2005

A Study for Branch Predictors to Alleviate the Aliasing Problem

Tieling Xie

Robert Evans

Yul Chu
The University of Texas Rio Grande Valley

Recommended Citation

Tieling Xie, R. Evans and Y. Chu, "A study for branch predictors to alleviate the aliasing problem [pipelining]," Proceedings. IEEE SoutheastCon, 2005., Ft. Lauderdale, FL, USA, 2005, pp. 603-608, doi: 10.1109/SECON.2005.1423313.

A Study for Branch Predictors to Alleviate the Aliasing Problem

Tieling Xie, Robert Evans, and Yul Chu
Electrical and Computer Engineering Department
Mississippi State University
chu@ece.msstate.edu

Abstract

Modern processors usually have a deep pipeline, superscalar architecture to obtain higher performance. As pipelines are getting deeper, accurate branch prediction is critical to achieve high performance since fetched instructions after a branch have to be flushed inside pipeline when prediction is wrong. This paper studies the performance of several types of branch predictors starting from local branch predictor and global branch predictor. Simulation results show that global history predictor outperforms local history predictor due to the characteristic that branches tend to be correlated. However, global history predictor still suffers aliasing problem that degrades performance. Four techniques are proposed to alleviate aliasing problem. The performance is evaluated by using Simplescalar with SPEC CINT95 benchmark programs. The proposed predictors display better performance over the conventional predictors after careful configuration for each.

1. Introduction

Pipelining is common to modern microprocessor architectures. It essentially overlaps the execution of instructions to improve performance, which is ILP (Instruction-Level Parallelism). The performance gained is largely affected by dependencies of instructions. Control dependency is one of them. In general, there is approximately one branch instruction out of five or six instructions in most of programs [1]. Hence, accurate branch prediction is significant to improve performance. It is even more critical for the deep pipeline processors since more instructions have to be flushed as the wrong instructions have been fetched.

Studies have shown that branch behavior is not random. They are either biased for taken or not taken, or correlate with each other [2]. "For" loop is an example of biased taken branch except for the beginning and ending of the loop. A sequence of ifs (ex: if -if -else) instructions can be the examples of highly correlated branches [1].

Several branch prediction schemes have been proposed [2, 3, 4, 5]. The primitive ones are static in which they predict always taken or not taken. Obviously static schemes will not satisfy the strict requirement of prediction accuracy for high performance superscalar processors. Another trivial way is to just use branch-target buffer (BTB) by recording taken branches in a buffer. The lower order bits of the instruction are used to index the buffer. If there is a hit, it predicts taken. Otherwise, it predicts not taken [2]. BTB usually gives about 80% accuracy whereas more accuracy (>95%) is required in modern superscalar processor design [6]. Hence, a more advanced design of branch predictor has to be defined.

This paper is explained as follows: Section 2 discusses the branch classification; section 3 introduces the proposed techniques to alleviate aliasing; section 4 discusses the simulation methodologies and results; section 5 gives the conclusions.

2. Branch Classification

Many researchers have studied branch prediction strategies extensively. To summarize, branch predictors fall into two categories. One is using a local history, in which the prediction is made solely based on the history of that branch itself. The other is using a global history, in which the history of the last few branches determines the direction of the current branch. However, the last few branches may not necessarily be the same as the current branch. Each class is suitable for certain types of branch. For example, simple branch such as for-loop can be better predicted by local history whereas high correlated branch such as if-else will get better results from global history based predictor. In general, most research indicate that global history predictors tend to give better results than local history since correlated branches seem to occur more than simple loop branches in the modern programs and benchmark programs [4, 7].

A conventional branch predictor in a global history might be the one proposed by Yeh and Patt [7], in which the global pattern history of last m branches is used to index the predictor table. This scheme is illustrated and compared to local history buffer in Figure 1. The global shift register used to index predictor table is m-bit wide, which records the branch behavior of the last m branches occurred. Once the current branch is resolved, its behavior will shift into global register. Both local and global predictors use 2-bit wide table entry, which is essentially an up-down saturating counter.

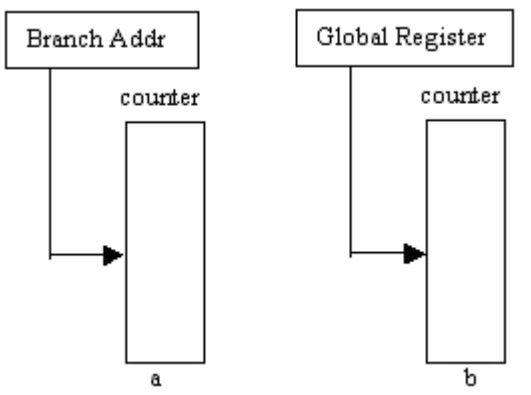

Figure 1. (a) Local history predictor vs. (b) global history predictor

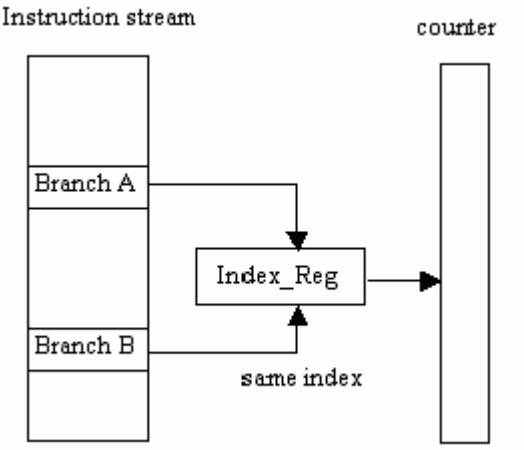

Figure 2. Aliasing problem for both schemes

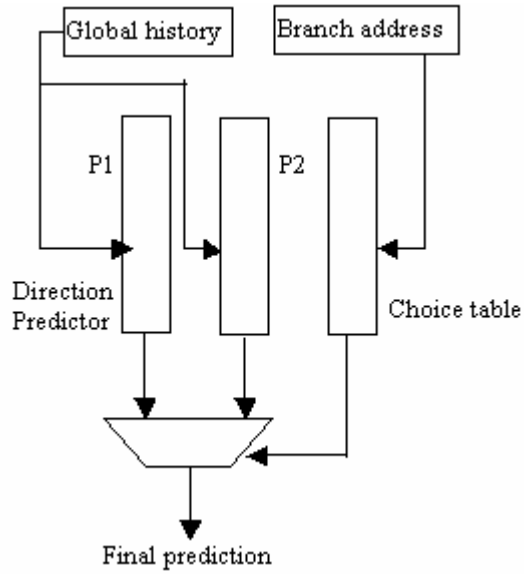

Figure 3. Split global history prediction

However, the way to index the table entry is different. After the branch is resolved, the corresponding entry will be updated. A serious problem in both schemes is that different branches could map to the same entry, which is called aliasing [9]. Aliasing is shown in Figure 2. If the two aliasing branches happen to bias to the same direction, they will not interfere with each other when updating the predictor table. This is called neutral aliasing [9], which does not cause a misprediction. However, if these two branches have different biasing branch direction, one updating the table entry will hurt the prediction for the other. This is destructive aliasing [9]. Solving the destructive aliasing problem is the key to improving predictor performance.

The simplest way is to increase the number of entries in the table so that fewer branches will map to the same entry, but this directly increases the cost of resources, and it is obviously not possible to use unlimited lines of the table. In this paper, we proposed and compared four techniques to alleviate the aliasing problem by using global and/or local histories.

3. Proposed Techniques to alleviate aliasing

The first technique to lower aliasing is called '*Correlating'*, in which the history table was split into m tables. Each table has the same number of entries, n. Instead of solely using local branch address or global history to index the history table, both global history (upper parts of index) and local address (lower parts of index) will be used. By using both, different branches will index to other entries even if they have the same global history pattern. This can eliminate most of the aliasing problem. The disadvantage is the number of global history bits has to be decreased and used to include local branch address to index the table. Since it is known that global history is superior to local history in predicting, reducing the bits of global history might affect the performance.

Split global history, *'Split'*, is another method to lessen the aliasing problem. As we discussed in Figure 2, there are two cases when two different branches have the same global history, neutral aliasing and destructive aliasing [9]. The purpose of Split is to separate destructive aliasing since aliasing does not impose negative effect. In this strategy, only global history is used to predict branch direction, but the whole global history pattern table is split into two tables of the same size. The two-predictor tables labeled P1 and P2 are shown in Figure 3. The global history shift register indexes both P1 and P2. The third table called choice table determines the final result. The choice table is indexed by the branch address. The entries in the choice table are also two bits wide up-down saturating counter. In the method of split history, the choice table will be updated according to the resolved branch direction.

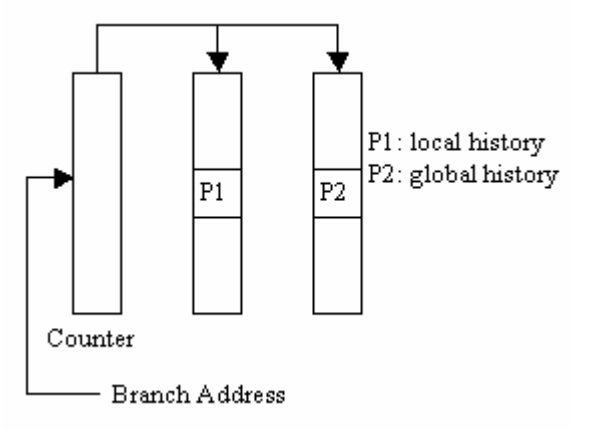

Figure 4. Combined branch predictor

Since destructive aliasing branches will bias in different directions, their corresponding entry in the choice table will choose different predictor, P1 or P2. These aliasing branches are ultimately separated to use different predictors. Additionally, only the selected table is updated as to not effect the other predictor. Hence, destructive aliasing branches are separated and should not interfere with each other.

The third way is called *'Combine'* which is based on the dynamic prediction [4] in selecting the direction of a branch by using 2-bit counters (*not taken: 00 and 01, taken: 10 and 11*). In this scheme, two different predictors are established, P1 and P2. Unlike the split global history, P1 and P2 are different predictors that could be any combination of effective predictors. This paper takes P1 as a local history predictor and takes P2 as a global history predictor. The two predictors are used in tandem to predict the branch direction. The underlying principle is that different types of predictors usually fit a certain types of branches whereas it is not suitable for every type of branches. By combining two types of predictors and making a decision based on which one is better suited to the certain types of branch, the prediction accuracy could be greatly improved. The key is how to select the more suitable predictor whenever a branch instruction occurs. As illustrated in Figure 4, an additional table containing 2-bit up-down saturating counters is used to select the best predictor. The branch address is used to index the selection table. The easiest way to make the selection is to just use the sign bit of the counter. If it is '1', P2 is selected, if it is '0', P1 is selected. The resolved behavior will update the counter. If P1 predicts right while P2 is wrong, then the counter is decremented. If P2 gets the right prediction while P1 is wrong, then the counter is incremented. If both P1 and P2 predict the same result whatever it is right or wrong, nothing is updated to the counter. We found that a particular branch would bias to the predictor that fits its behavior.

The fourth way is called *'Xor'* which can improve the performance of global history predictor from Yeh and Patt [7,8]. The idea is to take the local branch address and XOR it with global history pattern register to utilize the table by dispersing the addresses. This is a cost effective way to reduce the aliasing problem. Although the destructive aliasing branch instructions have the same global history pattern, they now are mapped to different entries in a global history pattern table. This could alleviate the aliasing issue to some extent, but this method's most important attribute is its cost for hardware implementation.

4. Simulation results

We implemented six types of predictors in this paper: 2 conventional branch predictors (*Bimodal and Global*) and 4 proposed predictors (*Correlating, Split, Combine, and Xor*). We modified Simplescalar to implement all the predictors and used SPEC CINT95 benchmark programs. Predictor table size is in the unit of bits. Figure 5 shows the results of both the local history prediction and global history prediction. Local history prediction, also called *Bimodal*, is saturated on the prediction rate after the table size is above 2K bits. Obviously, the global history prediction (called *Global*) outperforms local history prediction. This is because most of branch instructions tend to be correlated. The previous branches usually determine the direction of the following branches.

Although global history results in better prediction accuracy than local history, it still suffers the aliasing problem. The other four predictors presented in this paper use different techniques to try to eliminate aliasing. No matter what technique is adopted the ultimate purpose is to have aliasing branches map to different predictor table entries so they do not interfere with each other. Figure 6 gives the results of these four predictors versus a common global history predictor (*Global*).

It is observed that the combined predictor (*Combine*) has the best prediction accuracy when the predictor size is smaller than 8K. After that, *Xor* shows almost the same performance as the combined predictor. As the size keeps increasing, the performance of Correlating predictor and Global history predictor catch up after 16K. It is also clear that only the Xor and Combine predictors perform better than the Global predictor whereas Correlating and Split predictors do not. As described previously, Correlating predictor takes bits from both local address and global history pattern to index predictor table. Although this would be beneficial to the aliasing issue, it reduces the number of bits taken from the global history pattern as well when the same size of predictor table is used. Hence, it ultimately degrades the performance of the predictor.

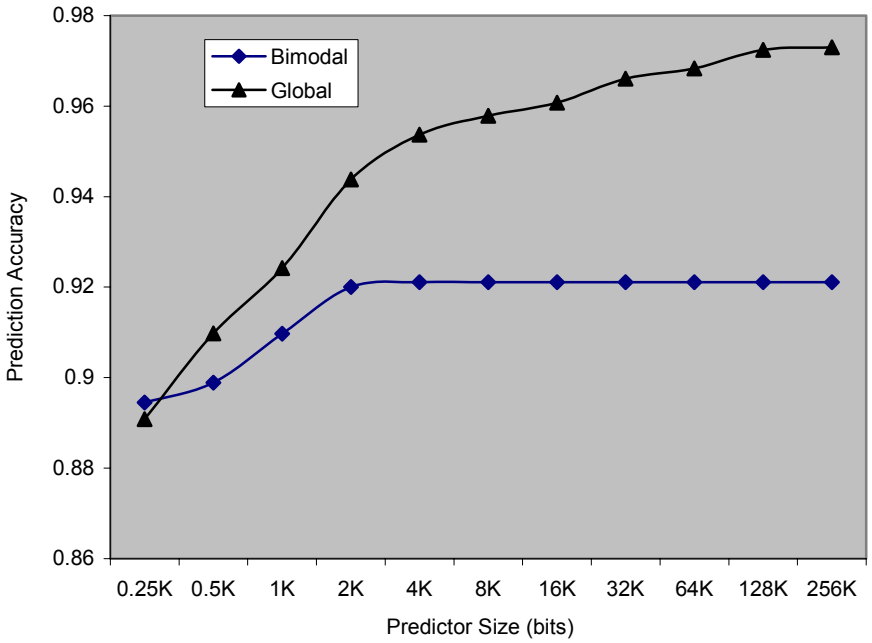

Figure 5. Performance of Bimodal predictor and Global history predictor

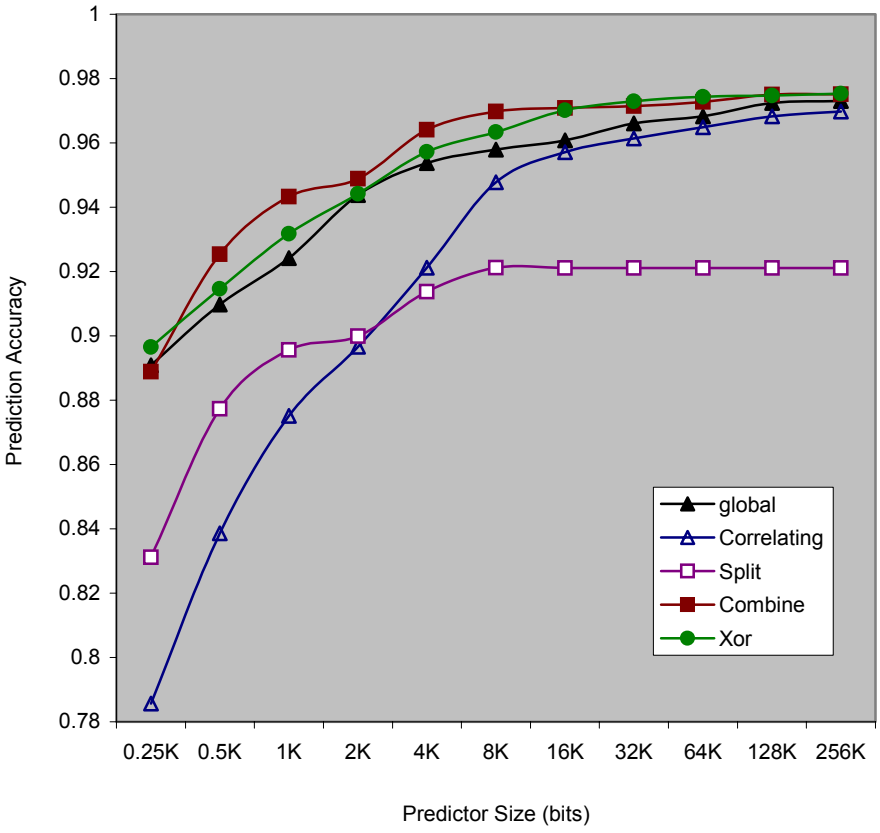

Figure 6. Performance of Global history predictor vs. other variations of predictor

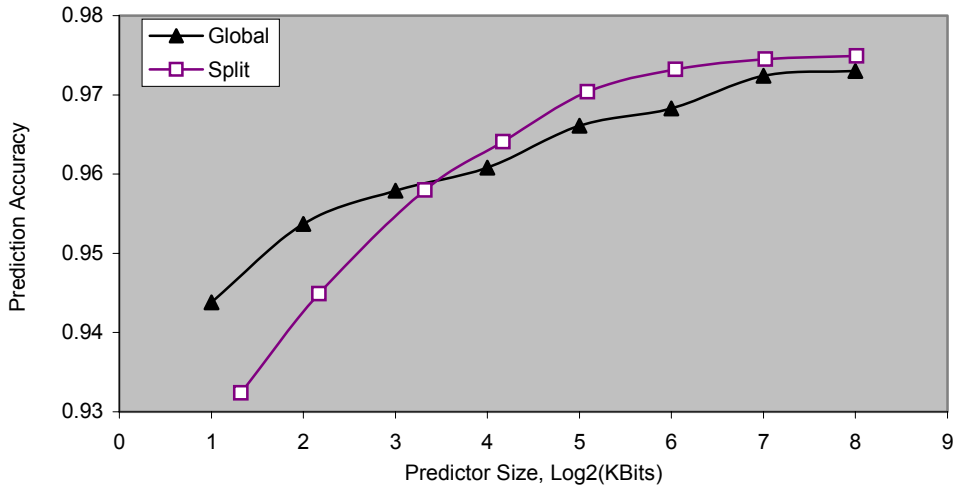

Figure 7. Performance of modified split predictor vs. global history predictor

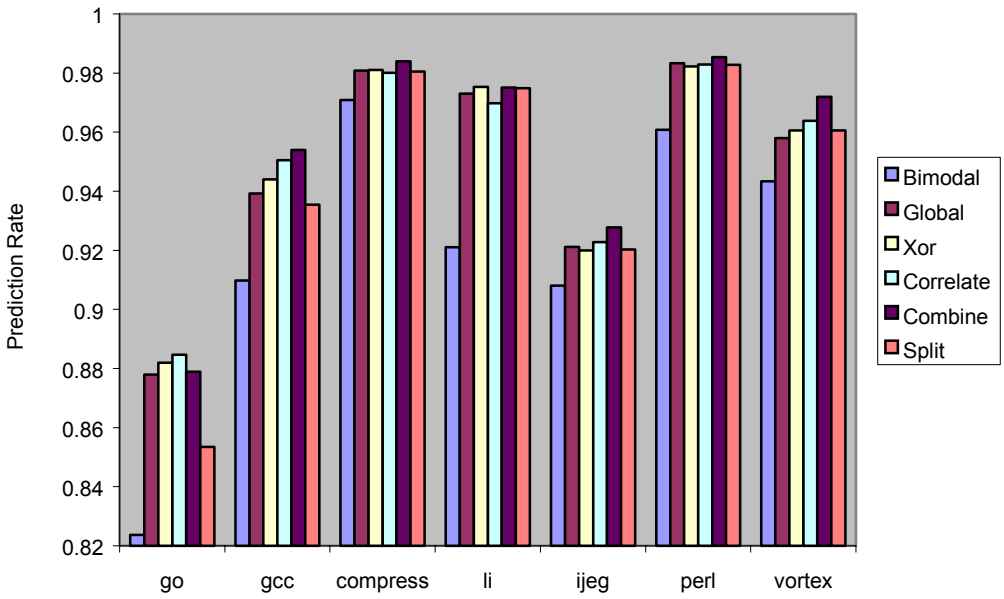

Figure 8. Performance of six predictors with size of 256K bits (except for Split having 258K bits)

Since we want to obtain a fair comparison between various predictors versus global history predictor, each predictor is forced to have the same number of bits used.

In terms of Split predictor, there are three tables in use. One is the choice table; the other two are prediction tables that are exactly the same. In order to have the same bits used in the tables, the choice table is twice size of prediction table so the total sum of bits used are a power of 2. Doing this is both beneficial and detrimental; the bigger choice table is helpful to relieve aliasing problem, but it also wastes bits that could be used for prediction. This problem is magnified because the Global predictor is in two pieces in order to separate destructive aliasing branches thus reducing capacity in half. Simulation results illustrate that taking a half of bit resources to resolve aliasing apparently exaggerates the problem. Based on this the configuration of Split predictor needs to be modified. The choice table contains small enough number of entries compared to the prediction table. Figure 7 shows the results of the new configuration for the Split predictor together with Global history predictor. The number of lines in the choice table is 256 when the total size of table is less than 5K bits. For the total table size equal to or above 5K, the number of entries used in choice table is 1024. It is clear

that the performance of Split predictor starts to exceed Global history predictor after the total size of predictor is above 8K. Figure 8 gives the results of performance of all the six predictors with size of 256K bits except for Split history predictor that has 258K bits. Obviously, predictors based on the global history perform better than local history predictor. Among the variations based on global history, Combined predictor seems to give the best accuracy.

5. Conclusions

This paper investigates several types of branch predictors. It started with local history predictor and global history predictor. Local history is more suitable for the branches that are just simple loops - the behavior of loop branches depends on its own history. Global history predictor exploits another characteristic of branches; they are highly correlated. The history of past branches usually determines the direction of next branch. Simulation results show that global history predictor gives better performance than local history, which shows that branches tend to be correlated.

Although global history predictor is good at predicting the correlated behavior of branches, it still suffers the destructive aliasing problem because it does not use the branch address. As stated before, aliasing is when different branches have the same global history pattern are mapped to the same predictor entry. If they display different branch bias directions, then they will interfere with each other due to the alternate update of the same entry. Separating the aliasing branches is a key to improve the performance of global history predictors. Four techniques were presented: Correlating, Split, Combine, and Xor. Although each technique is different in implementation, the common idea is to integrate the local branch address into the global history shift register so that both local history and global history can affect the prediction. Global history is better in predicting whereas local branch address is effective in separating aliasing branches

Our experimental results show that Combine has the best prediction accuracy when the prediction table is less than 8K bits. If the size is more than 8K bits, Xor works slightly better than (or almost similar to) Combine. Basically, Combine and Xor can improve branch prediction accuracy more than conventional branch predictors that use global and/or local history branch addresses.

One aspect needs to be pointed out is that some techniques implement integration by adding bits from the local address to the global history register. For a fixed size table, adding bits from branch address means reducing the same number of bits from the global history register. That is why Split does not work well for the smaller size (less than 8K bits) compared to the predictor that use global history.

6. References

[1] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach"*, Morgan Kaufmann Publish, New York, N.Y., 1993

[2] J.E. Smith, "A Study of Branch Prediction Strategies", *Proceedings of the 8th International Symposium on Computer Architecture,* May 1981, pp. 135-148.

[3] W.W. Hwn, T.M. Conte, and P.P. Chang, "Comparing Software and Hardware Schemes for Reducing the Cost of Branches", *Proceedings of the 16th International Symposium on Computer Architecture*, May 1989

[4] S. McFarling and J. Hennessy, "Reducing the Cost of Branches", *Proceedings of the 13th International Symposium on Computer Architecture*, 1986, pp. 396-403

[5] J. Lee and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", *IEEE Computer*, Jan. 1984, pp. 6-22

[6] P.Y. Chang, E. Hao, T.Y. Yeh and Y. Patt, "Branch Classification: a New Mechanism for Improving Branch Predictor Performance", *Proceedings of the 27th Annual International Symposium on Microarchitecture*, 30 Nov.-2 Dec. 1994, pp. 22-31

[7] T.Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction", *Proceeding of the 19th Annual International Symposium on Computer Architecture*, May 1992, pp. 124-134

[8] T.Y. Yeh and Y.N. Patt, "A Comparison of Dynamic Branch Predictors that Use Two-Levels of Branch History", *Proceeding of the 20th International Symposium on Computer Architecture*, May 1993, pp. 257-266

[9] A.N. Eden and T. Mudge, "the YAGS Branch Prediction Scheme", *Proceeding of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, Nov 1998, pp. 69-77