Analysis of Combined Bimodal and GShare Branch Prediction Schemes

Bhavya Daya
Electrical Engineering Department, University of Florida

Abstract

Taking advantage of instruction level parallelism is one of the key factors in improving computer performance. The main performance inhibitor in a pipelined processor is the branch penalty. Many techniques have been developed to reduce this penalty. The technique of branch prediction was explored further. Branch predictors are crucial in today's modern, superscalar processors for achieving high performance. They allow processors to fetch and execute instructions without waiting for a branch to be resolved. As a result the performance of the processor improves. Two branch prediction schemes were implemented and both resulted in an increase in performance. Combining these predictors and providing a selector to choose the most appropriate prediction scheme was implemented. A performance increase was observed with the combined prediction scheme versus the individual schemes alone. A local and global prediction scheme was utilized, the bimodal prediction served as the local predictor and the gshare prediction served as the global predictor. When combined, the prediction is as accurate at the local prediction but also takes into account dependencies on previous branches.

1. Introduction

Branches hurt pipeline performance because of the need to enforce control dependencies. These control dependencies are dealt with by stalling the pipeline. If the behavior of the branches can be predicted the performance losses incurred by branching will be reduced. The branch performance problem can be divided into two sub-problems. First, a prediction of the branch direction is needed. Second, for taken branches, the instructions from the branch target must be available for execution with minimal delay. One way to provide the target instruction quickly is to use a Branch Target Buffer, which is a special instruction cache designed to store the target instructions. The prediction strategies were focused on.

There are two main categories of branch prediction schemes, static and dynamic. Static prediction is completed before execution of the program. The prediction is not based on the history of previous branches. Dynamic prediction occurs during runtime. This prediction changes depending on the program being executed. Based on previous branching history, prediction is performed for the current branch. Dynamic prediction can be further divided into two types, local and global prediction. Both types were implemented and analyzed. The most well known local prediction technique, referred to as bimodal branch prediction, makes a prediction based on the direction the branch went the last few times it was executed. Global prediction technique uses the combined history of all recent branches in making a prediction. The bimodal technique works well when each branch is strongly biased in a particular direction. The global technique works particularly well when the direction taken by sequentially executed branches is highly correlated.

The local bimodal prediction scheme and global gshare scheme were implemented separately and combined for comparison purposes. The experiment results revealed that the two predictions schemes, when combined, yielded a better performance increase than when they were individually implemented. Advantages of both techniques are utilized in the combined branch prediction scheme. The strongly biased branches are accounted for as well as the correlation of branches.

Gshare is a variation on the basic global prediction scheme. The global share (gshare) algorithm uses two levels of branch-history information to dynamically predict the direction of branches. The first level registers the history of the last k branches faced. This represents the global branching behavior. This level is implemented by providing a global branch history register. This is basically a shift register that enters a 1 for every taken branch and a 0 for every untaken branch. The second level of branch history information registers the branching of the last s occurrences of the specific pattern of the k branches. This information is kept in the branch prediction table. The gshare algorithm works by taking the lower bits of the branch target address and exclusive oring them with the history register to get the index that should be used with the prediction table.

2. Related Work

Abundant research has been done on branch prediction. Now, combined branch prediction or hybrid prediction is emerging everywhere. The work mostly used was by McFarling [1] on combined branch predictors has led to the investigation performed in this paper. There are many dynamic branch prediction schemes.

The Pattern History Table (PHT) is limited in size, resulting in many branches mapping to the same location in the PHT. This causes several branches to address and change values in the same counter. This is called aliasing or interference.

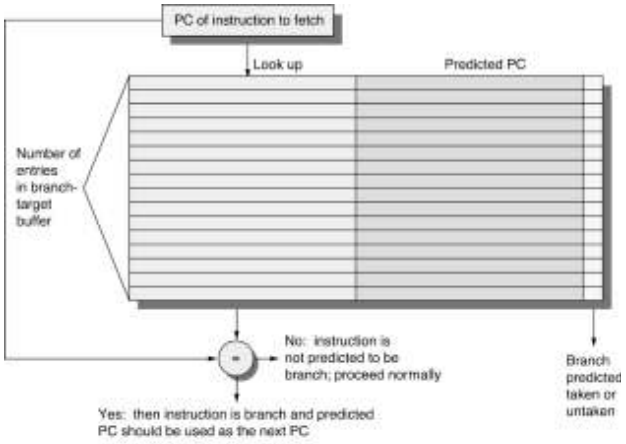
The Gshare scheme is one method of reducing aliasing but other methods are the agree predictor, bi-mode predictor, and the YAGS predictor.

YAGS, Yet Another Global Prediction Scheme, is implemented to reduce the amount of aliasing that occurs in the PHT. The reduction is performed by introducing tags into the PHT [3].

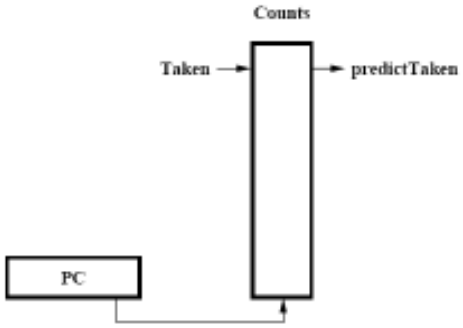
A lot of work has been done on hybrid predictors, or combined predictors. The prophet/critic hybrid prediction scheme contains an interesting methodology [2].

3. Methodology

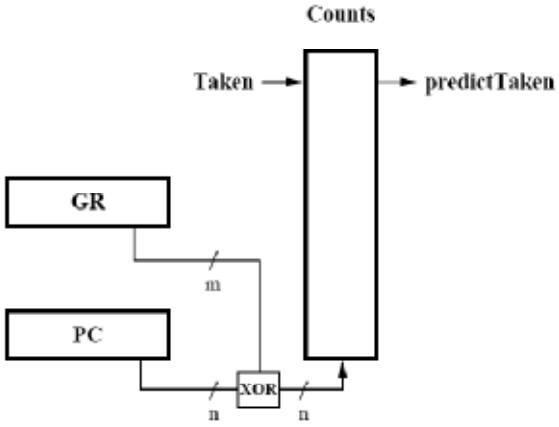
Before discussing the method for implementing the prediction schemes, the branch targeting needs to be addressed. The prediction is useless unless the target address to branch to is known before hand as well. The simplest approach used when implementing the branch target buffer is to index the buffer by the program counter value. Each slot in the buffer will hold the value of the last target address of the branch instruction corresponding to a particular branch instruction. The branch target buffer utilized can only hold indices up to 1024 values. The entire program counter value is used to index the buffer. For simulation purposes, 1 K buffer size seemed adequate. The figure below displays a diagram of the implementation of the branch target buffer.

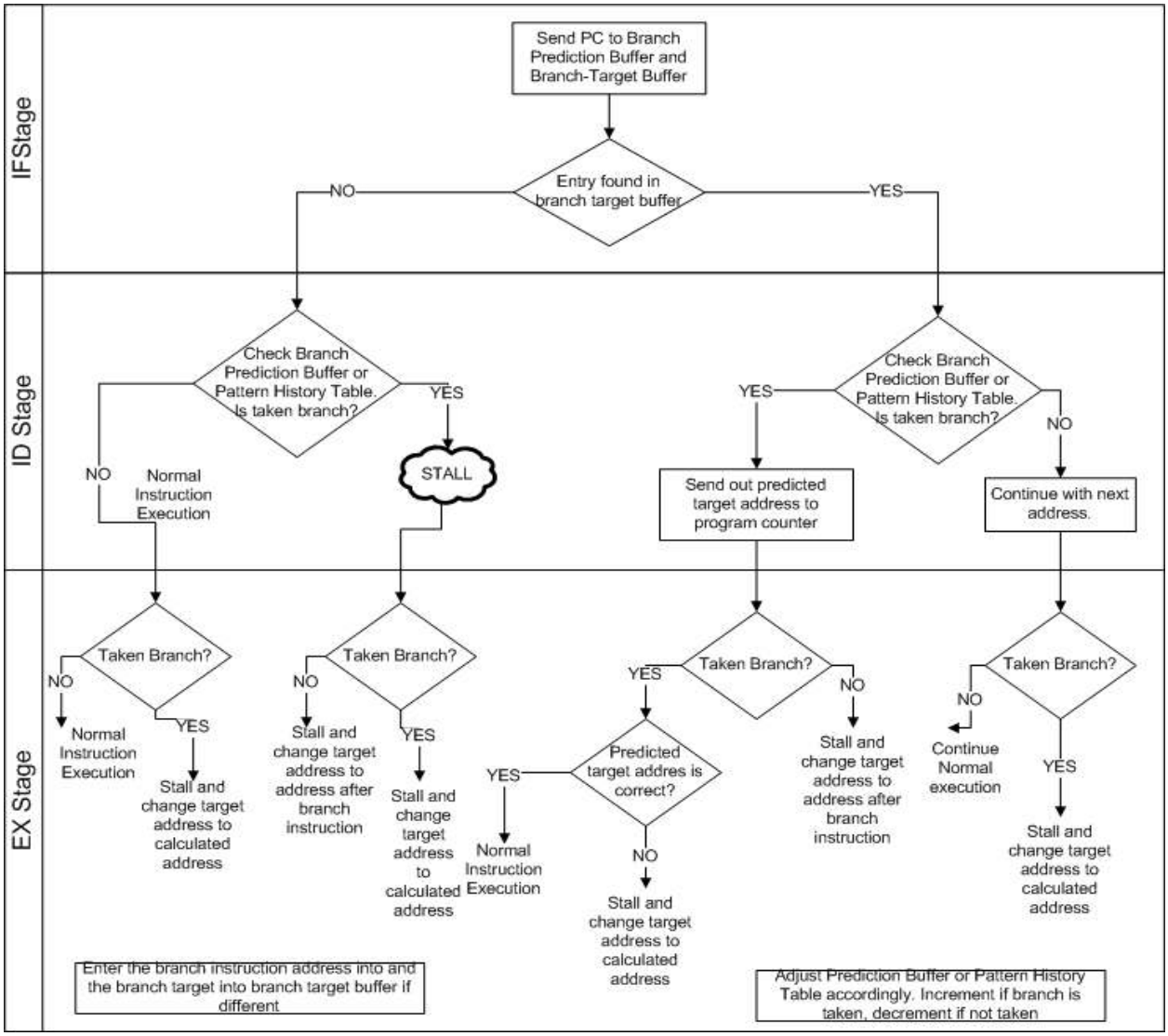


The predicted taken or untaken aspect of the target buffer is initially implemented using a bimodal prediction technique. The bimodal scheme requires the use of a prediction buffer indexed by the program counter value.



The global prediction scheme, gshare, uses both a global history register and the program counter to index the prediction buffer.

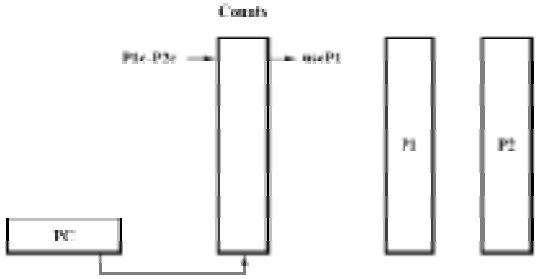




The global history register is a shift register that shifts in the value of one if the branch was taken and a value of zero if the branch was not taken. Previous work has shown that the global history information is less efficient at identifying the current branch than simply using the branch address. The suggestion is that a more efficient prediction might be made using both the branch address and the global history. The flowchart above shows the steps taken in implementation of the branch prediction for a pipelined architecture. The diagram is similar for both the global and local prediction schemes. The difference is the indexing method utilized. Instead of checking the branch prediction buffer, the pattern history table is checked for the global scheme. The size pattern history table developed is 256 entries. The global history register can shift in eight bits only. The program counter is assumed to be less than 256. Based on the layout of the value of zero is given to that prediction. The difference between the two predictions is taken and stored into the pipeline architecture, there can be a total of 64 instructions from the program counter and the shift register can range from 0 to 255. This seemed sufficient for testing and comparing the performance. If larger programs need to utilize this processor, then a larger global history register and pattern history table will be utilized. The prediction buffer chosen, for local prediction, will also have to be larger when running large applications.

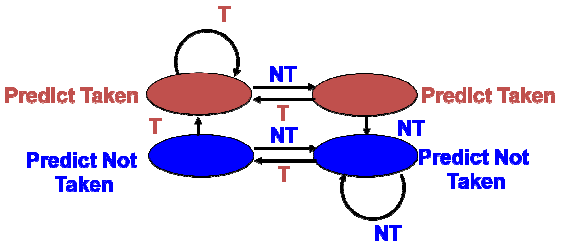
The combination of these two methods of prediction was performed by using a buffer. The buffer is indexed by the program counter and based on the value in the buffer the choice is made to use prediction one, local, or prediction two, global. The buffer is populated by comparing the branch predictions with the actual branch. If the prediction is correct, a value of one is given to that prediction, else a buffer. The difference, P1-P2, reveals that if a positive value is in the buffer then prediction one is correct. If a

negative value is in the buffer then prediction two is correct. If the value is zero, then either prediction method can be used. The figure below visually describes the combined prediction. The value in the buffer is assumed to be positive, therefore prediction one is chosen.



In the flowchart for the implementation, the buffer chooses whether to check the prediction buffer or the pattern history table. Also, the buffer chooses which prediction buffer to update as well.

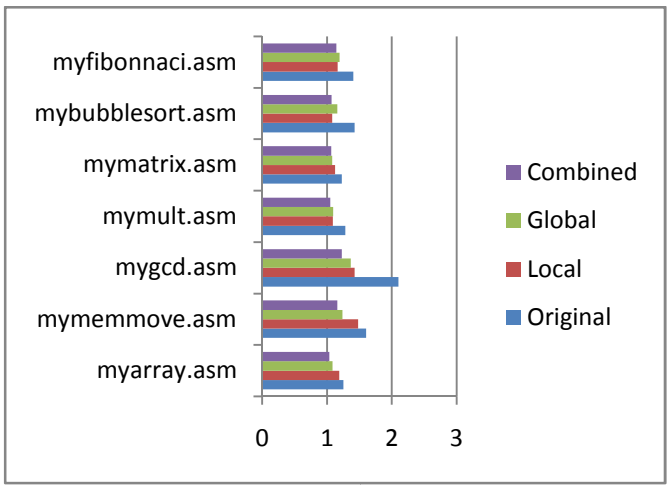
Each prediction buffer is composed of two bits. If the value is two or three, the branch is taken. If the value is zero of one, the branch is not taken. The figure below summarizes the function of the prediction buffer.



4. Experiment Results

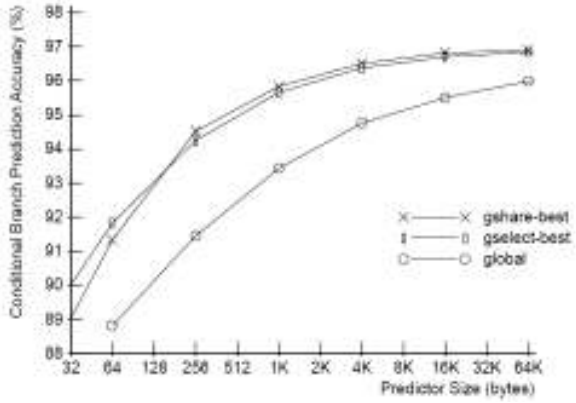
Different benchmarks were run and the performance of the system in terms of clock cycles per instruction (CPI) was compared to the original system.

	Original	Bimodal	Gshare	Combined
myarray.asm	1.25287	1.19157	1.08481	1.03802
mymemmove.asm	1.60465	1.47857	1.23841	1.16084
mygcd.asm	2.10204	1.42553	1.3662	1.23077
mymult.asm	1.28308	1.09212	1.09371	1.05098
mymatrix.asm	1.22824	1.1236	1.08065	1.06654
mybubblesort.asm	1.42527	1.08284	1.15964	1.07059
myfibonnaci.asm	1.40741	1.16667	1.19355	1.14545



The following figure and table above shows the comparison of the approaches. For each benchmark tested, it can be seen that the combined predictor increases performance more than each predictor alone. Although in some cases, the combined predictor was not much of an improvement, when it comes to larger benchmarks the combined prediction scheme may show a larger performance increase.

The accuracy of the gshare prediction scheme is approximately 94.5%. This value was obtained from previous work performed by McFarling on combined predictors. The following figure is used to determine the accuracy based on the predictor size.



Based on the results, the predictor size seems sufficient for testing small benchmarks or running small applications. For the bubble sort and fibonnaci benchmarks, the global scheme is worse than the local scheme. The combined predictor chooses the most appropriate predictor, therefore the performance improves.

5. Conclusions and Future Work

Individual schemes were compared to the combined scheme. The results were as expected; the combined scheme would yield a better performance increase.

Future work will be to see how the size of the prediction buffer affects the combined prediction scheme. Different benchmarks will be implemented and used for comparison purposes.

Another scheme to reduce aliasing in the prediction table should be developed and compared with schemes that are currently available.

6. References

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

[2] Ayose Falcon, Jared Stark, Alex Ramirez, Konrad Lai, Mateo Valero, "Prophet/Critic Hybrid Branch Prediction," *isca*, p. 250, 31st Annual International Symposium on Computer Architecture (ISCA'04), 2004.

[3] A. N. Eden, T. Mudge, “The YAGS branch prediction scheme,” Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture, p.69-77, November 1998, Dallas, Texas, United States