Branch Prediction Techniques and Optimizations

Raj Parihar
University of Rochester, NY, USA
parihar@ece.rochester.edu

Abstract

Branch prediction is one of the ancient performance improving techniques which still finds relevance into modern architectures. While the simple prediction techniques provide fast lookup and power efficiency they suffer from high misprediction rate. On the other hand, complex branch predictions – either neural based or variants of two-level branch prediction – provide better prediction accuracy but consume more power and complexity increases exponentially. In addition to this, in complex prediction techniques the time taken to predict the branches is itself very high – ranging from 2 to 5 cycles – which is comparable to the execution time of actual branches.

Branch prediction is essentially an optimization (minimization) problem where the emphasis is on to achieve lowest possible miss rate, low power consumption and low complexity with minimum resources. In this survey paper we review the traditional Two-level branch prediction techniques; their variants and the underlying principles which make them predict accurately. We also briefly discuss the Perceptron based technique which uses lightweight neural network technique to predict the outcome of branches.

Keywords: *Two-Level branch prediction, Global – History Register (GHR), Pattern History Table (PHT), Neural Branch Predictors*

1. Introduction

Branches change the normal flow of control in programs in unexpected manner which interrupts the normal fetch and issue operation of instructions. To resume the normal fetch it takes considerable number of cycles until the outcome of branches and new target is known. Branches are very frequent in general purpose programs – around 20% of the instructions are branches – and simplest solution to deal with branches is to stall the pipeline. Stalling pipeline doesn't violate the correctness of program. However, it does degrade the overall IPC which is even severe in longer pipelines. In order to maintain the high throughput branch prediction in modern high performance microarchitecture has become essential.

Static branches are not hard to predict and in most of the systems, compilers can provide very good coverage for such branches. Branches whose outcome is determined on run-time behavior are difficult to predict using compilers. In general, the branch outcomes are not random and they do have some sort of bimodal distribution. For such dynamic branches it is advantages to use a prediction mechanism which adapts in runtime and captures the run-time state of the system in order to make the predictions [1]. Although dynamic branch prediction techniques do provide very good prediction accuracy but there are still some caveats.

The thrust for higher performance coupled with the predictability of branch outcomes suggest to use some prediction mechanism to know the branch outcome even before the branches are executed. Even after having a good predictor, in most of the cases, the penalty of branch misprediction is as bad as stalling the pipeline. This is the reason precisely why we need really smart branch predictors.

Section 2 is an overview of basic branch prediction techniques. Section 3 presents a brief description of branch predictors which are incorporated in various commercial designs. In section 4, some of the performances impacting factors are discussed. Overview of neural-based branch predictors is presented in section 5. Section 6 concludes the survey and serves as high-level guideline for designing the more accurate branch predictors.

2. Basic Branch Prediction Schemes

2.1. Bi-Model Branch Predictors

A bimodal branch predictor has a table of n-bit entries, indexed with the least significant bits of the branch addresses. Unlike the caches, bimodal predictor entries typically do not have tags, and so a particular entry may be mapped to different branch instructions (this is called branch interference or branch aliasing), in which case it is likely to be less accurate.

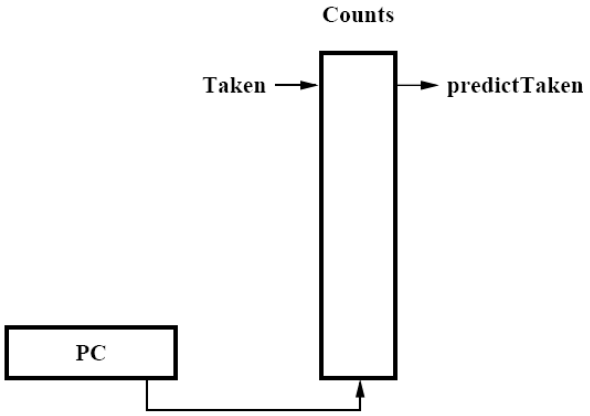

Fig1: Bimodal branch predictor

The typical hit rate of bimodal branch predictors with 4KB entries is around 90% for SPEC95 benchmarks [2].

2.2. Local Branch Predictors

Bimodal branch predictors mispredict the exit of every loop. For loops which tend to have the same loop count every time (and for many other branches with repetitive behavior), we can do much better. Local branch predictors keep two tables [2]. The first table is the local branch history table. It is indexed by the low-order bits of each branch instruction's address, and it records the taken/not-taken history of the n most recent executions of the branch.

The other table is the pattern history table. Like the bimodal predictor, this table contains bimodal counters; however, its index is generated from the branch history in the first table. To predict a branch, the branch history is looked up, and that history is then used to look up a bimodal counter which makes a prediction.

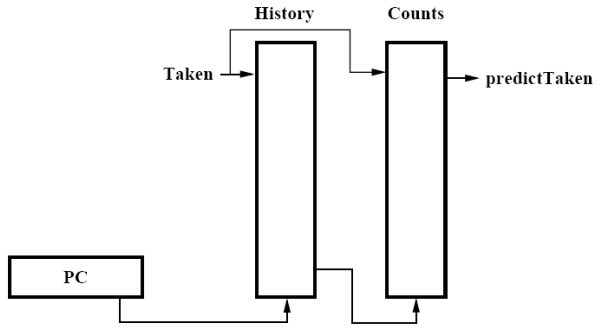

Fig2: Local branch predictor

Local branch predictors provide slightly better hit rate than bimodal predictors.

2.3. Global Branch Predictors

Global branch predictors make use of the fact that the behavior of many branches is strongly correlated with the history of other recently taken branches [1]. We can keep a single shift register updated with the recent history of every branch executed, and use this value to index into a table of bimodal counters. This scheme, by itself, is only better than the bimodal scheme for large table sizes.

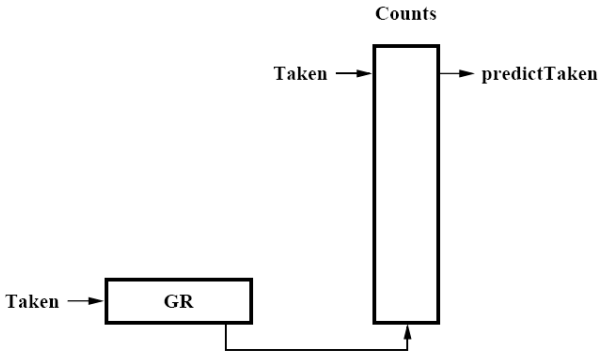

Fig3: Global branch predictor

2.4. Combining Branch Predictors

Scott McFarling proposed combined branch prediction in his 1993 paper [2]. Combined branch prediction is about as accurate as local prediction, and almost as fast as global prediction. Combined branch prediction uses three predictors in parallel: bimodal, gshare, and a bimodal-like predictor to pick which of bimodal or gshare to use on a branch-by-branch basis.

The choice predictor is yet another 2-bit up/down saturating counter, in this case the MSB choosing the prediction to use. In this case the counter

is updated whenever the bimodal and gshare predictions disagree, to favor whichever predictor was actually right.

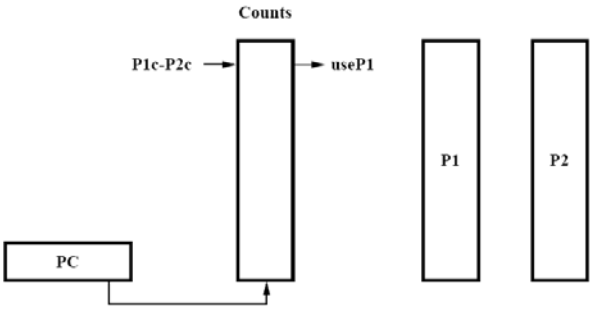

Fig4: Combined branch predictor

2.4. Agree Branch Prediction

Another technique to reduce destructive aliasing within the pattern history tables is an agree predictor [3]. In this technique, a predictor (e.g. a gskew predictor) makes predictions, but rather than predicting taken/not-taken, the predictor predicts agree/disagree with the base prediction. The intention is that if branches covered by the gskew predictor tend to be a bit biased in one direction, perhaps 70%/30%, then all those biases can be aligned so that the gskew pattern history table will tend to have more agree entries than disagree entries.

This reduces the likelihood that two aliasing branches would best have opposite values in the PHT.

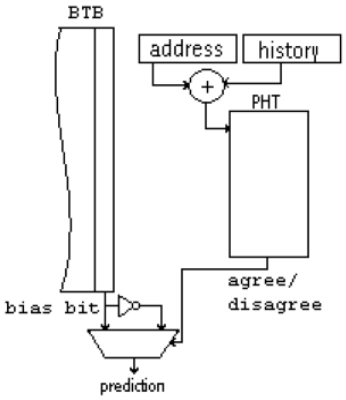

Fig5: Agree branch predictor

The focus is on to reduce the amount of negative aliasing and either to convert it to positive of neutral. Agree predictors work well with combined predictors, because the combined predictor usually has a bimodal predictor which can be used as the base for the agree predictor. Agree predictors do less well with branches that are not biased in one direction, if that causes the base predictor to give changing predictions. So an agree predictor may work best as part of a three-predictor scheme, with one agree predictor and another non-agree type predictor.

3. Commercial Branch Predictors

This section presents a brief overview of some of the state-of-art branch predictors which has been incorporated into the commercial designs.

3.1. POWER4 Branch Prediction

POWER4 prediction unit is very similar to combined predictor proposed by McFarling. POWER4 branch prediction unit consist of three set of branch history tables. First one is called local predictor (or Branch history table) which has 16 K entry and each entry is 1 bit. Second table is called global predictor which has 11-bit history register (or history vector). The content of history vector is XORed with branch address before indexing the history table. Apart from these two tables there is another table which is known as selector table. This has 16 K 1-bit entries and keeps track of better predictor.

In POWER4, fetching is re-directed based on prediction outcome. Eventually few cycle later branches are executed in BR unit. Upon execution of branches the predictor tables are updated if the prediction was not correct. Another feature which POWER4 provides is that dynamic branch prediction can be ***overdriven*** by software, if needed. In order to know the target of branch Link stack predict the target of branches.

3.2. Alpha 21264 Branch Predictors

Similar to POWER4 Alpha 21264 branch predictor is also composed of three units – Local predictor, Global predictor, and Choice predictor. Local predictor maintains the per-branch history and each entry is 2-bit saturation counter. There are total 1 K entries in local predictor. Global predictor uses 12 most recent branches to predict the outcome of a branch. There are total 4K entries in global predictor and each entry is 2-bit wide.

Choice predictor monitors the history of local and global predictors and chooses the best of two. It has 4K entry and each entry is 2-bit saturation counter. Block diagram of Alpha 21264 is shown in Fig 6.

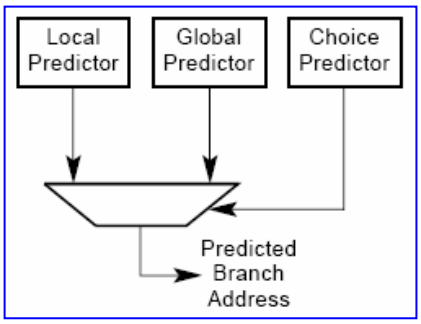

Fig6: Alpha 21264 branch prediction unit

3.3. Intel Branch Predictors

386 and 486 didn't have any sort of hardware based dynamic branch prediction block. All branches were statically predicted as NOT TAKEN. Pentium III has a two-level of local history based branch predictor where each entry is 2-bit saturating counter (also known as Lee-smith counter). Pentium M combines three branch predictors together – Bimodal, Global and Loop predictor. Loop predictor analyzes the branches to see if they have loop behavior.

4. Branch Prediction: Insights

This section presents some of the key factors which have great impact on performance of branch predictors.

4.1. Interference in Branch Predictions

Interference or branch aliasing refers to a situation where prediction mechanism uses same resource to predict the outcome of two branches (possibly two un-related and contradicting branches). Depending upon the correlation between those two branches interference could be either *positive* or *negative*. Two branches which behave completely different, if use the same resource, would create negative interference and the prediction accuracy would suffer. On the other hand, positive interference could improve the prediction accuracy. Neutral interference refers to a situation where two branches don't have any impact on each other.

From various studies it has been found out that in branch predictions the negative interference is more dominant than positive or neutral interference. Two-level branch prediction techniques with limited hardware resources suffered mainly because of negative interferences and subsequent research work tried to address this problem. The focus was to reduce the more dominant negative interference and convert it to either positive or neutral interferences. The techniques which tried to solve the problem of negative interferences i.e. Agree predictor, Skew predictor etc. achieved 30-50% improvement in misprediction rate over baseline two-level branch predictors.

4.2. Adaptive GHR Length

In general, branches show some sort of co-relation with their own previous outcomes or with the outcome of other branches. Co-relation with their own outcome is known as local co-relation whereas branches which are related with other branches known as globally correlation. Two-level techniques and their variant tried to exploit this behavior and achieved significant improvement on baseline branch predictor.

The branch predictors which exploit either the local or global co-relation use some sort of history table to make use of past outcomes. However, the main problem with the history based predictors is to determine the "appropriate amount" of history to predict the outcome. Determining the right history is not trivial and may require the profiling [6]. The following data shows the absolute number of Misprediction for 10 million instructions of various applications for different amount of history.

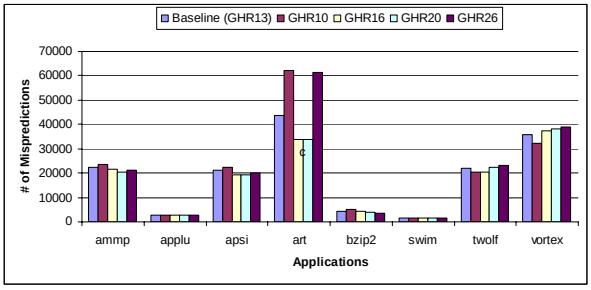

Fig 7: Misprediction for various history lengths

As we can see that by varying the amount of history used does improve or degrades the prediction accuracy. It is intuitive to think that larger history would give better accuracy which is not always true. The accuracy improves if we keep a larger history and then based on branch behavior use the appropriate amount of history. Neural branch prediction techniques try to use the appropriate amount of history by assigning the weights to them [5].

In another proposal branch predictor uses variable history register (table) size (lower or higher than PHT index) which is adjusted dynamically. Each branch address maintains an entry which tells the appropriate size of history to be used for that particular branch [6].

4.3. Utilization of Pattern History Table

Pattern History Table (PHT) keeps track of particular pattern and tries to use it for prediction when a similar pattern occurs again. The problem with PHT is that not all the entries are used uniformly. In general, some of the entries are used more often than others and which might be the cause for negative interference as well. The figure below shows the distribution of accesses of 8KB pattern history table for art application.

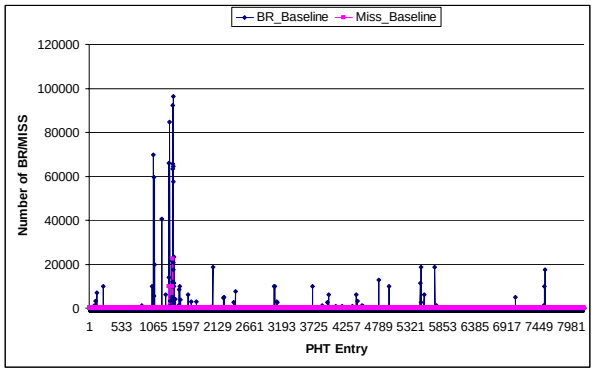

Fig8: PHT Utilization – Distribution of branches and misprediction per entry

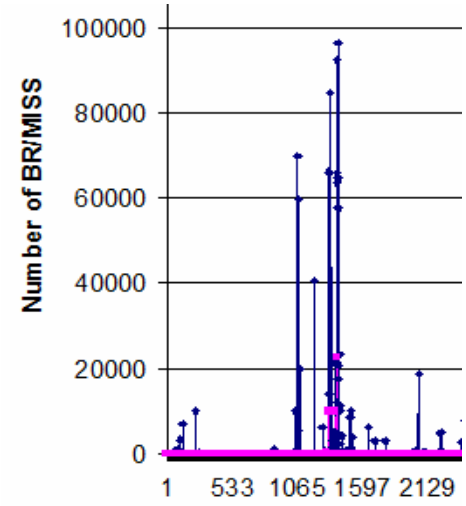

Fig 9: (Zoomed version of fig 8) PHT Utilization

4.4. Efficient Hash Function

The negative interference could be due to addressing of history table. The simplest form of history table addressing is indexing using branch addresses. If branch address merged with global history register it reduces the misprediction rate further. Another variation of this, known as gshare, does the XOR of branch history and branch address and then indexes the history table. Some of the more efficient hash functions enable the better use of PHT and the history is evenly distributed in table.

The key thing is that entries in PHT need to be distributed in uniform manner in order to avoid the negative interference. For this purpose variety of hash functions can be used. XOR is the simplest hash function but not efficient enough to provide the better scattering or allocations. Some of the common hash functions implements 6 to 7 level of shifting and XOR operations.

4.4. Impact of GHR on PHT Utilization

An appropriate amount of history also improves the utilizations of PHT which is shown in figure below. For different GHR the some of the branches are completely re-mapped into some other entries

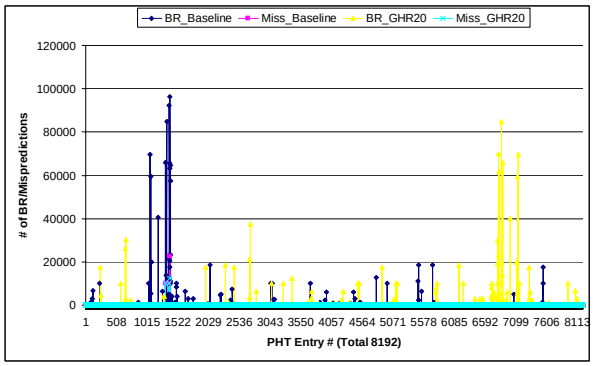

Fig 10: PHT Utilization – Distribution of branches and misprediction per entry for various GHR size

BR Baseline	Miss Baseline	BR GHR20	Miss GHR20
63351	22526	23256	12536
65558	0	3	0
0	0	1	0
6	0	0	0
0	0	0	0

Table 1: Taken from the history trace of PHT

5. Neural Branch Prediction: A Machine Learning Approach

The conditional branch prediction is a Machine learning problem where machine learns to predict the outcome of conditional branches. So the first natural question which arises immediately is that why not to apply a machine learning algorithm?

Artificial neural network tries to model the neural networks in brain cells. It is very efficient to learn to recognize and classify patterns. Similarly the branch predictor also uses the classification of branches and tries to keep the co-related branches together [5].

5.1. Perceptron Based Predictors

The perceptron is the simplest version of the neural methods. And for the even simplest perceptron, a single-layer perceptron that consists of several input units by weighted edges to one output unit as depicted in figure 11. The inputs to a neuron are branch outcome histories (x1,…,xn). A perceptron learns a target Boolean function t(x1,…,xn) of n inputs as shown in below.

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

In this case, the xi are the bits of a global HR, and the target function predicts whether a particular branch will be taken or not. Thus, intuitively, a perceptron keeps track of positive and negative correlations between branch outcomes in the global HR and the branch being predicted.

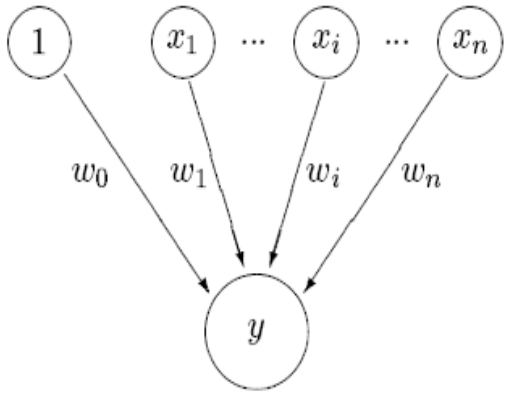

Fig 11: Perceptron based branch predictor

5.2. Advantages and Disadvantages

Neural prediction could be incorporated into future CPUs. Accuracy is very good however complexity is still a bottleneck. The main advantage of the neural predictor is its ability to exploit long histories while requiring only linear resource growth. Classical predictors require exponential resource growth.

The main disadvantage of the perceptron predictor is its high latency. Even after taking advantage of high-speed arithmetic tricks, the computation latency is relatively high compared to the clock periods of even modern Microarchitectures.

6. Conclusion

There are two aspects of branch predictions. First is to determine the outcome (Taken or Not taken) of branch and if taken then the knowledge of target address. A static branch prediction scheme relies on the information incorporated in prior to runtime whereas the dynamic (adaptive) branch prediction logic tries to extract the information and predicts the behavior of branches in run-time. Most of the adaptive predictors use two piece of information. The history of last k branches and their specific behavior s is taken into account in order to implement the high performance branch prediction logic. The reason why dynamic branch prediction beats static is because the kind of data appears in run time is very much different from the sample (trace) data which is used for profiling.

Bi-model predictor is simplest kind of predictor which uses the branch address and n-bit saturating counter based pattern table to predict the outcome. If miss predicted the pattern table entry is corrected by incorporating the actual outcome. In such predictors, the prediction accuracy is function of size of pattern table and the maximum achievable hit rate is 93 – 94%. Local branch predictors add one more level of hardware which is known as History Table in addition to n-bit saturation counter based pattern table. Saturation counter can be replaced by various state machines which are described into YEH 1992 paper.

7. References

[1] “Alternative Implementations of Two-Level Adaptive Branch Prediction” by Tse-Yu Yeh and Yale N. Patt

[2] “Combining Branch Predictors” by Scott McFarling

[3] “The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference” by Sprangle, et al

[4] “Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction” by Toni Juan et al

[5] “Neural Methods for Dynamic Branch Predictor” by Daniel A. Jimenez et al.

[6] Maria-Dana Tarlescu, Kevin B. Theobald, and Guang R. Gao. Elastic History Buffer: A Low-Cost Method to Improve Branch Prediction Accuracy. ICCD, October 1996.