Hybrid Branch Predictor Using Perceptrons and Separated Weight Tables

Rahmaan Lodhia Brian Roney
School of Electrical and Computer Engineering
Georgia Institute of Technology
rahmaanl@gatech.edu

Abstract
This paper presents a new branch predictor method that combines two conditional branch predictors to increase prediction accuracy and minimize hardware usage to allow the allocation of a 32 KB indirect predictor using a BTB. This new branch predictor is a hybrid predictor that combines an improved perceptron predictor proposed by Jimènez et al [1] and the perceptron branch predictor with separated weight taken/not-take weight tables (SWP) proposed by Shi et al [2]. A meta-predictor is used to distinguish which of the two predictors to use. The total allocation of the hybrid predictor is less than 32 KB, and it results in an improved prediction rate in comparison to a 64 KB gshare predictor.

This paper describes the methodology behind this predictor. It further shows how to allocate the space for this hardware in a 64.25 KB limit. Finally, it presents the predictors performance in comparison to a 64 KB gshare predictor and a 64 KB indirect predictor.

1. Introduction

Improving the overall branch prediction in modern computers is one key technique in getting the most performance out of modern machines. It is vital to have a strong predictor inside a machine to reduce stalls and latency issue when dealing with branches. Jimènez et al introduced the idea of using the simple neural network perceptron method to predict branches [2]. Shi et al further took the idea of using a perceptron for branch prediction and created the SWP predictor. This paper examines the combination of these two predictors and how they allow for an efficient use of hardware in computing predictions. This reduction in hardware allows the inclusion of a simple indirect predictor to create an overall system that allows the minimization of both conditional and indirect branch miss prediction rates.

This paper first covers the methodology behind this hybrid predictor. It will discuss how the original perceptron method introduced by Jimènez et al [1] was modified to allow for a reduction in aliasing among weights and optimizing its results. Then the paper will discuss how the SWP functions, and how using a small version of it as side predictor allows us to improve upon the overall prediction rate of our method. The paper will then discuss the hardware costs related to the method. Finally, the paper will demonstrate the results of this method in comparison to the popular *gshare* predictor when using several trace tests from the Championship Branch Prediction (CBP) contest.

2. Methodology

2.1. Improved Perceptron Predictor

The foundation of this hybrid predictor is the perceptron predictor first introduced by Jimènez et al. Jimènez et al presented the perceptron as a new way of predicting conditional branching using a simple neural network. Perceptrons were a natural choice for branch prediction due to their simplicity and effective use of hardware space. The perceptron method is simple. The neural network trains weights based on input vectors it receives. The weights then are applied on more incoming vectors, and the resulting dot product is compared against zero and a threshold to determine which class of data the given vector is in and if further training is required. To apply this to branch prediction is straightforward, and the implementation is shown in the block diagram in Figure 1. The input vector given is the global history register (GHR), with a bias bit of one for taken, of the branches taken or not taken so far. The current branch address is then taken and hashed into an entry in a perceptron table, where each entry contains a vector of weight counters corresponding to the length of the GHR. The dot product of the weights and the GHR are then calculated, and the output is compared against zero. If the output is greater than the zero, the result is that the branch is predicted taken. If the result is less than zero, the branch is predicted not taken. The output is then compared against a threshold to determine if training of the current entry in the perceptron table is required. The process then continues for any conditional branch encountered by the machine [1].

To improve on this initial design, the predictor used the optimization techniques provided by Jimènez et al. The authors of the original method found that the optimal history length was 62 entries. Furthermore, they arrived at an equation to calculate the optimal threshold value θ for a given history h, $\theta = 1.93h + 14$, which was used in our final predictor [1].

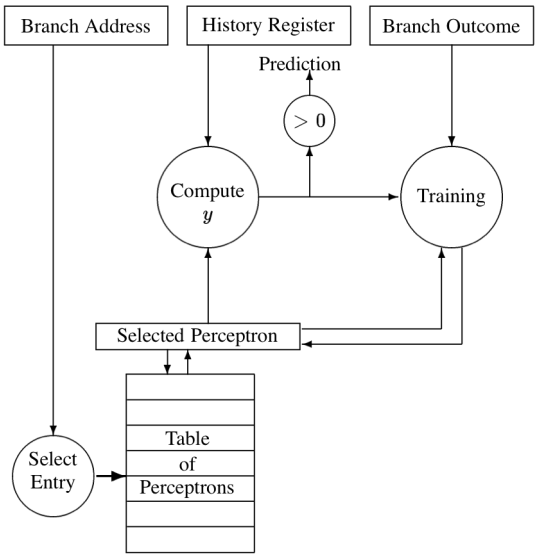

Figure 1. Perceptron predictor block diagram detailing how the prediction is made using the perceptron weights [1].

To further improve the perceptron predictor, the hash function was modified to take into account both the branch address and the path history of branches encountered so far in the program, an idea proposed by Jimènez in another paper [3]. This allows for less aliasing in the weight table as the weights resulted in a more uniform distribution across all the entries. The second improvement was the reduction in the size of the weight counters from 8-bits to 6-bits. This reduced hardware space significantly, and only lowered accuracy by a very negligible amount. With these modifications, the resulting perceptron predictor performs well and only requires 23 KB of space.

2.2. Separated Weights Perceptron Predictor

In the 2011 CBP, Shi et al introduced an improvement on the original perceptron predictor called the SWP. The SWP was unique in that it used the perceptron strategy present in the original method, but it takes the weight tables and separates them into taken and not taken tables. This allows for the elimination of a bias bit that is required in the perceptron method. This separation also allows the predictor to deal better with linearly inseparable histories, which is an issue for the perceptron method. The implementation of the method is described in Figure 2. The branch address given by the PC is hashed using the path history in the same manner as our improved perceptron method. The resulting index is then accessed for the separated weight tables. Depending on the current GHR input, one of the two tables is chosen for calculation of the output. Shi et al found that the separate weight tables worked best only dealing with the first 20 entries of the GHR, leaving the rest to be used with a regular perceptron table. This is due to the fact that the separated weight table is very hardware intensive with longer histories. As displayed in Figure 2, following the separated weights is a regular weight table that is used for calculating the output for the rest of the branch history [2].

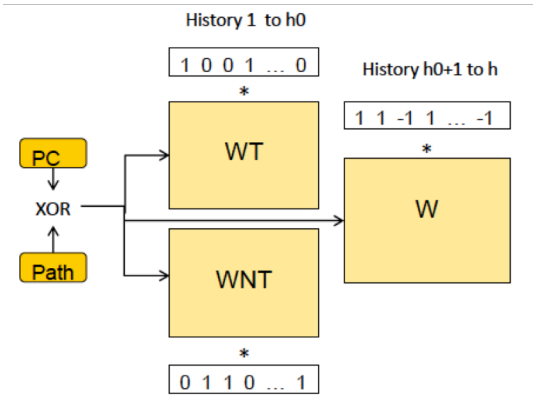

Figure 2. SWP predictor block diagram detailing how the use of separated perceptron tables are used to make predictions [2].

In applying this method, we decided to separate the predictor into three table sections as proposed by Shi et al. The first 20 entries of the GHR are calculated using the separated weight table. The next 16 entries are calculated using a perceptron weight table with the same amount of entries as the each of the separated weight table. Finally, to help reduce hardware cost, a third table of weights is used on the last 29 entries of our GHR using a table with an entry size lowered by a factor of two compared to the previous two table sections. The calculation of each output was very similar to the original perceptron, as the use of the dot product remained the same. Due to the nature of this predictor, the threshold had to be empirically calculated, and found for this setup to be 107. Finally, the last design choice was to make the weight counters for all the predictors 7-bits versus the original perceptron's 8-bits to reduce this predictor's total cost to 7 KB and help ensure accuracy [2].

2.3. Hybrid Predictor

The hybrid predictor was constructed using the two previously discussed predictors. To make use of both predictors, a third predictor was used to determine which of the two prediction outputs to use at runtime. The construction of this predictor simply involved looking at the past 10 predictions of each method. A 4-bit counter was used for each to calculate which

method had the most correct prediction for the past 10 branches. The one with the highest count would be decided at runtime to supply the branch prediction. In case of equal counts or the same prediction outputs, the program will choose the improved perceptron method's prediction over the SWP. This design choice was due to the fact that the original perceptron was less hardware intensive than the SWP, so we could allocate more space for it, giving a stronger prediction.

2.4. Indirect Predictor

The overall cost for our conditional predictor was approximately 32 KB, with 23 KB for the original perceptron and 7 KB for the SWP. The remaining 32 KB was devoted for implementing an indirect predictor. The indirect predictor was implemented using a simple 8192 entry BTB table, where each entry is indexed using the instruction address. The predictor was chosen, because compared to its 64 KB counterpart, it showed a small decrease in performance as a result of its smaller size. This observation led us to choose this method due to its simplicity and efficiency.

3. Hardware Costs

The total hardware cost for this predictor is summarized in Table 1. The table details which components are needed in this implementation and the bit costs associated with them. Overall, the proposed predictor falls within the budget of 64.25 KB with a value of 62.77 KB. The conditional predictor using the hybrid predictor falls below 32 KB in costs, while the indirect predictor used in this hardware is exactly 32 KB in cost.

Table 1. Hardware Storage Budget Calculation

Component	Size (Bits)
Perceptron Table	512*6*62 = 190,464
Separated Weight Table	128*2*7*16 = 28,672
Weight Table 1	128*7*20 = 17,920
Weight Table 2	64*7*29 = 11,136
GHR	1*128 = 128
SGHR	2*128 = 256
STrain	2*128 =256
out1 & out2	32*2 = 64
GA	10*128 = 1,280
Speculation Counter	8
Prediction Counters	2*4 = 8
Indirect BTB	8192*32 = 262,144
Total Bits	514,192
Total in Bytes	64,274 = 62.77 KB

4. Performance

Our predictor was tested using all of the traces used in the 2011 CBP. The results of these tests are shown below. Figure 3 shows how our 32 KB hybrid conditional predictor performed in comparison to a 64 KB *gshare* predictor in terms of MPKI. As can be observed, our predictor performed better in each type of trace. The overall average MPKI of 5.698% for our predictor was lower in comparison to the overall average MPKI of 6.865% for *gshare*. Figure 4 shows the results of our 32 KB indirect predictor compared with its 64 KB. As noted before, this predictor was expected not to perform better than its counterpart. However, the difference between their final averages was approximately 0.1%, which is negligible considering the amount of hardware saved using the smaller predictor. Table 2 summarizes the data collected for these trace runs, further detailing our performance.

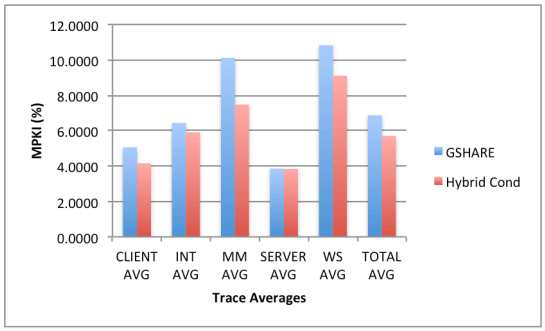

Figure 3. Plot of arithmetic means of MPKI over several trace tests comparing conditional branch prediction performance between the hybrid predictor and *gshare*.

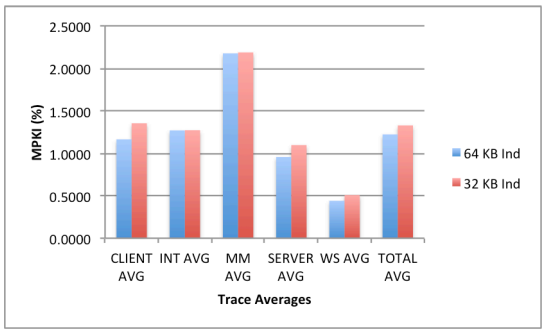

Figure 4. Plot of arithmetic means of MPKI over several trace tests comparing indirect branch prediction performance between a 64 KB indirect predictor and a 32 KB indirect predictor using a BTB.

Table 2. Summary of Average MPKI Rates for All Trace Test for the Branch Predictors

Traces	GSHARE	64 KB Ind	Hybrid Cond	32 KB Ind
CLIENT AVG	5.059	1.164	4.150	1.353
INT AVG	6.433	1.269	5.908	1.271
MM AVG	10.117	2.176	7.471	2.187
SERVER AVG	3.856	0.956	3.839	1.096
WS AVG	10.826	0.440	9.101	0.508
TOTAL AVG	6.865	1.222	5.698	1.328

5. Conclusions

We proposed a hybrid predictor combining an improved perceptron predictor and a SWP predictor. This design allows for an effective 32 KB predictor to be used, allowing for another 32 KB of the proposed budget to be dedicated to a simple indirect predictor. Our final build was 62.77 KB total, which is lower than our maximum budget of 64.25 KB. Experiments using the traces provided by the 2011 CBP show that our predictor performed favorable in comparison to a 64 KB *ghsare* predictor and a 64 KB indirect predictor. The advantages of using these two types of conditional predictors include efficient use of hardware space and the ability to properly deal with long histories, whether they are linearly separable or not.

References

[1] D. A. Jimènez and C. Lin, "Dynamic branch predictions with perceptrons," emph7th International Symposium on High-Performance Computer Architecture (HPCA) pp. 197-206, 2001.

[2] G. Shi and M. Lipasti, "Perceptron Branch Prediction with Separated Taken/Not-Taken Weight Tables," in Proceedings of the 2nd JILP Workshop on Computer Architecture Competitions: Championship Branch Prediction (JWAC-2), San Jose, CA, June 2011.

[3] D. A. Jimènez, "Piecewise linear branch prediction," Computer Architecture, 2005, ISCA '05, Proceedings. 32nd International Symposium on , vol., no., pp.382,393, 4-8 June 2005.