

[3] R. Nair, “Dynamic Path-Based Branch Correlation,” Proceedings of the 28th Annual ACM/IEEE International Symposium on Microarchitecture, pages 15-23, 1995.

[4] D. Burger and T. M. Austin, “The SimpleScalar Tool Set Version 2.0,” Technical Report 1342, Computer Sciences Department, University of Wisconsin, Madison, WI, 1997.

[5] Pierre Michaud, Andre Seznec, Richard Uhlig, “Trading Conflict and Capacity Aliasing in Conditional Branch Predictors,” 24th Intl. Symp. On Computer Architecture, pp. 292-303, June 1997.

[6] J. K. F. Lee and A. Smith, ”Branch Prediction Strategies and Branch Target Design,” Computer, vol. 17,no. 1, pp. 6-22, Jan. 1984.