Department of Electrical and Computer Engineering

Computer Architecture

Instructor: Naghmeh Karimi

Evaluating Branch Prediction Strategies

Junjie Feng, Ruotian Zhang, Lulu Jiang

December 15, 2015

1. Introduction

It is well known that branch instructions can break the smooth flow of instruction fetching and execution in a highly parallel computer system. A delay will happen because the branch that is taken changes the location of instruction fetches and the issuing of instructions must often wait until conditional branch decisions are made. And the deeper the pipelining is, the greater performance loss is. In order to reduce the performance loss result from the branch instructions, a predictor can be designed to predict the direction that the branch instruction will be taken. However, when a wrong prediction is made by the predictor, it means that more delay will happen. In this way, the designers of the predictor have to work on improving the accuracy of the predictor making a right prediction to make the system a best performance.

In our report, first of all we discuss the default branch prediction strategy in SimpleScalar which is an open source computer architecture simulator. Then we introduce our own way to improve the accuracy of branch prediction.

We conducted a simulation to test the performance of every strategy mentioned in this study. The simulation is based on SimpleScalar simulator platform. In order to get the ideal test statics, we choose these following six benchmarks in SPEC 95 benchmark suites: 1) apsi 2) ijpeg 3) perl 4) tomcatv 5) cc1 6) vortex. Table 1 gives the description of these benchmarks.

Benchmark	Description
apsi	Temperature, wind velocity and distribution of pollutant
ijpeg	Graphic compression and decompression
perl	Manipulates strings and prime numbers in Perl
tomcatv	A mesh generation program
cc1	An old perl interpreter
vortex	A database program

Table 1 Description of the benchmarks

2. Implementation

2.1. Default Strategies in SimpleScalar

SimpleScalar branch prediction simulator provides a number of branch prediction mechanisms. For static prediction, it has taken and not taken strategies. Taken strategy, as the name implies, predicts all the branches will be taken. Comparatively, not taken strategy predicts all the branches will be not taken. For dynamic prediction, there is a predict strategy called bimodal. The bimodal strategy uses a Branch History Table which has 2048 entries and each entry contains a 2-bit counter. It use the low-order 11-bit of the Branch Address to index the entry. The architecture of bimodal predictor is shown in Fig.1.

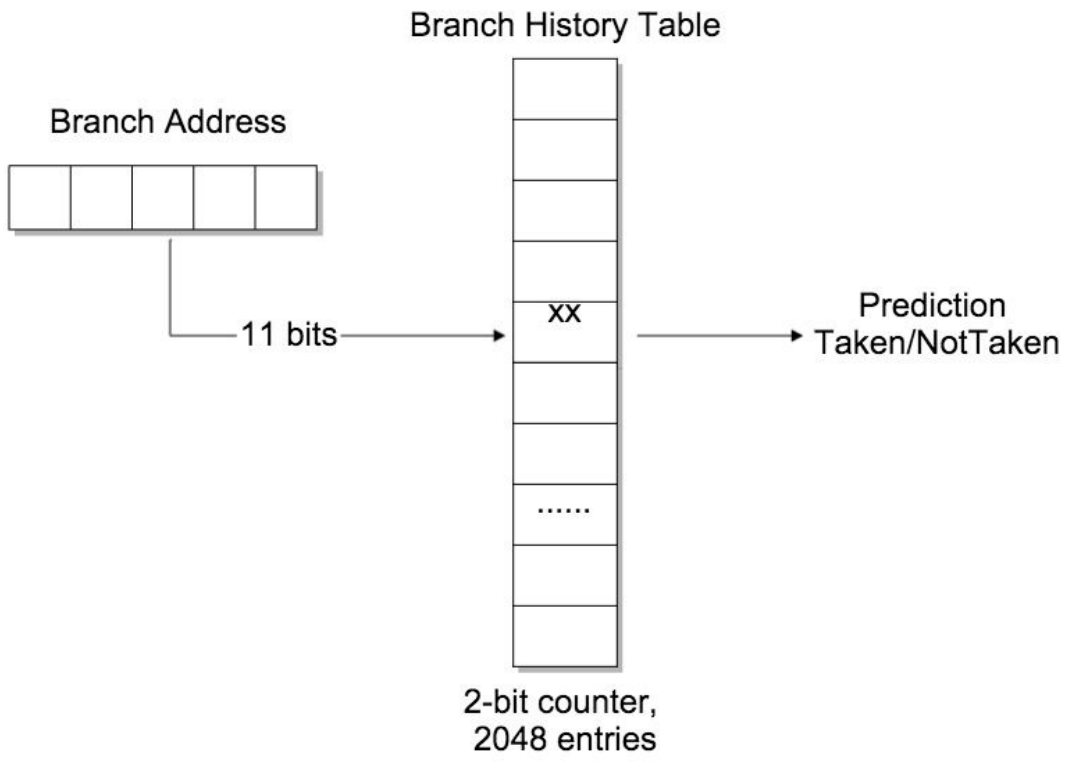


Fig.1 Bimodal predictor architecture

For this scheme, the 2-bit counter is a significant component which contains the recent branch outcome information. The value in the counter is from 00 to 11. 00 is strongly not-taken state, 01 is weakly not-taken state. 10 is weakly taken state and 11 is strongly taken state. The counter is incremented on a taken branch and decremented on a not taken branch. The initial state is randomly chosen between 01 and 10, because if the first prediction is wrong, the counter will change state to a different prediction. This kind of initialization makes the prediction more flexible. Fig.2 shows the finite-state machine of bimodal strategy.

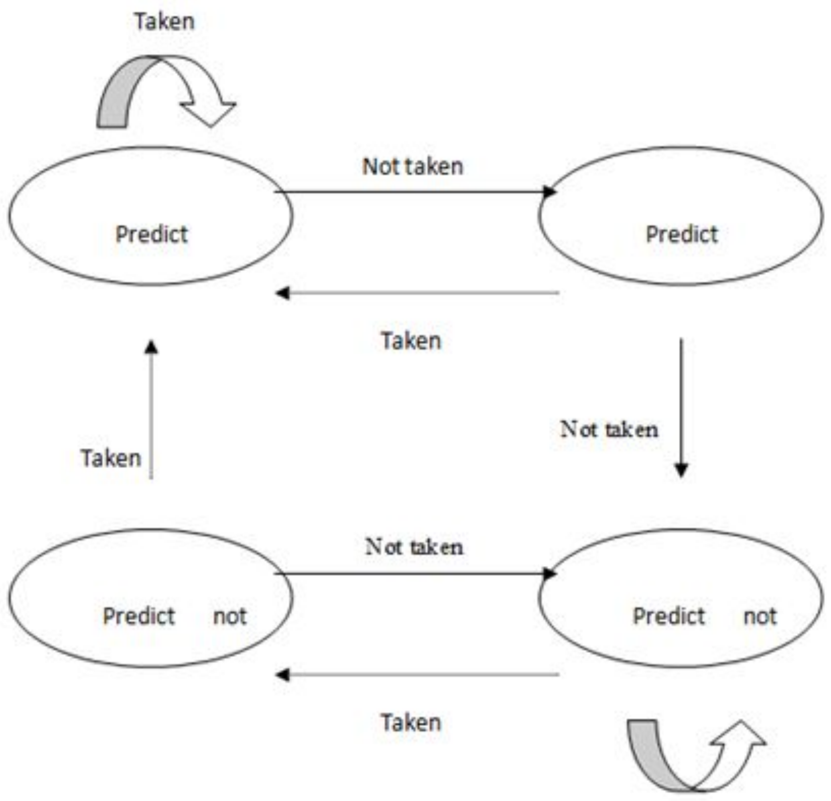
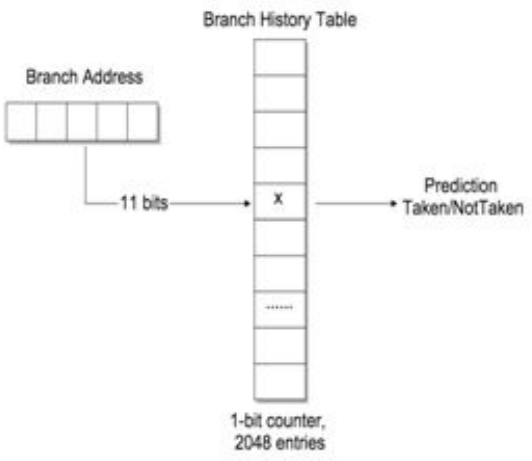


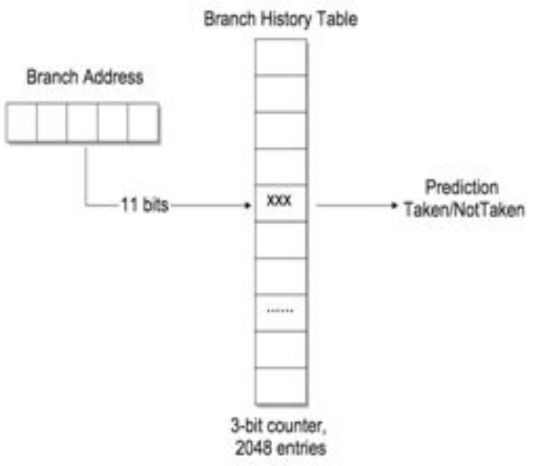
Fig.2 Finite-state machine of bimodal strategy

2.2. Strategy 6 and strategy 7 form the paper

The paper “A Study of Branch Prediction Strategies” published by JAMES E. SMITH introduces several branch prediction strategies. The most impressive two of them are strategy 6 and strategy 7. These two branch prediction strategies are very similar to the bimodal strategy mentioned above. Strategy 6 and 7 also have a Branch History Table which has 2048 entries and they also use the low-order 11-bit of Branch Address to index the entries. The only difference between strategy 6&7 and bimodal strategy is that the bit number of counter in each entry is not the same. Strategy 6 uses a 1-bit counter in each entry and strategy 7 uses a 3-bit counter in each entry. Fig.3 gives the architecture of strategy 6 and strategy 7.



Strategy 6



Strategy 7

Fig.3 Architecture of strategy 6 and strategy 7

For strategy 6, the 1-bit counter records the most recent branch outcome. 0 represents the predictor predicts the branch will be not taken and 1 represents will be taken. If the predictor is confirmed made a wrong prediction, the number in the counter will be changed.

For strategy 7, the counter is changed to 3-bit which the first bit is set as a sign bit. If the sign bit is 0, the prediction is the branch will be not taken and if it is 1, the prediction is taken. The value of counter is between 000 and 111. The counter is incremented on a taken branch, just like bimodal strategy, and decremented on a not taken branch. The initial state is randomly chosen between 011 and 100.

2.3. New Branch Prediction Strategies by two-level adaptive predictor

In our new prediction strategies, we still focus on the dynamic branch prediction, but the difference is that we utilize 2-level branch prediction and execute all benchmarks by this new strategy on SimpleScalar simulation platform.

2.3.1. The reason of using 2-level prediction strategy

In this part, we will explain why we decide to use 2-level predictor and how much in degree it can improve the prediction accuracy. Let we use two simple examples in C programming language to show you the importance of using 2-level predictor.

```
1. if (aa == 5) aa = 0;     if (bb == 5) bb = 0;     if (aa != bb) { /*code*/ }
2. if (x[i] < 10) then y += 2;     if (x[i] < 5) then c -= 4;
```

Above are two programs which both contain branch in them. The first program has three branches in it and we can tell that if the first and the second branch are both taken then the third branch will be not taken. In a word, behavior of branch 3 is correlated with the first and second branch. The second program has two branches in it and one interesting thing is that if the first not taken by the condition, then the second branch must be not taken too because of some correlations among the first and the second branch.

So it's easy to see the fact that the outcome of the branch depends not only on the branch address but also on the outcome of other recent branches. So if there is some strategy which could record the history of the recent branches' outcomes and make predictions based on previous results in a new predictor, the accuracy of branch prediction will be better normally. It becomes obvious that behavior of longer sequence of branch execution history often provides more accurate prediction outcome.

Therefore, we let global history bits record the preceding n branches' outcomes. Each time the predictor need to make a prediction, the hashed branch address and the global history bits would work together to decide to use which 2-bits counter in Branch history table by now predictor.

In next two sections, we would introduce two variations of 2-level predictor which have been designed and executed in our project and show you the results of these two variations predictor compared to the strategy 6, strategy 7 and the bimodal predictor.

2.3.2. Correlating branch prediction

Firstly, we set the parameters in our correlating branch prediction. Below are three important features that we considerate during the process of build this predictor.

- Using global history shift register which contains last 2 branch outcomes.
- Low-order 11 bits from branch address are used to choose the pattern history table.

- The global history bits are used to index the entry of the pattern history table. Each entry contains 2-bit counters.

And here is the diagram of correlating branch predictor in this figure.

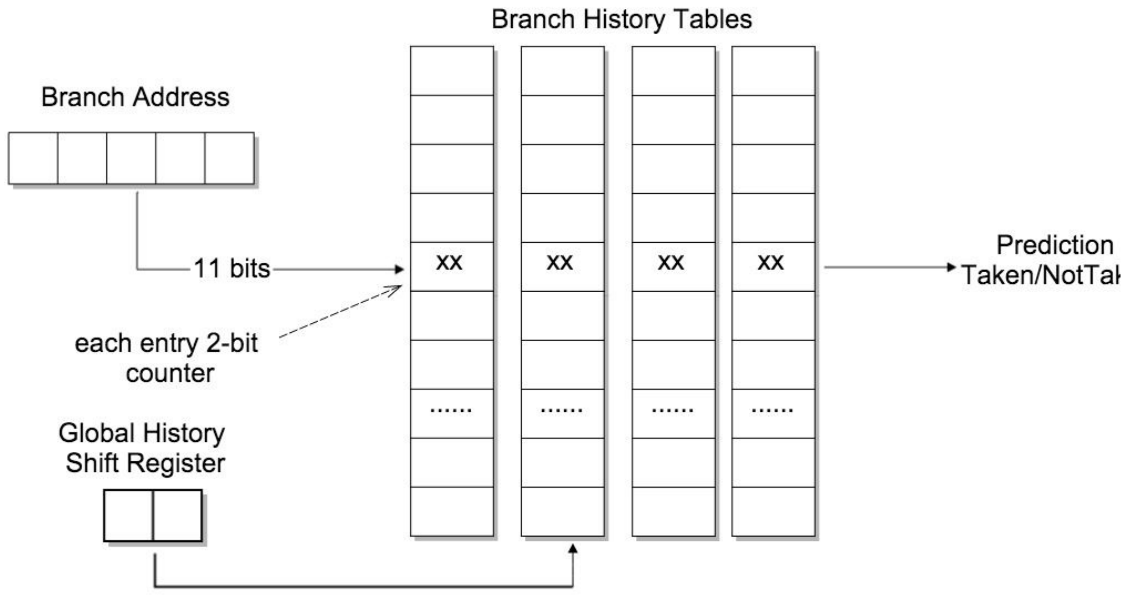


Fig.4 Architecture of correlating branch predictor

In the figure 4, we draw the branch history table in two-dimension way for ease to understand. There are 2048 entries in each branch history table and each entry is indexed by the lower 11 bits which hashed from the whole branch address, and the global history shift register contains two bits so that it could link to 4 entries which include 2-bits counter in each self. By this mean, different global bits may lead to different status in branch history table due to correlation is considerate in our new predictor.

The results of frequency of mis-prediction between correlating branch prediction, strategy 6, strategy 7 and bimodal strategy will be presented in the next results section.

2.3.3. Correlating branch prediction by Gshare (Global history with index sharing) Predictor

Also, there is one important issue that affects branch prediction and it is aliasing. Aliasing is the overlapping of the different branch address to the same branch history table entry. And the causes of this problem can be concluded into two reasons: 1. Due to a limited table (Branch history table) size, such that there are more branch instruction addressed than table entries. 2. Due to their nature, aliasing occurs only with direct mapping and hushing.

So in this situation, we replace one part in the correlating branch predictor and change it into a Gshare predictor. We use the result of low- order 2 bits of branch address to Exclusive OR with the 2 bits in the global history register and then make the result of them be the index to address the entry. By this improvement, this strategy could effectively deal with aliasing issue. Here is the diagram of Gshare branch predictor in this figure.

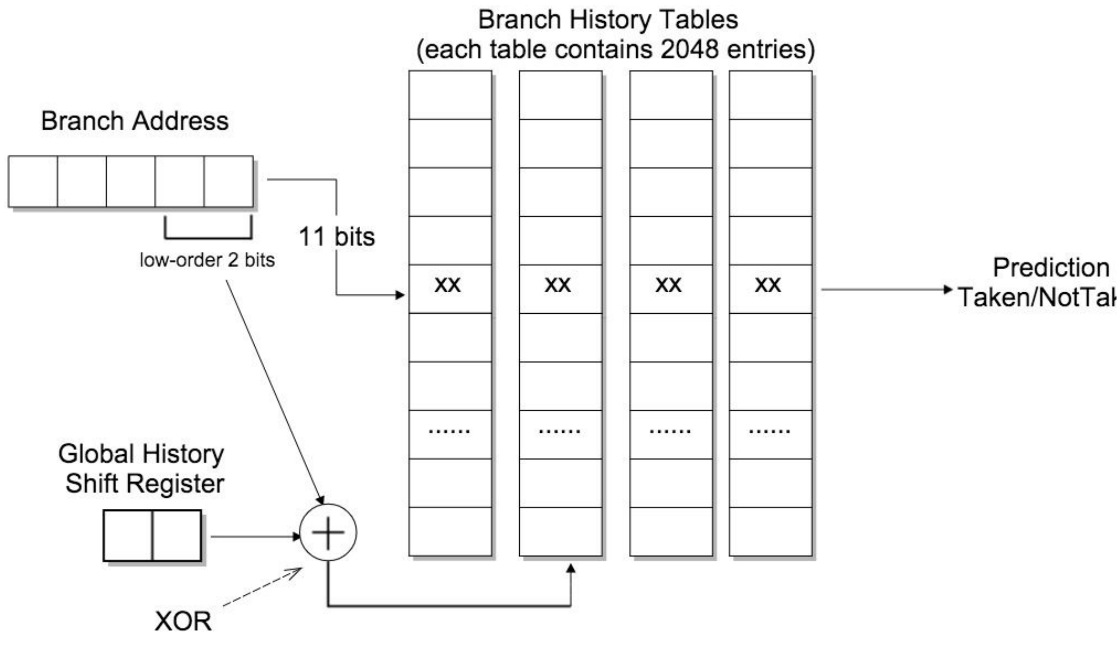


Fig.5 Architecture of Gshare branch predictor

The results of frequency of mis-prediction between correlating branch prediction and Gshare strategy will be presented in the next results section.

2.4. New Branch Prediction: Combining Branch predictor

In the previous sections, we explain that the bimodal predictor and 2-level adaptive predictor have different advantages. For instance, if the outcome of current branch depends strongly on other recent previous branches, using the 2-level predictor will get the better result. For the branches which are usually taken or not taken, using the bimodal predictor will get better results.

However, it is probable that one benchmark could contain all kinds of branches behaviors. If we use one of these predictors for a given benchmark, many branches will be predicted inaccurately. In order to get more accuracy of branch prediction, we use a new strategy which is called combining branch predictor.

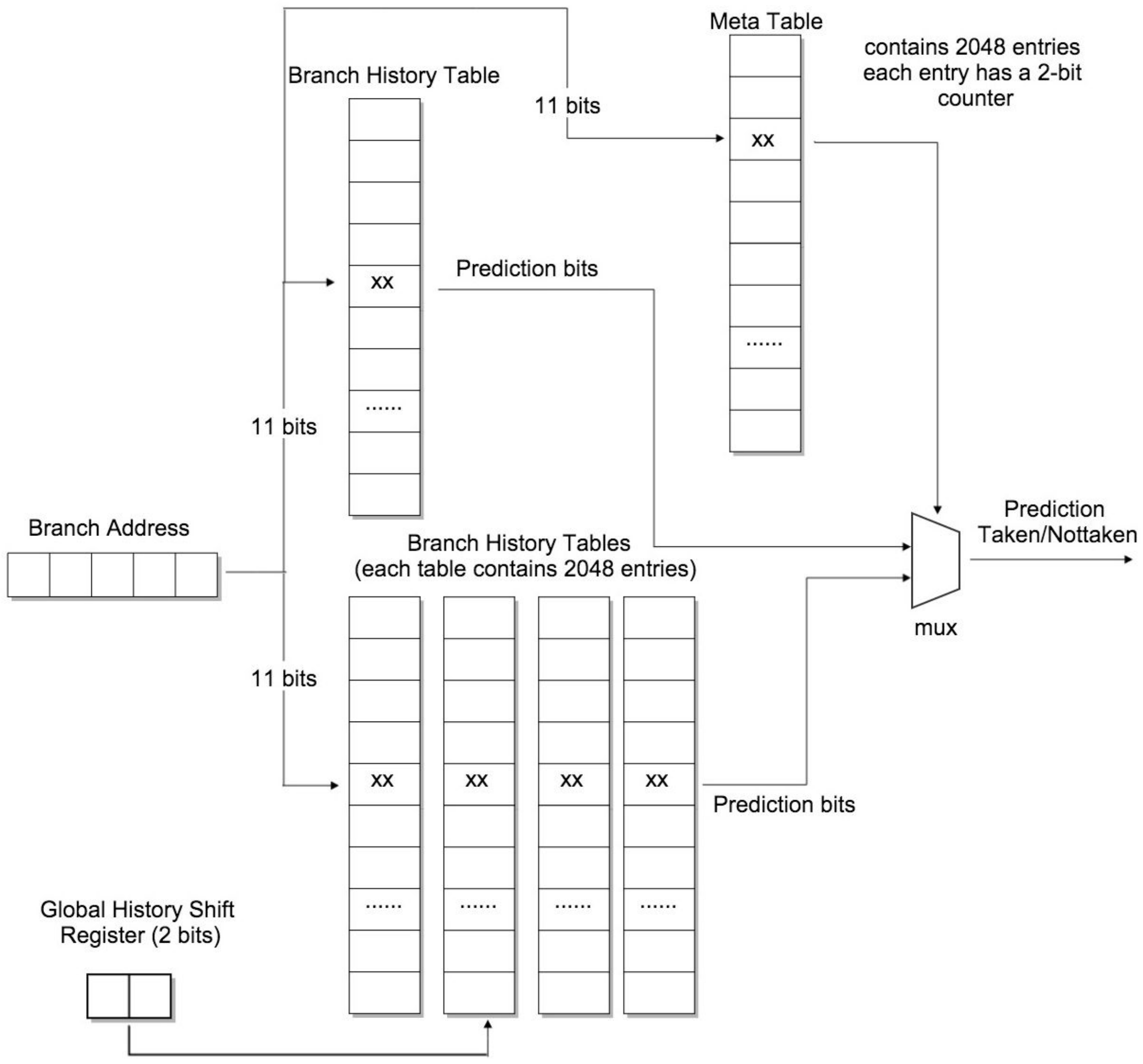

Meta Table
contains 2048 entries each entry has a 2-bit counter
Branch History Table
11 bits
xx
Prediction bits
......
11 bits
Branch Address
Branch History Tables (each table contains 2048 entries)
11 bits
Prediction
Taken/Nottaken
mux
Prediction bits
Global History Shift Register (2 bits)

Fig.6 Architecture of combining branch predictor

Combining branch predictor, as its name suggests, combines two branch predictors together, one is bimodal branch predictor, and another is correlating branch predictor. The figure 6 shows the hardware structure of this branch predictor. On the left side of the graph, there are two predictors which we introduced in the previous sections. On the right side of this graph, the additional hardware is the meta-table and the multiplexer. The meta-table contains the same number of entries as bimodal predictor does, and each entry contains a 2-bit counter. It works like a selector.

In combining branch predictor, the counter in meta-table will record the history for each branch. If the counter is 00 or 01 which means that in the history, the bimodal predictor gets the more accuracy of prediction on the given branch address. Otherwise, if the counter is 10 or 11, that means the correlating predictor gets the more accuracy of prediction. So, how the counter in meat table changes become the vital part in this predictor.

The table 2 shows the update strategy of meta-table, when the predictions of two predictors are same, no matter their prediction is correct or incorrect; the counter in meta-table will not change. If only one predictor make the correct prediction, the counter in meta-table will decrement or increment by 1. Meanwhile, if the two component predictor makes the wrong prediction, the system will update their Branch History Tables as normal way which we introduced in previous sections.

Bimodal	Correlating	Counter in meta table
Correct	Correct	No change
Incorrect	Incorrect	No change
Correct	Incorrect	Increment
Incorrect	Correct	Decrement

Table 2 Update strategy

In the next section, we will show the frequency of mis-prediction of combining predictor and compare the result of different predictors.

3. Results

In order to compare the performance of different strategies, we use the fixed number of address bits (11 bits), which will limits the aliasing influence to the same level in different strategies. According to the number of branch address bits, the number of entries in branch history table is 2^11, or 2K.

For the 2-level adaptive predictors, we use 2 bits of global history which means only consider the last two branches outcomes. As a result, the number of entries in the second level is 2^2*2^11, or 8192, or 8K.

3.1. Default Branch Predictors

The prediction accuracy of the three default strategies in SimpleScalar, taken, not taken and bimodal, is given in figure 7. From the result graph, the taken strategy always performs better than not taken strategy and bimodal strategy performs much better than the other two. This results from the fact that the branches in the benchmarks are taken in most cases.

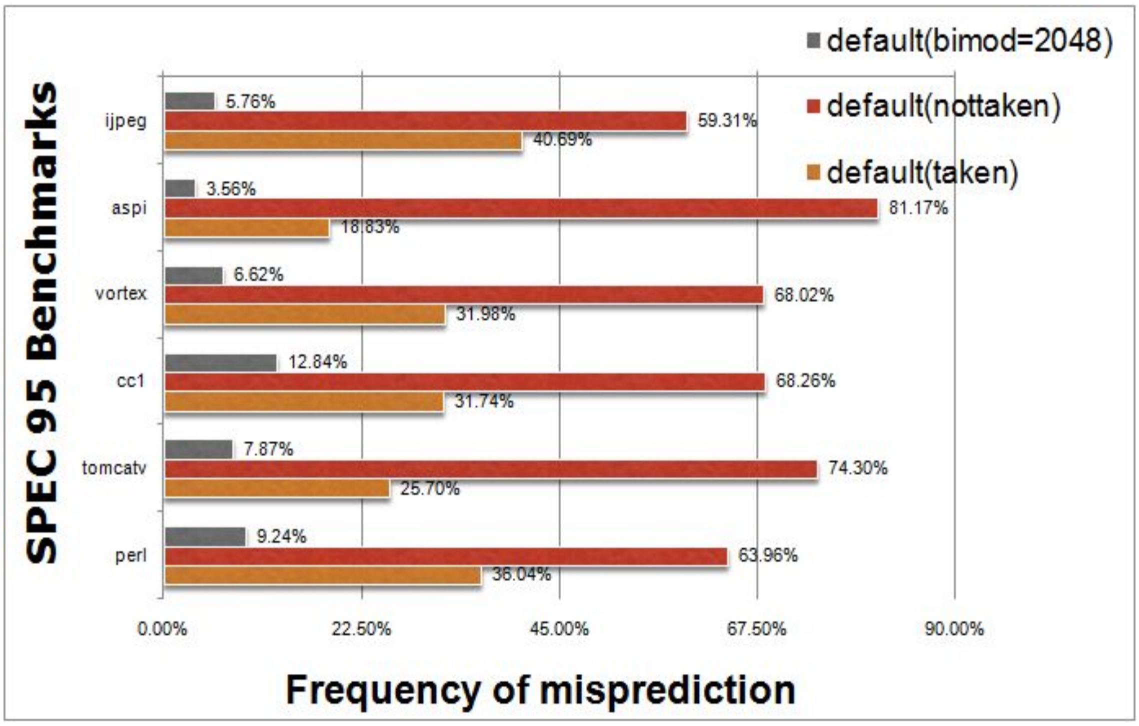


Fig.7 Frequency of misprediction of the three default strategies in SimpleScalar

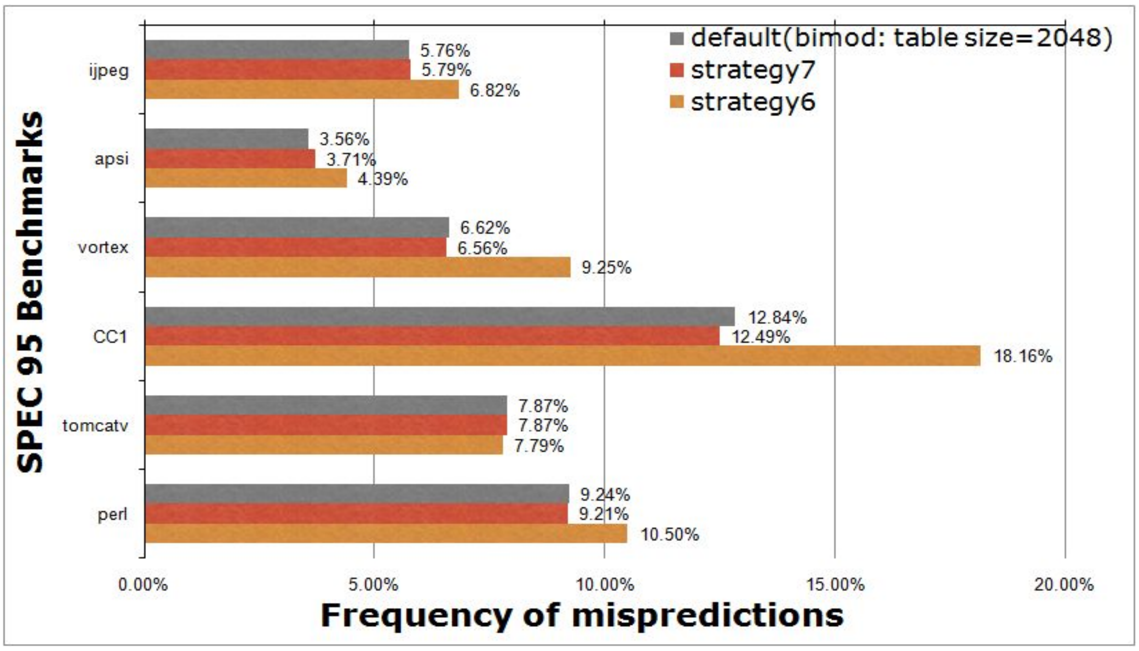


Fig.8 Frequency of misprediction of bimodal predictor, strategy 6 and strategy7

The figure 8 shows the result of prediction accuracy between bimodal strategy, strategy 6 and strategy 7. The reason why bimodal strategy can always gain more accuracy can be explained by the following:

```
for(i=0;i<1000;i++)
    {     for(j=0;j<100;j++)
      {
          /*code*/
      }
    }
```

For instance, it is possible that the benchmark which we tested has a lot of loops like this. Assuming that the initial prediction is not taken, bimodal strategy which has a 2-bit counter will only miss one time. However, strategy 6 will miss every time when entry the inner loop. In this way bimodal strategy will have 999 less misses. As for strategy 7and bimodal strategy, bimodal strategy sometimes performs better due to the "inertia" which can be built up with a larger counter in which history in the too-distant past is used.

3.2. Two Level Adaptive Branch Predictors

The results between correlating branch prediction, strategy 6, strategy 7 and bimodal predictor:

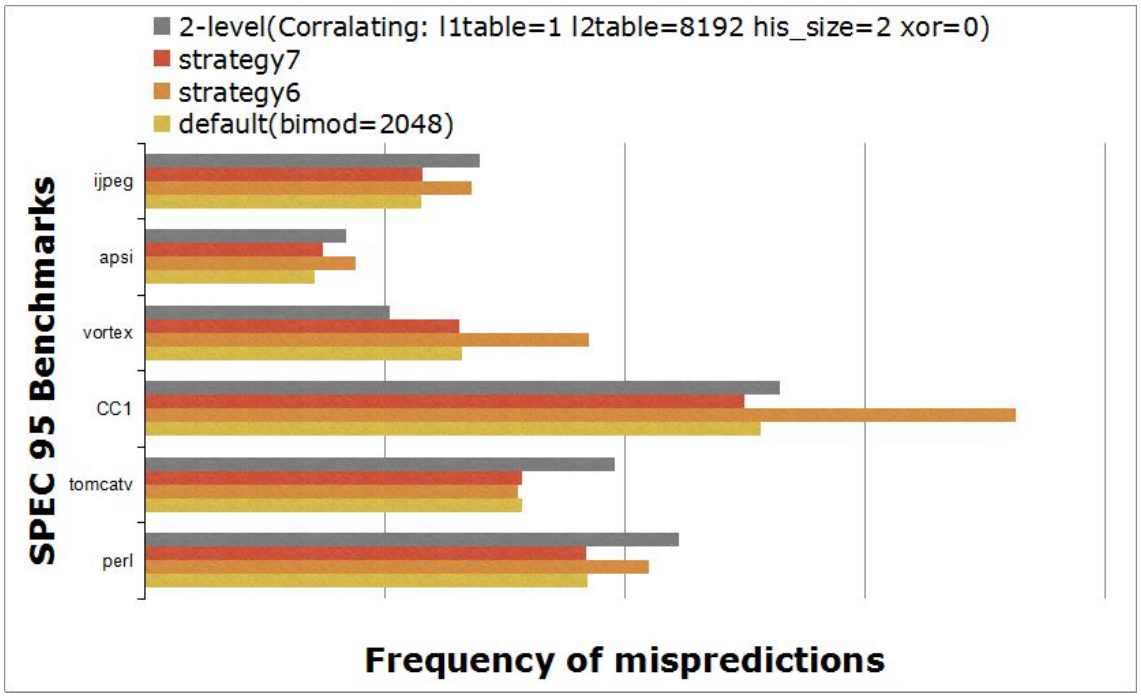


Fig.9 Frequency of misprediction of Correlating predictor and previous predictors

In the figure 9, we can see that in these two benchmarks vortex and cc1, correlating branch predictor predicts much better than the others. In benchmark apsi, correlating branch predictor is only better than strategy 6, and it is because that maybe the recent branch outcomes are less correlating with the branch that in prediction. Meanwhile, in benchmarks tomcatv, ijepg and perl, correlating branch predictor predicts not well as others. So it proves that correlations between branches in this program may be negative to the prediction.

The results between correlating branch predictor and Gshare prediction:

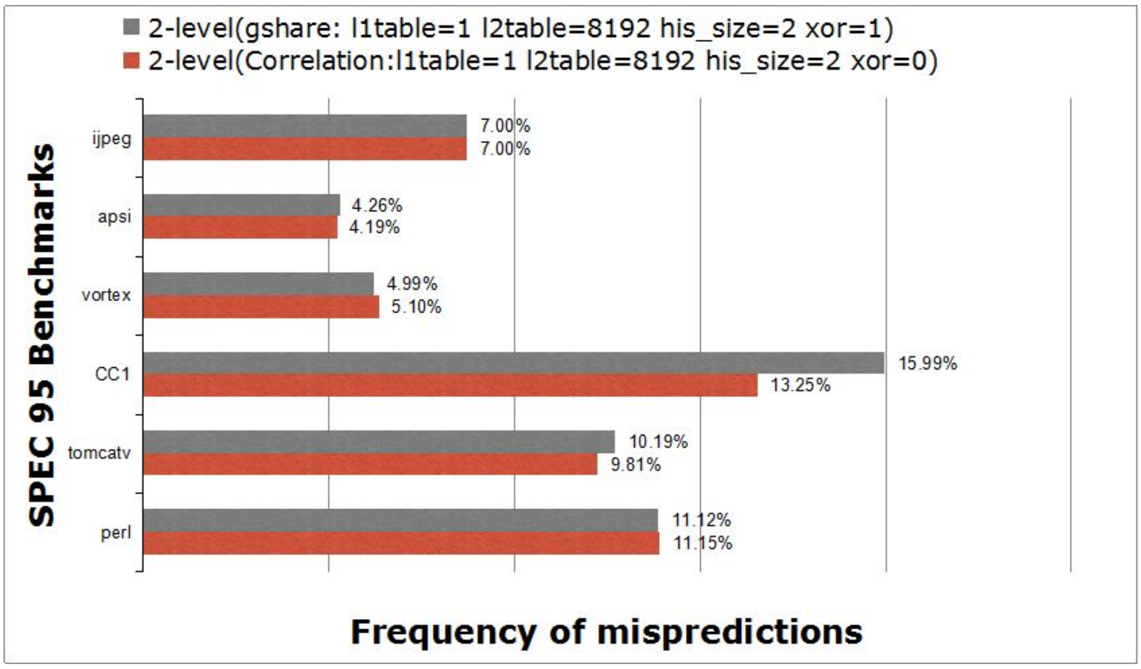


Fig.10 Frequency of misprediction of correlating and Gshare branch prdictor

In the figure 10 , for the benchmark vortex and perl, Gshare predictor is better than correlating predictor. In the benchmark ijpeg, tomcatv and apsi, Gshare performs as well as or a bit worse than correlating predictor. In the benchmark cc1, Gshare predictor is worse than correlating predictor.

In general, aliasing does not necessarily mean that the number of correct branch predictions is reduced. It is even possible that aliasing improves the current prediction rate. In this case, this is called constructive aliasing. So maybe there are constructive aliasing in the benchmark cc1 program and the performance will be influenced by eliminating the aliasing out. If aliasing actually reduces the correct prediction rate, it speaks of destructive aliasing. Therefore, there are destructive aliasing in the benchmark vortex and perl. The accuracy of prediction got improved by eliminating aliasing out. And in the end, if aliasing does not affect branch prediction at all, it is refer to as harmless aliasing which occurs in the benchmark ijpeg and apsi.

3.3. Combining Branch Predictor

The results of all the branch predictors:

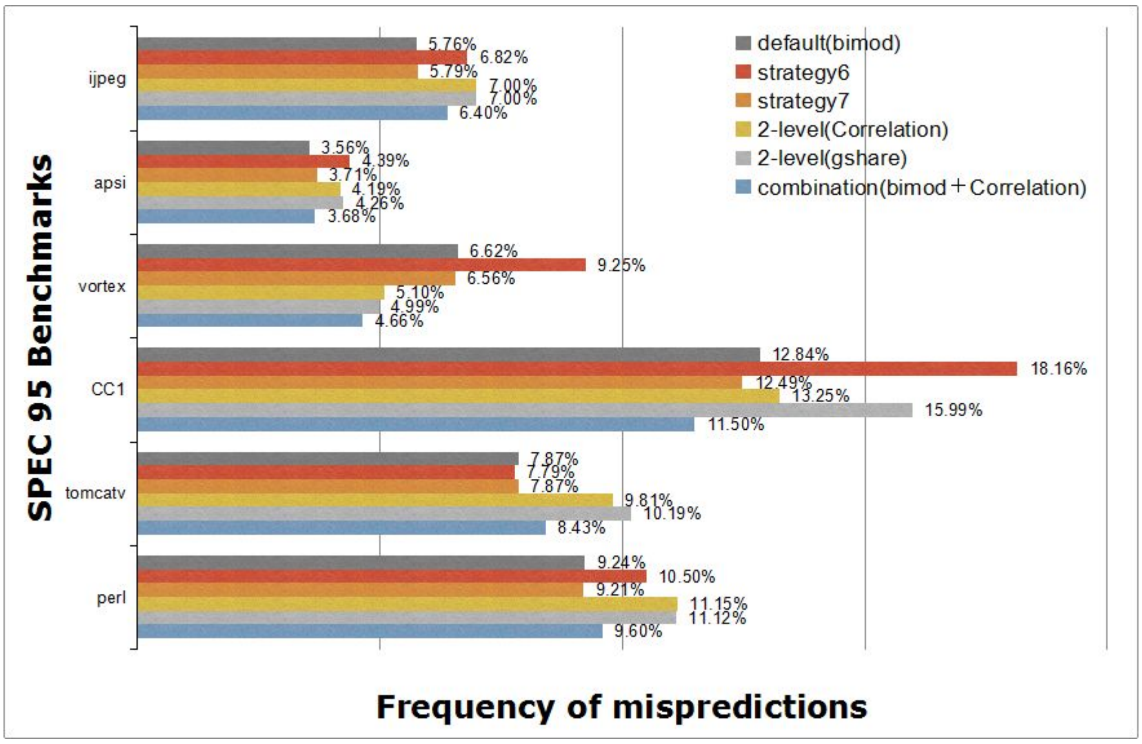


Fig.11 Frequency of misprediction of all the branch predictors

In the figure 11, for the benchmark vortex and cc1, the combining predictor is the best predictor among all the predictors mentioned in the report. This result is easy to understand, because in combining strategy, the system could select the better predictor for each branch. Many branches may use bimodal predictor and the others use correlating predictor. As a result, the total number of mis-prediction will decrease.

For benchmark apsi, the combining predictor and the bimodal predictor have almost same accuracy of prediction. For the benchmark ijpeg and tomcatv and perl, the bimodal predictor has the best performance. These results seem to be unacceptable. However, there are some reasonable issues which affect the performance of combining predictor.

First issue is about the initial value of all the branch history tables and meta table. The initial value of all the counters is randomly chosen between 10 and 01. For some special combination of initialization, the counters need long training time to be updated. In the training time, there is high possibility of the serial miss predictions. Especially for the benchmark which contains small number of branches, the mis-prediction occurs in the training period has a great effect on the overall accuracy.

Second reason for this fact is that, in the benchmark apsi, maybe most branches are using the prediction of the bimodal predictor. As a result, the correlating predictor has little influence on the overall accuracy. So the combining predictor has the similar performance to the bimodal predictor.

4. Conclusion

In our branch prediction project, we have displayed two new predictors for better branch prediction accuracy. First, we came out with 2-level predictor used correlating prediction. And then considering aliasing issue existed, we showed that using 2 bits exclusive OR of the global branch history and the branch address to access predictor counters results in branch history table for a given counter array size. Also there are maybe constructive aliasing among different programs, but we proved the performance could be improved by eliminating destructive aliasing. Our next predictor is called combining branch predictor. In this part, we showed that the advantages of multiple branch predictors can be combined by providing multiple predictors and by keeping track of which predictor is more accurate for the current branch. These methods permit construction of predictors that are more accurate for a given size than previously mentioned strategies. Also, combined predictors using its meta table information which could reach a prediction accuracy of 96.32% at best as compared to 95.74% for the previously our 2-level prediction scheme. The strategies presented here should be increasingly helpful as computer architecture designers attempt to take advantage of instruction level parallelism and reduction in frequency of miss-predicted branches.

5. References

1. James E. Smith. A Study of Branch Prediction Strategies
2. Tse-Yu Yeh and Yale N. Patt. Two-Level Adaptive Training Branch Prediction
3. Scott McFarling. Combining Branch Predictors.
4. Tse-Yu Yeh and Yale N. Patt. A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History
5. ZiJun Feng, Junhua Xiao, and Longbing Zhang. History and Present processor branch prediction research