Perceptron Branch Prediction with Separated Taken/Not-Taken Weight Tables

Guangyu Shi
Department of ECE,
UW-Madison
gshi2@wisc.edu

Mikko Lipasti
Department of ECE,
UW-Madison
mikko@engr.wisc.edu

Abstract

Perceptron predictor is well known as its ability to exploit long history information. A limitation of perceptron predictor is that it cannot effectively predict linearly inseparable correlations. In this paper, we present a revised perceptron predictor with separated Taken/Not-Taken weight tables (SWP). It can treat the taken/not taken correlation separately and therefore predict some linearly inseparable branches effectively. Compared to the existing perceptron branch predictors, it requires less number of bits in each weight to achieve the same accuracy, and thus less computational latency of making a prediction.

1 Introduction

In modern superscalar computers, average branch instruction latency is crucial to the overall performance of the computer. Branch misprediction could result in a significant performance lost. Therefore, accurate branch predictor is desired for further improvement on instruction-level parallelism. Among all the branch predictors that have been proposed, perceptron-based branch predictors [1] [2] [3] tend to have a better performance than traditional branch predictors, especially its scalability on long branch history. Despite their reported high accuracies, however, there are two disadvantages of perceptron branch predictors. First, perceptron branch predictors cannot predict linearly inseparable branches. Although some perceptron-based branch predictor, such as piecewise linear branch predictor [3] can efficiently predict the behavior of certain linearly inseparable branches, it has to rely on execution path information to distinguish the predictions. Second, to make the prediction, perceptron branch predictor needs to compute the summation of the 7 or 8-bit weights, which results in large computational latency that makes the predictor difficult to implement.

In this paper, we propose a new perceptron branch predictor, the Separated Weight Predictor (SWP), which separates the correlation of the predicting branch with Taken histories from the correlation with Not-Taken histories. Predictor with separated weight tables attacks both of the two disadvantages of perceptron predictor: it separates correlations with Taken/Not-Taken history, and as a result, it requires less number of bits in each weight value, which reduces the computational latency.

This paper is organized as follows. Section 2 describes the related work on perceptron-based branch predictors. Section 3 introduces our new algorithms for computing the prediction and updating SWP as well as some simulation result on CBP-3 framework. Section 4 describes the customized SWP for CBP-3. Finally, section 5 concludes the paper.

2 Related Work

Jiménez et al proposed the original perceptron branch predictor in 2001 [1]. It achieves higher accuracy on SPEC 2000 integer benchmarks than Gshare and Bi-Mode predictors under the same hardware budget ($\geq$2KB). However, the accuracy of perceptron predictor is worse than Gshare when the linearly inseparable branches are 50% or more of the total number of branches. Meanwhile, huge prediction latency of perceptron predictor makes it infeasible for practical hardware implementation.

To overcome the latency problem, Jiménez proposed Fast Path-based Neural Branch Predictor [2], which uses ahead-pipelining to partially compute the summation of weights before the branch to be predicted is fetched. Ahead-pipelining is also used in Piecewise Linear Branch Predictor [3], which effectively uses path history as well as current branch address to access weight tables. The practical piecewise linear branch predictor also uses ahead-pipelining to mitigate latency problem. Ninomiya et al proposed A$\hat{3}$PBP [5] that uses local history to avoid destructive aliasing. In 2008, Amant et al proposed Scaled Analog Neural Predictor (SNAP) [4], which removes the ahead-pipeline by using analog circuits to calculate the summation of the weights.

3 Prediction with Separated Weight Tables

In this section, we will first describe a limitation of original perceptron branch predictors. Then we will introduce the prediction and update algorithm of SWP, and

	History	Prediction
1	Taken	Taken
2	Taken	Not-Taken
3	Not-Taken	Taken
4	Not-Taken	Not-Taken

Table 1: 4 types of correlations between past history branches and the branch to be predicted

Program 1 A code example

```
// x is an unknown value
If (x>=1000)   // Branch A
{ /* do some task */ }
If (x>= 500)  // Branch B
{ /* do some other task */}
```

describe how the problem of traditional perceptron predictor can be overcome by SWP.

3.1 Intuition

Table 1 shows four possible correlations between the branch to be predicted and the history leading to it. Note that weights perceptron branch predictors are trained for either positive correlations or negative correlations. That is, one can choose to strengthen correlations 1 and 4, or to strengthen correlations 2 and 3. However, for some applications, it is desired to have strong correlation 1 but weak correlation 4, or strong correlation 2 but weak correlation 3, and vice versa. A code example is given in program 1.

In program 1, Branch A serves as a history of younger branch B. It is not difficult to observe that, if branch A is taken, then branch B will also be taken. However, if A is not taken, then we do not know if B will be taken or not. This is a simple example of branch patterns that has a strong correlation 1, but weak correlation 4 in table 1. In perceptron branch predictors, if the branch A is taken at certain frequency, it is likely that the weight associated with this history will be trained toward a positive value. When branch A is not taken, branch B will be indicated as Not-Taken, although in fact there is no strong correlation between them.

Although piecewise linear branch predictor can predict certain types of linearly inseparable branches, the prediction relies on the different execution path information. If the same execution path leads to different branch outcomes (similar as the example in figure 1), piecewise linear predictor cannot distinguish the predictions. Therefore, it is necessary to have different weight tables for taken and not-taken histories.

3.2 Prediction and Update Algorithm

Program 2 and 3 presents the pseudo-code of the prediction and update algorithm of SWP. In this code, WT and WNT are the Taken and Not-Taken weight tables, respectively. GHR is the global history register. HA is the path history address register that stores the addresses of the past executed branches. Integer *ghl* is the length of the global history. Integer *address* is the address of the branch to be predicted.

Program 2 Prediction Algorithm

```
function predict: boolean
begin
  sum := W0[address];
  for i in 1 to ghl do
    index := hash (address, HA[i]);
    if GHR[i] = true then
      sum := sum + WT[index, i];
    else
      sum := sum + WNT[index, i];
  end for
  predict := (sum>=0);
end
```

Program 3 Update Algorithm

```
function update
begin
  if |sum|<threshold or predict != br_taken
    for i in 1..ghl do
      index := hash (address, HA[i]);
      if GHR[i] = true && br_taken = true
        WT[index,i] := WT[index,i] +1;
      else if GHR[i]=true && br_taken=false
        WT[index,i] := WT[index,i] -1;
      else if GHR[i] = false && br_taken=true
        WT[index,i] := WNT[index,i] +1;
      else if GHR[i]=false && br_taken=false
        WT[index,i] := WNT[index,i] -1;
      end if
    end for
  end if
end
```

Figure 1 further illustrates the prediction and update algorithm. Now weights in traditional perceptron predictors become weight pairs: Taken and Not-Taken weights. To calculate the summation, only one of the weights in each pair will be chosen. In figure 3, colored squares are weights selected. Similarly, when updating the predictor, only these selected weights will be updated, while other weights remain unchanged. Only addition is performed during prediction, although both incrementing and decrementing operations are performed during the update. These algorithms allow the predictor to treat the correlations separately.

Figure 2 shows the simulation result on 12 traces out of the 40 traces from CBP-3. We compare SWP with piecewise linear branch predictor [3], both of which have the same history length. We can observe that SWP outperform piecewise linear branch prediction, especially when weights are small.

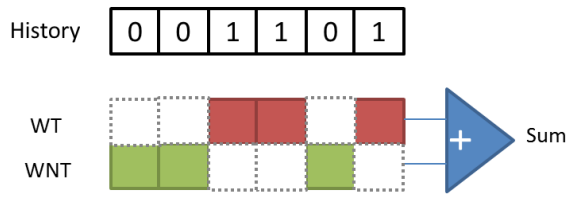

Figure 1: An example of weight selection by SWP

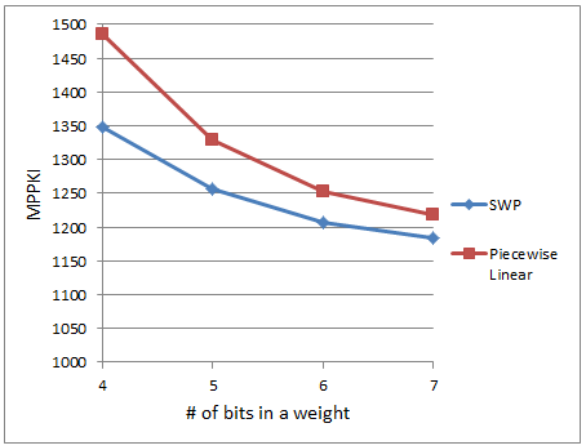

Figure 2: Number of bits in a weight vs. MPPKI

3.3 Implementation of SWP

Figure 3 shows the structure of SWP. Other than the separated weight tables, its implementation is no different with previous proposed perceptron predictors. A number of multiplexors are used in the weight table to select the desired weight from Taken and Not-Taken weight tables. This weight selection is performed in parallel and thus will not increase latency of the prediction.

Another advantage of SWP is that no 2's complement of weight needs to be calculated. In perceptron branch predictor, if the history is not-taken, the negation of corresponding weight needs to be calculated. In practical implementation, this is done by flipping all the bits in that weight. In SWP, since there is not subtract operation needed, we do not need to calculate the negation of a weight, which further improves both latency and accuracy.

3.4 Partially Separated Weight Tables

It is not difficult to realize that for the same length of history and the same number of entries in the weight table, SWP requires twice as much storage space as a regular perceptron predictor. [4] states that recent branches have stronger correlations with the branch to be predicted than old branches. Therefore, we use partially separated weight table in our predictor.

Figure 4 is the structure of SWP with partially separated weight tables and figure 5 is a set of simulation result with partially separated weight table. The horizontal axis shows the history length associated with separated weight tables (h0 in figure 4). The total length of the history is 64. We observe that when the most recent 20 branches are associated with separated weight tables, the MPPKI is reduced by 2.0%. When we further increase h0 to 64 (fully separated weight tables), the MPPKI is further reduced by only 0.9%. Therefore, It is most efficient to use separated weight tables on only a few most recent branches, and use a single weight table to explore long history information.

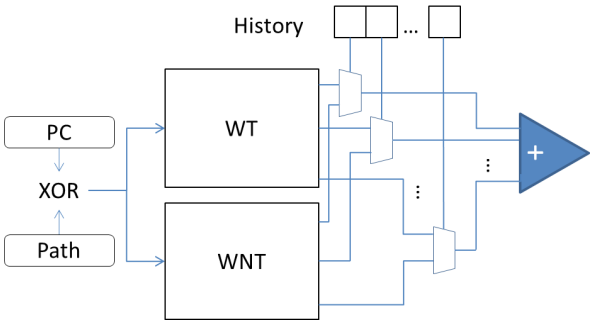

Figure 3: Structure of SWP

3.5 Combined with Other Optimization Schemes

Most of the optimization schemes for perceptron predictors can be applied on top of SWP. In particular, we use the proposed method in piecewise linear branch prediction design [3] that uses path information to reduce aliasing in weight table. We also used dynamic threshold [6] to set training threshold adaptively. Result shows that these optimization schemes further improves the branch prediction accuracy.

Traditional perceptron predictors are usually associated with a bias table, and the bias weights often have a larger correlation coefficient than other weights. Bias weight mitigates the inability of perceptron predictors to some extent. In SWP, however, bias weight is not needed since we separate T/NT correlations. Simulation result shows that removal of bias weights in fact improves the accuracy of the predictor.

4 Final Configuration for CBP-3

We use partially separated weight tables as well as weight tables with different width to optimize the storage space. We also use global history registers and a speculation counter to keep track of each branch in the pipeline. Note that since the number of branches in the pipeline can be larger than the depth of the weight tables, the global history registers (128) is deeper than weight tables (65). Table 2 summarize the parameters we choose for the predictor. Table 3 calculates the total storage budget of our submission. The total number of bits is kept under the storage budget of CBP-3 (64KB+1KB).

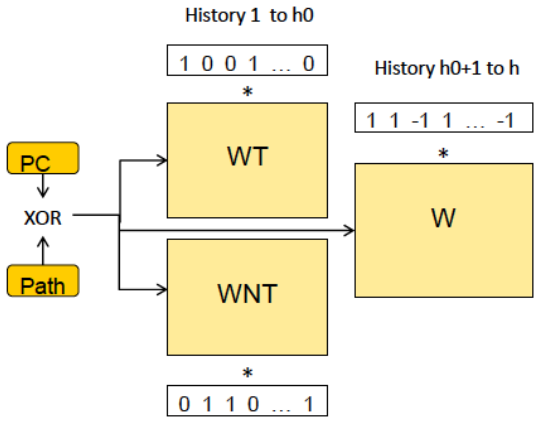

Figure 4: Structure of SWP with partially separated weight tables

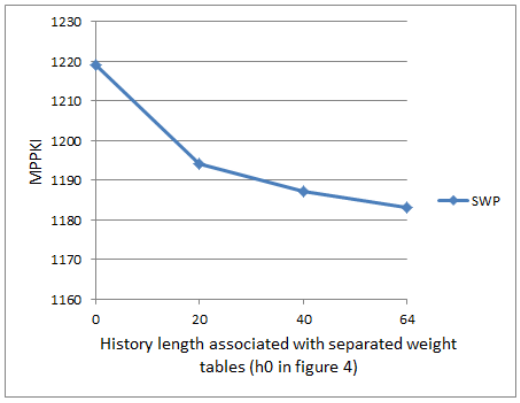

Figure 5: History length associated with separated weight tables (h0) vs. MPPKI

5 Conclusion

We proposed perceptron branch prediction with separated weight tables. In our predictor, Taken and Not-Taken correlation of the same history is treated separately using two different weight tables. Experiment result shows that our proposed branch predictor realizes high prediction accuracy. Its advantage on predicting some linearly inseparable branches allows it to use less number of bits in the weights and a shorter history length, which can reduce the latency of prediction. Further more, using separated weight table on most recent branches and using single perceptron on the rest branches is proved to be a good strategy to improve prediction accuracy with minimum storage overhead.

Parameter	Value
Depth of 1024-entry separated weight tables	20
Depth of 1024-entry single weight table	16
Depth of 512-entry single weight table	29
Total History length for weight tables	65
History length for global registers	128
number of bits in a weigh	7
Training threshold	107

Table 2: Selected parameters of SWP

Component	Size (bits)
WeightT/NT	1024*2*7*20 = 286,720
Weight1	1024*7*16 = 114,688
Weight2	512*7*29 = 103,936
GHR	1 * 128 = 128
SGHR	1 * 128 = 128
HTrain	1 * 128 = 128
accum = 32	
HA	10 * 128 = 1,280
Weight Table Indices	10 * 65 = 650
Speculation counter	8
TC, threshold	7+8 = 15
Total	507,713 < 532,480

Table 3: Storage budget calculation

References

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

[2] D. A. Jiménez, "Fast path-based neural branch prediction", *36th International Symposium on Microarchitecture (MICRO)* pp. 243-252, 2003 1

[3] D. A. Jiménez, "Piecewise linear branch prediction", *32nd International Symposium on Computer Architecture (ISCA)*, pp 382 - 393, 2005 1, 3

[4] R. Amant, D. A. Jiménez and D. Burger, "Low-power, high-performance analog neural branch prediction", *41st International Symposium on Microarchitecture (MICRO)*, pp. 447 - 458, 2008 1, 3

[5] Y. Ninomiya and K. Abe, "Path traced perceptron branch predictor using local history for weight selection", *Second JILP Championship Branch Prediction Competition (CBP-2)*, pp. 7 - 12, 2007 1

[6] A. Seznec Analysis of the o-geometric history length branch predictor, in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 394 405, 2005 3