Hybrid Learning-Based Branch Predictor

Parshvi Z. Shah
Department of Electronics Engineering
Fr.C.R.C.E
Mumbai, India

Sapna U. Prabhu
Department of Electronics Engineering
Fr.C.R.C.E
Mumbai, India

Abstract— Accurate branch prediction mechanisms contributes to the optimization of processor performance. As the number of pipelines stages in modern processors are steadily increasing, the importance of a good branch prediction logic is rising.

Traditional branch prediction relies on history of branch for prediction. Several researchers have worked on branch prediction accuracy. Modern branch predictors rely heavily on longer global history to produce accurate branch prediction. The entire length of the program is however, not related with recently executed branches. For these parts of the program, the extra information encoded in the global history does more harm than good. This limitation gave us motivation for looking beyond using larger history length. In the Hybrid branch predictor, branch prediction accuracy can be improved by basing prediction on the outcome of neighboring branches (global behavior) and outcome of the branch itself (local behavior).

This paper deals with the implementation of the Hybrid branch predictor with the help of neural networks, which provides higher predictive capabilities than commonly used global branch predictors. The hardware resources required for Learning based predictor scale linearly with the history length, in contrast with other purely dynamic schemes that need exponential memory that permits our predictor to consider longer branch histories.

Keywords—Branch Predictor, Alloyed predictor, Global History, Perceptron , Path History.

I. INTRODUCTION

The techniques of instruction level parallelism and pipelining have been used well to speed up the execution of instructions. High performance and high-accuracy designs call for deeper pipelines, which lead to costly misprediction penalties. The conditional branches are the critical factor to the effectiveness of a deep pipeline. The question of how to predict the direction of conditional branches has received intense study in recent years. Branch prediction has been a very important method to improve processor performance which tries to solve control dependency issues exposed in program instructions and improve instruction level parallelism, thus enabling deeper processor pipeline for better performance results.

Branch prediction techniques can be classified into two groups: static or dynamic. Static Branch Prediction are simple compile-time schemes in which predictions are Static. They are fixed for each branch during the entire execution, and the predictions are compiling time guesses. In Dynamic Branch Prediction, the hardware influences the prediction while execution proceeds. Prediction is decided on the computation history of the program.

Many researchers have been focusing on branch direction algorithms and recent improvement on the predictive accuracy in literatures is achieved by increasing the complexity of algorithms.

Traditional branch predictor keeps a separate history buffer for each conditional branch. As correlation between the branches are important, global history buffer have been used. It keeps a shared history of all conditional branches. The advantage of global history is that any correlation between different conditional branches is part of making a prediction. Generally speaking, dynamic branch prediction gives better results than static one at the cost of increased hardware complexity, and has caught a lot of attention in the past few years. There are many proposed algorithms for dynamic branch predictors, such as bimodal, two-level and hybrid which are extensively implemented in to predict the branch directions. Bimodal prediction uses a table of 2-bit saturating counters indexed by branch addresses to predict the most common direction. Two-level prediction employs two levels of information such as branch history records and pattern history table for the direction prediction. Hybrid prediction takes advantage of combined strategies for the same purpose.

As the length of the application program is steadily increasing, longer global history length needs to be used to provide better prediction accuracy. Several researchers have been reported that larger history length branch prediction gives higher prediction accuracy.

A neural branch prediction which was first proposed in 1999 and is very promising now a day's uses more complex algorithms to weight for the possible paths in order to select a definite way for branch direction. Neural branch predictor using perceptron, one of the simplest possible neural networks. Perceptrons are easy to understand, simple to implement, and have several attractive properties that differentiate them from more complex neural networks. Perceptron based branch predictor has shown promise in attaining high prediction accuracy due to linear resource growth compare to the exponential resource growth of traditional predictor.

Larger history length branch predictor using neural network, such as FPB,PLB,TAGE and O-GEHL gives better prediction accuracy than traditional counter parts such as G-Share.[1][2][3][4][5].

The objective of the branch predictor is to reduce the probability of making an incorrect decision, to avoid fetching instructions that eventually ought to be discarded. Branch predictors plays an important role in achieving highly effective performance in several modern pipelined microprocessors.

II. RELATED WORK

Several researchers have worked on improving on branch prediction accuracy.[5][1][9][3][4][7][10][11].

Yeh and Patt [1] introduced the global history register (GHR) in their two level predictor; to keep outcomes of the previously executed branch.GHR provides the correlation of current branch with previously executed branches. The advantages of GHR further used in G-share/G-select predictor. Traditional predictors were using 10-15 bits for GHR which provides short distance correlations. Generally, longer branch history enables branch correlation with more distant branches, so recent predictors tend to adopt long history at the cost of complicated hardware.

To boost the accuracy of branch predictor several researchers have worked on combining global history and local history of branch Zhi et al. introduced alloyed predictor, a new hardware-based two-level branch predictor that concatenate global and local history with some bits of program counter [9] .

Jimenez et al. [3, 4] introduced neural based perceptron predictor, which uses global history to train a neural network. A perceptron of weights which depends on the length of the global history register. When a branch arrives, predictor will input the has a number branch address to the hash function to generate the index to the perceptron weight table to pick a perceptron. The weights of the selected perceptron are then used with the global branch histories to compute the dot product and accumulate results .The branch prediction is determined by the value of summation, if the value is positive, then the branch is predicted Taken, otherwise Not Taken,. Finally the final branch outcomes will update the perceptron weights. If the branch outcome agrees with the prediction, then weights will be incremented, otherwise decremented accordingly.

Path history should provide better correlation information than pattern history, because path history is a superset of pattern history. Path information includes the branches by which the current branch was reached, not just the pattern of directions that they went to reach the current branch. Jimenez et al [5] introduced Fast Path-Based (FPB) branch predictor is an improved neural perceptron predictor. this predictor improves accuracy by combining path and pattern history to overcome limitations inherent to previous neural based predictors. It provides lower latency than previous neural predictors.

Seznec [7] introduced O-GEHL predictor using geometric series principle. up to get the prediction result positive to predict Taken and vice versa. Seznec [8] introduced TAGE predictor combines a traditional predictor such as gshare table, T0, which is default predictor, with some tables of variable history length for branch forecast. Each table has independent hash function using different historical length; the tag field of each selected table entry will be matched against the tag field of incoming hashed index. If there is a match in a higher table, the resulting pred field will be selected and override results from lower tables .If there is no match among higher tables, the prediction from the default predictor will be used.

Daniel [10] introduced scaling based neural branch predictor based on the principle of inverse linearity curve because each branch doesn't contribute equally; unsurprisingly, more recent weights tends to have a stronger correlation with branch outcomes. Each weight of the perceptron table scale with correlation coefficients which can determine from the inverse linear curve. The correlation coefficients were generated using the publicly distributed traces for the CBP competition. By multiplying weights with coefficients proportional to their correlation, the predictor achieves higher accuracy.

Daniel [11] introduced optimized version of scaling based neural network ,hybridized with two simple two-level adaptive predictors .The predictor achieves higher accuracy by using some tricks such as value of threshold and coefficient vectors are selected empirically during run time of program.

III. PROPOSED ALGORITHM

Learning based branch predictor can be implemented using CBP framework. The following variables are used by the algorithm:

W: A two-dimensional variable array of integers weights for global prediction.7-bits are used for weights saturate at +63 and - 64.
h: The global history length. This is an integer, variable in the implementation.
Lh: The local history length. This is an integer, 17 in the implementation.
LW: A two-dimensional array of integers weights for local prediction.7-bits are used for weights saturate at +63 and -64.This array has 96 entries.
A: An array of addresses. As branches are executed, their addresses are shifted in to the first position of this array. In the implementation, the elements of the array are the lower
9 bits of the branch address.
C: An array of scaling coefficient. The Coefficients are chosen as c[i]=1/(a+b*i).where a=0.1111 and b=0.037 [10].
sum_global: An integer. This integer is the dot product of a weight vector chosen dynamically from W and the global history register.
sum_local: An integer. This integer is the dot product of a weight vector chosen dynamically from LW and the local history register.

Sum: An integer. This integer is the sum of sum_pos, sum_local and bias weight.

The neural prediction algorithm presented below achieves higher accuracies than previously proposed neural algorithms. Predictors are divided in to two parts one is for global prediction and other is for per-branch prediction i.e. Local prediction. The higher accuracies result from:
1. Accessing the weights using a function of the PC and the path for global prediction and function of PC for local prediction.
2. Breaking the weights into a number of independently accessible tables for global prediction.
3. Scaling the weights by the coefficients as previously described.
4. Taking the dot product of a global branch history vector and the scaled weights to get the value of sum_global.
5. Taking the dot product of local branch history which can access using a function of PC and local weights also can access using a function of PC to get value of sum_local.
6. Finally add the result of local predictor i.e. sum_local, global predictor i.e. sum_global with bias weight to get final result **sum**.
7. Once the actual outcomes of the branch become known, the training algorithm uses this outcome and the value of sum to update the weights in local and global weights table. Training algorithm is same as basic perceptron training algorithm with
Threshold value $\Theta = [1.93h + 14]$,[7] where h is history length.

A. Global History based prediction

Fig 1. shows a block diagram of the global prediction algorithm. The two key parameters of the predictor are h, the global history length, with which the dot product is computed, and R the number of rows in each weights table. Other inputs to the global predictor are A, a vector of the low-order 9 bit of each of the past h branch addresses (A is effectively a path vector).

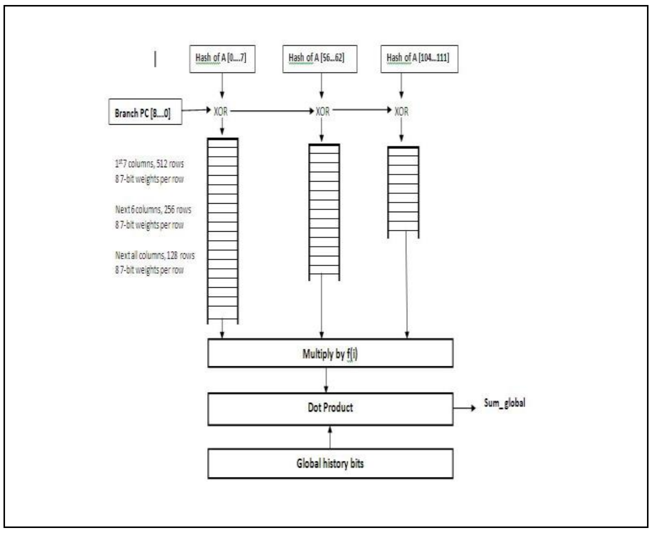

Fig. 1.Global prediction Algorithm

The two components of the dot-product computation are the history vector and the weights vector. The history vector consists of h, which provides history bits of the most recent h branches. The second component of the dot-product computation, the weights vector, is obtained by reading eight weights from each of columns. The first 7 columns, containing the weights for the most recent history bits, have 512 entries than other columns because the most recent weights are the most important. Next six columns have 256 entries. Rest of the columns has 128 entries.

For e.g. in the design if h=128 and r = 512, 256, or 128. The columns are partitioned into 16, rather than just one large indexed row of 128 weights (num_col=history length/block size), because the separation reduces aliasing and achieves higher accuracy with low additional complexity. To index each table, an eight-bit fraction of the A vector is XORed with the low-order eight bits of the branch PC, resulting in an eight-bit index for one of the rows. In the first 7 columns with 512 entries, an extra bit of the branch PC is XORed with a second address bit from the most recent branch to produce a nine-bit index. The bias weight table is indexed with 11 lower-order bits from the branch PC.

B. Per-branch history prediction

Fig 2.Shows a block diagram of the local branch prediction.
To improve the accuracy, combination of global and per-branch history is used rather than just global history as outlined in the algorithm above.

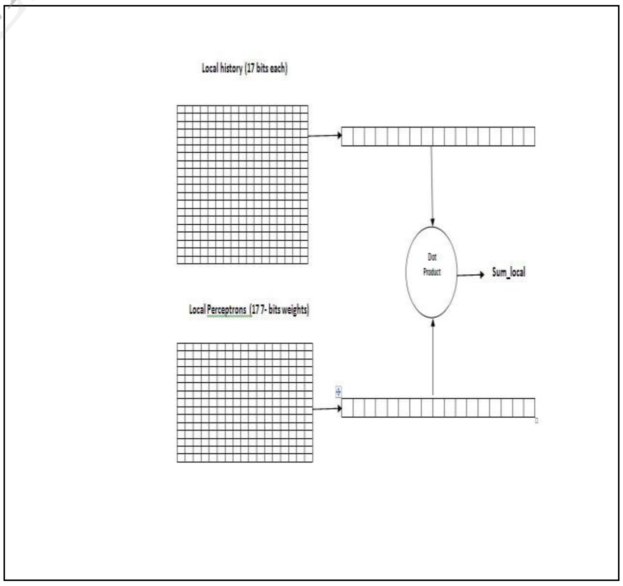
Local history (17 bits each)
Dot Product
Sum_local
Local Perceptrons (17 7- bits weights)

Fig. 2. Local branch prediction

A table of per-branch histories is kept and indexed by branch address modulo numbers of histories. Weights for local
Perceptrons are kept separately from global weights. These histories are incorporated into the computations for the prediction and training in the same way as the global histories.

C. Predictor Update

Updating the predictor consists of three phases, some of which can occur in parallel.

1. Updating histories:

When the outcome of a branch becomes known, it is shifted into H. The lowest-order bit of the branch address is shifted into A.

2. Training the predictor:

If the prediction was incorrect, or if the magnitude of the predictor output was under a set threshold (ϴ = [1.93h + 14]), where h is history length, then the predictor invokes its training algorithm. As in previous neural predictors, the weights responsible for the output are incremented if the corresponding history outcome matches the current branch Outcome, and decremented otherwise. The weights use saturating arithmetic.

3. Updating the training threshold:

An adaptive threshold training algorithm is used to dynamically adjust the threshold at which training will be invoked for a correct prediction.

This algorithm is the same as the one used for O-GEHL [7].The threshold is increased after a certain number of incorrect predictions, and decreased after a certain number of correct predictions whose outputs were not as large as the current threshold. Seznec observed that good accuracy is achieved when the training algorithm is invoked equally many times after correct and incorrect predictions [9]; this threshold training strategy strives to achieve that balance. This training requires single saturated Threshold counter.(TC).

```
if ((P!=Out)  {TC= TC + 1; if (θ is saturated positive)
{θ=θ+1; TC=0 ;}}
if ((P == Out) & (|s|<θ)) { TC= TC - 1; if (θ is saturated
negative)
{ θ=θ-1; TC=0 ;}}
```

IV. RESULTS

This section presents the experimental results of the Hybrid learning-based predictor. The experimental data presented in this report were collected using CBP Traces. The Traces are compiled to execute on the Framework model. The framework models a simple out-of-order core with 256 entry re-order buffer, 3 schedulers, 4 14 stage wide pipeline and 2-level cache. The trace set includes 40 traces, classified into 5 categories: CLIENT, INT (Integer), MM (Multimedia), SERVER and WS (Workstation).

Experiments have been conducted for the following different combinations:

1. without path and without scaling,
2. without path and with scaling
3. without scaling and with path
4. with all

With variation of global history length (128,256 and 512 bits) by running 40 traces of 5 different classes. All results are compared in term of Miss Prediction per kilo instructions (MPKI) where,

MPKI= (mispredicted branches/number of instructions)*1000

A. Analysis of Results

Fig. 3 and Table I. shows the average MPKI for all four combinations, with three different history lengths.

TABLE I. COMPARATIVE ANALYSIS

	HISTORY LENGTH= 128	HISTORY LENGTH= 256	HISTORY LENGTH= 512
With All	4.93	4.79	4.77
Without Scaling And With Path	5.14	5.03	5
Without Path And With Scaling	5.37	5.26	5.26
Without Path And With Scaling	5.72	5.49	5.44

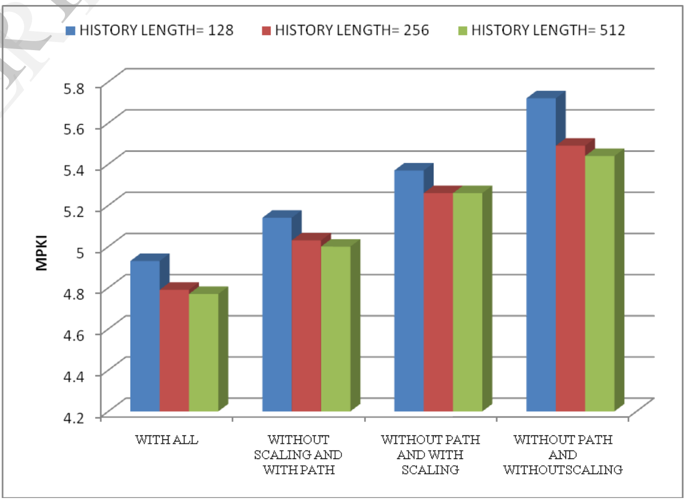

Fig. 3. Comparative Analysis

1. The performance of without path and without scaling with history length 256 is better than with history length 128 and gives improvement of approximately 4.02%.simillarly with history length 512 gives improvement of 0.9% over with history length 256 and 4.89% improvement over history length 128.
2. The performance of without path and with scaling with history length 256 is better than with history length 128 and gives improvement of approximately 2.04%.simillarly with history length 512 doesn't gives any improvement over with history length 256 and 2.04% improvement over history length 128.
3. The performance of without scaling and with path with history length 256 is better than with history length 128 and gives improvement of approximately 2.72%.simillarly with history

length 512 gives improvement of 0.59% over with history length 256 and 2.72 % improvement over history length 128.

4. The performance of with all with history length 256 is better than with history length 128 and offers improvement of approximately 2.83%.simillarly with history length 512 provides improvement of 0.41% over with history length 256 and 3.24 % improvement over history length 128.
5. Figure 3. Shows final output with four combinations and three different history lengths. It shows that combination "with all" gives better improvement irrespective of the history lengths.

B. Size of the predictor

The number of bits going to be used by the predictor has importance since there is always a tradeoff between hardware implementation and accuracy of the predictor.

TABLE II. COMPARATIVE ANALYSIS OF NO. OF BITS USING FOR PREDICTOR

Source of bits	Quantity of bits for HL=128	Quantity of bits for HL=256	Quantity of bits for HL=512
Local history	384x17=6,528	384x17=6,528	384x17=6,528
Local weights	7x96x17=11424	7x96x17=11424	7x96x17=11424
Global weights:			
1st 7 columns	7x(8x7x512)=2,00,704	7x(8x7x512)=2,00,704	7x(8x7x512)=2,00,704
Next 6 columns	6x(8x7x256)=86,016	6x(8x7x256)=86,016	6x(8x7x256)=86,016
Next 3 columns	3x(8x7x128)=21504	19x(8x7x128)=1,36,192	51x(8x7x128)=3,65,568
Global history	128	128	128
path history	9x128=1152	9x256=2304	9x512=4608
Threshold counter	12	12	12
Total No. Of bits	**3,27,468 bits**	**4,43,436 bits**	**6,75,372 bits**

.	No. of bits	MPKI for WITH ALL
HL=128	335872	4.93
HL=256	450560	4.79
HL=512	671744	4.77

TABLE III. COMPARATIVE ANALYSIS OF NO. OF BITS AND MPKI

	MPKI	% of improvement
G-Share	7.75	0.00%
HL=128	4.93	36.39%
HL=256	4.79	38.19%
HL=512	4.77	38.45%

Table II. shows no of bits used by predictor for three different history lengths. With history length of 128 predictor uses approximately 41KB (kilobits).with history length of 256 predictor uses approximately 55KB and with history length of 512 predictor uses approximately 82KB.

Table III. shows the relation of number of bits with MPKI. Total MPKI can reduce by 2.83% using large history length of 256 with approximately 14 KB increasing in Number of bits. Similarly MPKI can reduce by 0.41% using larger history length of 512 with approximately 27KB increasing in number of bits.

Using a larger history length of 512 with increasing bits does not give much benefit. There for it's always trade of between to large hardware and higher accuracy.

C. Comparision with G-share predictor

G-Share predictor is the traditional Global branch predictor. For the purpose of analysis, global history length 18 is used for G-share. The approximate size of the G-share predictor is 64KB.

TABLE IV. ANALYSIS OF MPKI AND PERCENTAGE OF IMPROVEMENT WITH G-SHARE

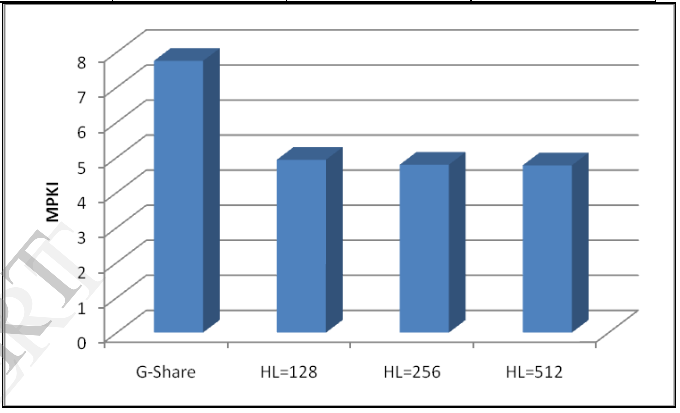

Fig. 4. Analysis of MPKI with G-Share

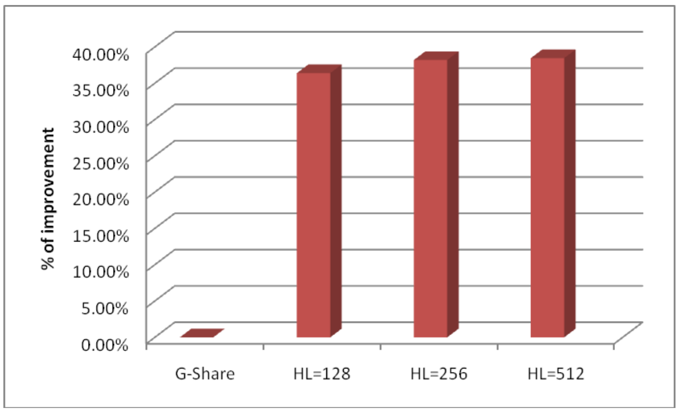

Fig. 5. Analysis of percentage of improvement with G-Share

Table IV. And Fig. 4 and Fig.5 shows comparisons of Hybrid learning- based predictor with traditional G-Share Predictor. It shows that implemented predictor gives improvement of 36.39% when history length is 128, 38.195% when history length is 256 and 38.45% when history length is 512.

Results show that hybrid learning based predictor implemented achieves better performance than the G-share predictor.

V. CONCLUSIONS

Branch predictor continues to evolve and improving branch prediction accuracy is still an open problem. The proposed branch predictor uses Neural learning techniques-the perceptron-as basic mechanism with the variation of global history length. It includes the features like scaling of global history length to reduce noise due to larger history length and path history to provide the correlation information amongst the branches. The proposed branch predictor ensures increase in prediction accuracy at the cost of increased in hardware.

The future scope includes investigations of other techniques liked mixed -signal branch predictors which can be used to reduce mispredictions in a power-efficient manner.

REFERENCES

[1] Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in proc. 24th Annual IEEE/ACM International Symposium, 1991, pp. 51–61
[2] E.R.E.Kessler and D.A.Webb,"The alpha 21264 microprocessor architecture," in Proc.International Conference on Computer Design, 1998, pp. 90–95
[3] D. A. J. Ì. nez and C. Lin, "Dynamic branch prediction with perceptrons," in Proc.7th International Symposium on High Performance Computer Architecture, January 2001, pp. 197–206.
[4] ——, "Neural methods for dynamic branch prediction," in Proc.ACM Transactions on Computer Systems, November 2002.
[5] D. A. J. Ì. nez, "Fast path-based neural branch prediction," in Proc.36th International Symposium on Micro architecture, 2003.
[6] ——, "Piecewise linear branch prediction," in Proc. 32nd Annual International Con ference on Computer Architecture, June 2005, pp. 382–393.
[7] A. Ì. Seznec, "Analysis of the o-geometric history length branch predictor," in Proc 32nd Annual International Conference on Computer Architecture, June 2005, pp.394–405.
[8] ——, "The l-tage predictor," Journal of Instruction Level Parallelism, April 2007.
[9] M. R. S. K. S. Zhijian Lu, John Lach, "Alloyed branch history: Combining global and local branch history for robust performance," International Journal of Parallel Programming, vol. 31, 2003
[10] D. A. J. RenÂt'ee St.Amant and D. Burger, "Low-power, high-performance analog neural branch prediction," in Proc.41th Annual IEEE/ACM International Symposium on Micro architecture,
[11] D. A. J. Ì. nez, "Oh-snap: Optimized hybrid scaled neural analog predictor," Journal of Instruction Level Parallelism, 2011.