Hardware Efficient Piecewise Linear Branch Predictor

Jiajin Tu, Jian Chen, Lizy K.John
Department of Electrical and Computer Engineering
The University of Texas at Austin
tujiajin@mail.utexas.edu, {jchen2,ljohn}@ece.utexas.edu

Abstract- *Piecewise linear branch predictor has been demonstrated to have superior prediction accuracy; however, its huge hardware overhead prevents the predictor from being practical in the VLSI design. This paper presents two novel techniques targeting at reducing the hardware cost of the predictor, i.e., history skewed indexing and stack-based misprediction recovery. The former is designed to reduce the number of ahead-pipelined paths by introducing the history bits in the index of the weight table, while the latter employs stacks instead of arrays of registers to recover predictor states from misprediction. Experimental results show that history skewed indexing helps the predictor improve prediction accuracy by 5.8% at the same harware cost. Moreover, the combination of these techniques can achieve about 30% area reduction with less than 3% IPC loss compared with the original piecewise linear predictor.*

I. INTRODUCTION

Modern high-performance microprocessor design features deep pipelining and wide instruction issue. These widely used techniques have created tremendous impact on the branch predictor from two sides. On one hand, branch predictor is required to produce more accurate prediction results. Although modern aggressive branch predictor like 2Bc-*gskew* [9] is able to achieve an average prediction accuracy of around 96%, the 4% misprediction rate still contributes a significant portion of performance loss considering tens of instruction cycles of misprediction penalty. On the other hand, the predictor has to be fast enough to prevent performance degradation caused by long prediction latency. Prediction overriding provides a viable solution to multi-cycle predictors, however, this method leads to more complicated fonrt-end pipeline design without fully exploiting the potential of the complex predictor.

To enhance prediction accuracy, a new breed of branch predictors based on neural network has been proposed, which are demonstrated to be more accurate than its saturating counter based counter part [1][7]. The perceptron predictor achieves competitive prediction accuracy by learning the behavior of branches via a simple linear neuron [2]. However, it is unable to learn linearly inseparable branches [3]. Moreover, it suffers from high prediction latency [8], and can only be used as an overriding predictor. The path-based neural branch predictor improves the prediction latency by selecting a neuron dynamically along the path to the branch and staggering computation in time [4]. Piecewise linear branch predictor generalizes the these two types of neural branch predictors. It develops a set of linear functions, one for each program path to the branch to be predicted, to separate predicted taken from predicted not taken branches [1]. With ahead-pipeline, the predictor is able to make a prediction in one cycle. Unfortunately, piecewise linear branch predictor is extremely hardware hungry due to the three-dimensional integer array as well as the additional registers and adders required for ahead pipeline. The huge hardware overhead practically limits the width of the address used to index the integer array to a very small number, preventing the predictor from achieving its full potential.

This paper proposes a modified piecewise linear branch predictor, which significantly reduces the hardware cost with negligible performance degradations. The key idea of this method is to use the combination of the branch address and speculative global branch history bits to index the integer array. Since these global history bits are known during prediction, they can filter out unnecessary copies of speculative predictor states in the ahead pipeline. In addition, the hardware cost can be further reduced down by using a circular stack to recover the branch predictor instead of using the non-speculative checkpoint for all the predictor states. Experimental result shows a substantial area reduction with less than 3% IPC degradation compared with the original piecewise linear predictor.

The rest of the paper is organized as follows: Section II makes a brief review of the background about the piecewise linear predictor. Section III presents the algorithm and hardware implementation of the modified piecewise linear branch predictor. Section IV provides the simulation and implementation results. Section V concludes the paper.

II. BACKGROUND

The idealized piecewise linear predictor maintains a three-dimensional memory array of integers (*W*), a global history register (*GHR*) with history length *h*, and an array of previous *h* branch addresses (*GA*). As shown in Figure 1, *n* bits of current branch address activates one bank of memory array, which will be further indexed by *m* bits of previous branch addresses stored in *GA*. Then, the weights fetched from the *W* will dot product with the corresponding bits in *GHR*. The prediction is made according to the dot-product result plus a bias weight representing the overall tendency of the branch.

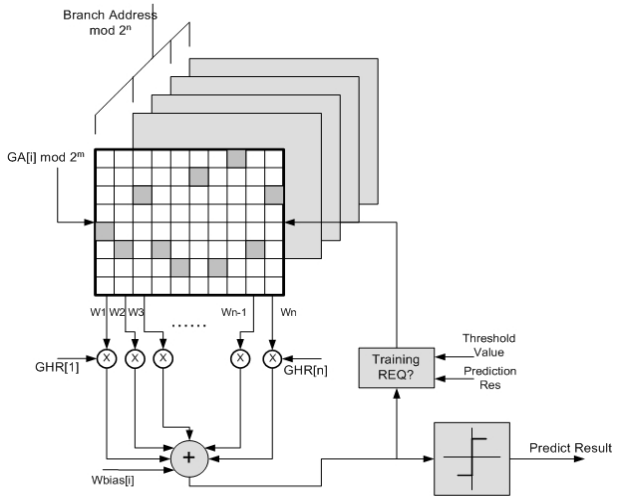

Figure 1.Block diagram of ideal piecewise linear predictor

The ahead-pipelined piecewise linear branch predictor proposed in [1] keeps 2^n copies of the speculative predictor states to drive 2^n possible predictions, where n is the width of the branch address used to index the W array. These speculative states are maintained in SR, which is a $2^n \times h$ two-dimensional shift array of small integers. $SR[i,j]$ holds the speculative partial sum for the jth branch in the future whose address modulo 2^n is i. In order to recover the predictor state from misprediction, an additional two-dimensional shift array R is required. R has the same size and same structure as SR, but it is updated non-speculatively when a branch is resolved.

Although the proposed structure in [1] gives the opportunity for the predictor to make a prediction in one cycle, it suffers from several disadvantages. First, the size of both SR and R is in exponential relationship with n, which practically limits n to be a very small number. This limitation in effect confines the branch predictor in an operational region that its performance is noticeably below what it can achieve in theory. Second, the proposed ahead-pipeline is inefficient in terms of hardware cost and power consumption. The ahead-pipeline has to compute 2^n partial sums in almost every stage of the ahead-pipeline, although only one out of these 2^n computation results is useful. That means a large amount of hardware is dedicated on computing the partial sums for nothing. In addition, the non-speculative R is only useful when there is a branch misprediction. However, it needs to be updated every time a branch instruction is resolved, wasting power for most of the time.

III. Modified Piecewise Linear Branch Predictor

This section presents the modified piecewise linear branch predictor, which overcomes the afore-mentioned disadvantages of the original one in [1]. Two major techniques, i.e., history skewed indexing and recovering state with stack, are employed to make the predictor more hardware and power efficient.

A. History Skewed Indexing

The idea of history skewed indexing is to use the combination of branch address and history pattern to index into W array, as oppose to using branch address only in the original piecewise linear predictor. The mapping of index after combination is as follows:

$$index = (b\_addr << his\_bit) + (history \& (2^{his\_bit}-1)) \bmod 2^n$$

where *b_addr*, *history*, *his_bit*, and n stand for branch address, global history register, number of history bits in index, and the width of index respectively. Note that *his_bit* is smaller than n and the width of index can remain unchanged after the combination. The detailed algorithm of the history skewed indexing is as follows.

Algorithm 1. Piecewise Linear Branch Predictor with History Skewed Indexing

```
function prediction (b_addr, history, his_bit: integer):boolean
begin
    i:= (b_addr<<his_bit) +
        (history&((1<<his_bit)-1)) mod 2^n;
    j:= (b_addr<<his_bit) +
        (history&((1<<his_bit)-1)) mod 2^m;
    k:= b_addr mod 2^(n-his_bit) ;
    output: = SR[k,h]+W[i,j,0];
    if output >= 0 then
        prediction := taken;
    else
        prediction := not_taken;
    endif
    for i in 0..( 2^(n-his_bit)-1) in parallel do
      for p in 1..h in parallel do
        a_p = h – p;
        w_index=(i<<his_bit)+(history&((1<<his_bit)-1));
        if prediction == taken then
          SR'[i, a_p+1] := SR[i, a_p] + W[w_index, j, p];
        else
          SR'[i, a_p+1] := SR[i, a_p] - W[w_index, j, p];
        endif
      endfor
      SR[i, 0..h] :=SR'[i,0..h];
      SR[i, 0] := 0;
    endfor
    history := (history<<1) | prediction;
end
```

The main advantage of introducing history bits into index is that it decouples the size of SR(R) from the index width n. Since there are only portion of index coming from branch address, the size of SR and R can be substantially smaller than the original one. As shown in the algorithm above, the number of ahead-pipeline paths is reduced to 2^{n-his_bit}, which means at least half of the area of SR(R) can be saved with history skewed indexing. Figure 2 shows the ahead-pipeline structure for piecewise linear predictor when n=2. The original one in [1] has four ahead-pipeline paths, however, the number of paths is reduced to two with one bit history skewed indexing. The introduced history bit is used to select the weights from the W array through a 4-to-2 multiplexer.

History skewed indexing also helps to improve the prediction accuracy of practical piecewise linear predictor.

The history bit brings path information to the index, therefore can reduce aliasing in a way similar with McFarling's gshare predictor [6]. Moreover, piecewise linear predictor delivers the highest prediction accuracy when *m* and *n* are approximately the same [1], yet it is extremely difficult to achieve it in practice due to hardware constraints. History skewed indexing, however, opens the possibility to tune *m* and *n* flexibly to achieve the best performance that piecewise linear predictor has to offer.

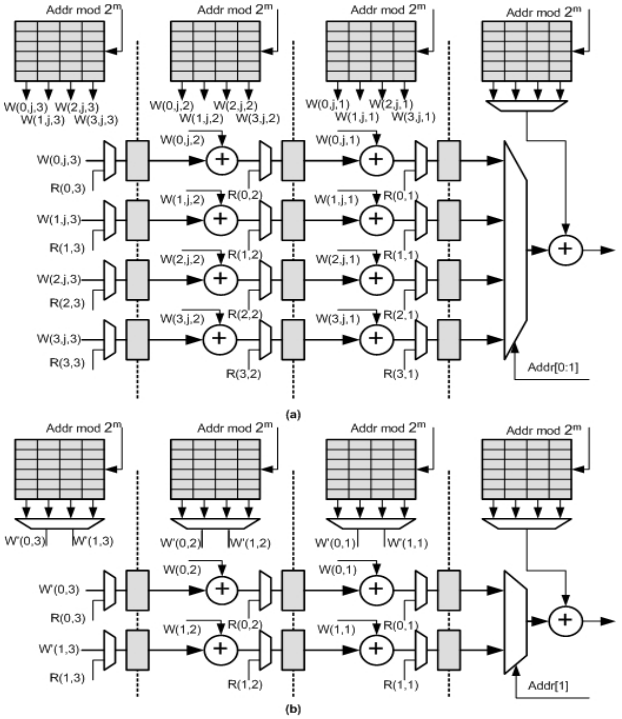

Figure 2 Ahead-pipeline in piecewise linear predictor with *n*=2. (a). Original ahead-pipeline proposed in [1]. (b). Ahead-pipeline with 1 bit history skewed indexing.

In practice, speculative global history is used due to the latency between branch prediction and resolve. Therefore, a small speculative global history register is required for this technique.

B. Recovering Predictor with Stack

The purpose of *R* is to restore the *SR* to the non-speculative start point from misprediction. Although it has the capability to restore the state in one clock cycle, it has a large area penalty and wastes power for most of the time considering the high accuracy of the predictor.

The basic idea of recovering predictor state with stack is based on the observation shown in the Figure 3. Assume the global history length *h* is 4, and each branch is predicted as *taken*. $W_i(i=1,2,3,4)$ is the weight for i_{th} branch instruction in the speculative path. On a misprediction, the state of *SR* can be recovered in the reversed order by subtracting the weights cycle by cycle. Therefore, *R* and associated adders are not necessary for misprediction recovery, which can further reduce area and avoid unnecessary power consumptions without losing prediction accuracy. The following subsections present the components required for this solution as well as the analysis of its disadvantages.

	SR_1 Contents	SR_2 Contents	SR_3 Contents	SR_4 Contents
Initial values	a	b	c	d
Branch 1	a1	a+w1	b+w1	c+w1
Branch 2	a2	a1+w2	a+w1+w2	b+w1+w2
Branch 3	a3	a2+w3	a1+w2+w3	a+w1+w2+w3
Branch 4	a4	a3+w4	a2+w3+w4	a1+w2+w3+w4
Branch 1 Mispredicts, Recover				
Branch 3	a3	a2+w3	a1+w2+w3	a+w1+w2+w3
Branch 2	a2	a1+w2	a+w1+w2	b+w1+w2
Branch 1	a1	a+w1	b+w1	c+w1

Figure 3. Misprediction recovery with multiple cycles. The solid arrow shows the forward *SR* update path during prediction; the dashed arrow shows the backward *SR* update path during recovery.

1) Stack for the Branches in Speculative Path

In the backward calculation for prediction recovery, the branch address in the speculative path is used as the index of *W* again, but in reversed sequence. As shown in Figure 3, the address of branch *4* indexes into *W* array for W_4, followed by branch *3* for W_3 and so on. This First In Last Out (FILO) restore sequence can be captured by a stack structure. In the worst case each instruction in the speculative path is branch instruction, which should be pushed onto the stack during prediction time. Therefore, the depth of the stack *d* depends on the number of in-flight branch instructions between instruction fetch and branch resolve. The content of each stack entry is composed of branch address and the information required for misprediction recovery. Because the bit number of branch address used as indices for *W* is bounded by *MAX(m,n)* , *q (q = MAX(m,n))* least significant bits of branch address should be stored in the stack. Two additional bits are needed, one for the prediction result of the corresponding branch, and the other for whether *W* array training is required. As a result, the size of the stack is $(q+2) \times d$ bit. It should be noted that these information also need to be stored in memory cells even in the original ahead-pipelined piecewise linear predictor proposed in [1]. Therefore, by substituting those memory cells with the stack, it will not contribute extra hardware cost.

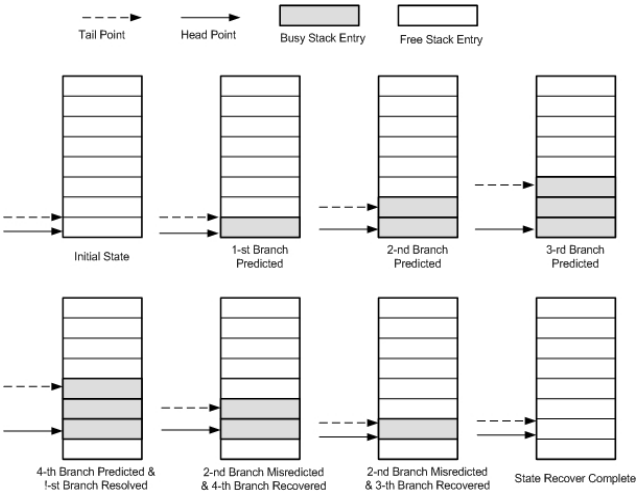

Figure 5. Stack operations in prediction and recovery

Figure 5 shows the detailed stack operations for both prediction and recovery. The stack maintains two pointers, i.e., tail pointer and head pointer. Each time a branch is

predicted, the data will be pushed onto the stack and the tail pointer will be incremented by one. When the branch that the head pointer points to is resolved and the prediction is correct, the head pointer is incremented by one. Otherwise, if it is a misprediction, the head pointer remains in its original position, and the entries between head and tail are popped up cycle by cycle. Therefore, the stack is a circular stack.

2) Structure for Backward Calculation

The ahead-pipeline has to be modified in order to support for backward calculation. Since forward prediction and backward recovery does not happen simultaneously, one adder can be used for both operations. Additional control logic is required to switch between the two operational modes. Figure 6 shows the logic structure between any two neighboring *SR* array elements, *SR[i,j]* and *SR[i,j+1]* (*i*=1..*n* and *j*=1..*h*-1). When the predictor is in the forwarding prediction mode, MUX1 selects the data from forward path L1. Otherwise, backward path L2 is chosen. The prediction result, either from the predictor or from the stack, is used to determine whether addition or subtraction is required. Since the backward calculation is opposite to the forward calculation, the prediction result from stack needs to be complemented.

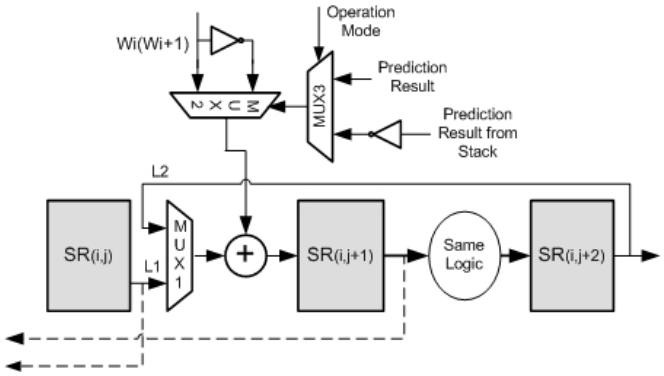

Figure 6 Modified ahead-pipeline supporting for backward calculation. Note that subtraction can be implemented by complementing the weight and setting the carry in port of the adder to one.

The critical path for the forward calculation of the ahead-pipelined piecewise linear predictor is composed of a table lookup, a multiplexer, an adder, and a comparator. On the other hand, the critical path of the proposed recovery structure includes a table lookup, one inverter, two multiplexers, and an adder. Note that the access to stack can be hidden with the rest of recovery operations , and that the overall delay of one inverter and one multiplexer is less than a comparator. Therefore, the proposed recovery technique will not lead to performance degradation of the ahead-pipeline.

3) Penalty of Stack Based Misprediction Recovery

With stack based misprediction recovery, the number of recovery cycles dynamically depends on the number of in-flight branch instructions between instruction fetch and branch resolve. Therefore, it may take multiple cycles to restore the predictor to the latest non-speculative state, as opposed to one cycle with the recovery scheme in [1]. In other words, stack based misprediction recovery could increase the branch misprediction penalty and lead to degradation of instruction-per-cycle (IPC) rate. Fortunately, the IPC loss due to the increase of misprediction penalty is small considering the high prediction accuracy of piecewise linear predictor. Only less than 3% IPC loss is observed in the experiment.

C. Combining Two Techniques

While history skewed indexing reduces the number of forward pipeline paths on both *SR* and *R* arrays, the stack based misprediction recovery completely removes the pipeline of R. The combination of these two techniques can further reduce the area requirement of the predictor. In addition, the potential prediction accuracy improvement with history skewed indexing can help to alleviate the IPC loss caused by stack based misprediction recovery.

IV. Experimental Results

This section describes the methods used to evaluate the proposed techniques. The analysis for the obtained data is presented as well.

A. Experiment Setup

The experiment is composed of two parts, i.e., architectural level simulation and hardware implementation. The former part is used to analyze the performance of the proposed techniques, and the latter is used to evaluate the effectiveness of area reduction.

The simulation is based on SimpleScalar [10] configured to support Alpha ISA. The microarchicture parameters of the simulated microprocessor are listed in Table I. All 12 SPEC CPU2000 integer benchmarks are selected for simulation. Data are collected after running 100 million instructions for each benchmark.

TABLE I
MICROARCHITECTURAL CONFIGURATION OF THE PROCESSOR SIMULATED

Parameter	Configuration
L1 I-Cache	16KB, 64Byte block, 2-way
L1 D-Cache	16KB, 64Byte block, 4-way
L2 Unified Cache	1MB, 64Byte block, 4-way
Decode/Issue Width	8
L2 hit latency	7 cycles
L2 miss latency	200 cycles

We also implemented piecewise linear predictors with Verilog HDL, and mapped them to TSMC 0.18µm standard cell library with Synopsys Design Compiler. The logic synthesis was performed in worst corner case with wireload model and 7ns clock timing constraint. The area data are collected according to the report of DesignCompiler. Although the area information reported in Design Compiler

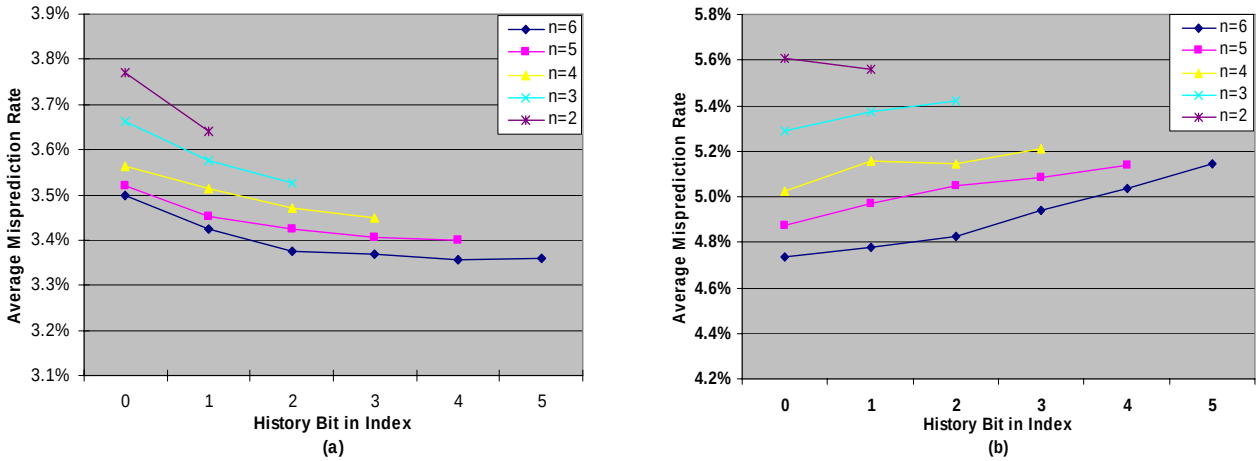

Figure 7 (a) Average misprediction rate of ideal piecewise linear predictor with history skewed indexing . (b)Average misprediction rate of ahead-pipelined piecewise Linear Predicctor with History Skewed Indexing. In both (a) and (b), *h*=16, and *m*=8.

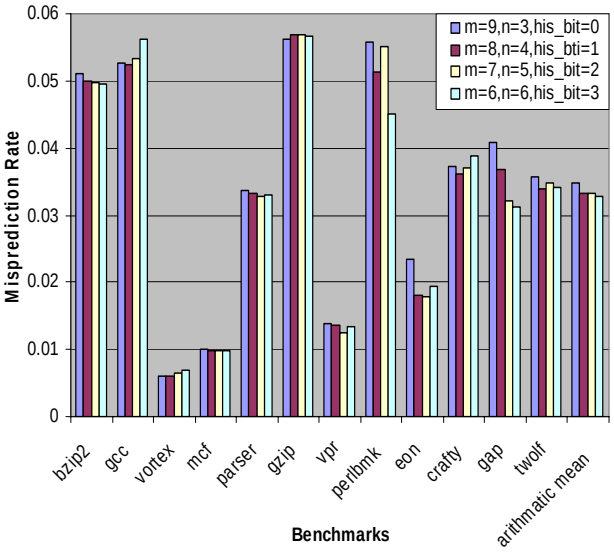

Figure 8. Misprediction rate per benchmark. The size of W array is $10x2^{12}$bit, and *h*=16 for all four configurations.

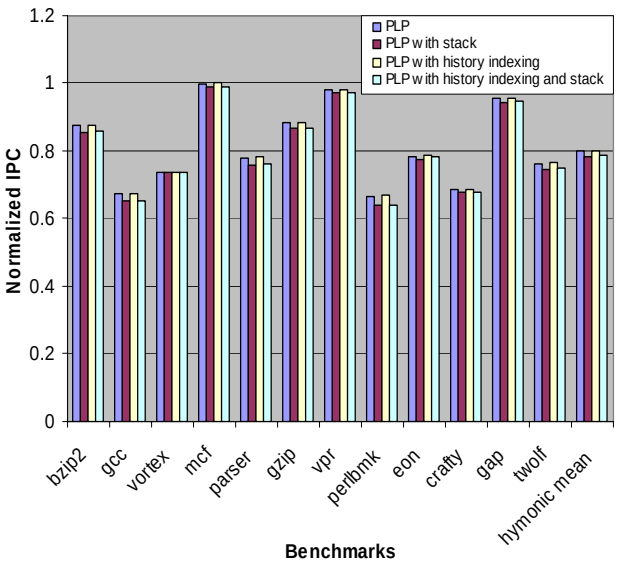

Figure 9. Normalized IPC rate. The size of W array is $10x2^{12}$bit, the number of ahead-pipeline path is 8, and *h*=16 for all cases

is only rough estimation of the final area in silicon, it provides enough precision for comparison between circuits with similar structures.

B. Experimental Result Analysis

Figure 7 shows the average misprediction rate of piecewise linear predictor with history skewed indexing. As the history bit increases, the misprediction rate of idealized one decreases, which means history skewed indexing helps improve prediction accuracy. However, the misprediction rate of ahead-pipelined predictor increases as the history bit in index increases. A total 8.6% misprediction rate degradation is observed as the history index bit number goes from zero to five. The reason for this is that the history bits in index are speculative in ahead-pipelined predictor, as opposed to non-speculative in idealized one.

Figure 8 shows the misprediction rate of ahead-pipelined piecewise linear predictor across different benchmarks. The hardware budget, including the size of *W* array and the number of paths in ahead-pipeline (8 in this case), is the same in all configurations. On average, the misprediction rate decreases by 5.3% as the number of history bits in index increases from 0 to 3. That means the benefit from balancing *m* and *n* removes the prediction accuracy degradation caused by speculative history bits in index.

Figure 9 highlights the effect of the proposed techniques on normalized IPC rate. Stack based misprediction recovery alone leads to 2% IPC loss compared with the original predictor. The IPC loss is reduced to 1.4% when both of the proposed techniques are employed. The reason is that with history skewed indexing, the predictor is tuned to be more accurate with the same hardware cost.

Table II summarizes the circuit area of piecewise linear predictor after logic synthesis. The data are normalized to the area of original piecewise linear predictor with the

configuration of *h*=7,*m*=4,*n*=3. It should be noted that the percentage of area reduction depends on the ratio of the area dedicated for the ahead-pipeline versus the area for the *W* array. Since both of them are linearly proportional to *h*, the increase of *h* has no effect on the percentage of area reduction. However, as *m* increases, the size of *W* would increase exponentially and the size of SR remains the same. Therefore, the percentage of area reduction will decrease.

TABLE II
NORMALIZED AREA OF PIECEWISE LINEAR PREDICTOR

Configuration	Normalized Area			Max. area Reduction
	Original	Stack Only	Stack and History*	
h=7,m=4,n=3	1	0.79	0.68	32.0%
h=10,m=4,n=3	1.38	1.10	0.93	32.6%
h=7,m=5,n=3	1.72	1.37	1.28	25.6%
h=10,m=5,n=3	2.43	1.95	1.84	24.3%

*Note that only 1 history bit is included in the index.

TABLE III
NORMALIZED AVERAGE IPC RATE

Configuration	Normalized IPC			Max.IPC Reduction
	Original	Stack Only	Stack and History*	
h=10,m=4,n=3	0.992	0.967	0.963	2.9%
h=10,m=5,n=3	1	0.976	0.975	2.5%

*Note that only 1 history bit is included in the index.

Table III shows the normalized average IPC rate for the hardware implemented predictors. Due to the small value of *h* and *m*, the prediction accuracy of these predictors is not as high as those in Figure 8. Therefore, the IPC degradation caused by the introduction of stack would be higher. However, the IPC rate decreases only by about 3%. In particular, for a piecewise linear predictor with *h=10, m=4* and *n=3*, the combination of the two techniques proposed in the paper achieves about 30% area reduction with 2.9% IPC loss.

V. CONCLUSION

This paper presents two techniques that can make the piecewise linear branch predictor hardware efficient in VLSI design, meanwhile maintaining its prediction accuracy. History skewed indexing is designed to reduce the number of ahead-pipelined paths by introducing the history bits in the index of weight table. It decouples the number of ahead-pipeline paths from the number of index bits, and opens the possibility to tune the width of the weight table indices for higher accuracy without additional hardware cost. Stack based misprediction recovery employs stacks to restore the predictor from misprediction. It completely removes the inefficient pipeline used for misprediction recovery without losing prediction accuracy. Experimental results show that history skewed indexing is able to improve the prediction accuracy by 5.8% without additional hardware cost. For a particular configuration, the combination of these techniques can achieve about 30% area reduction with less than 3% IPC loss compared with the original piecewise linear predictor.

REFERENCES

[1] Daniel A.Jimenez, "Piecewise linear branch predictor," *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA-32)*, June 2005.

[2] Daniel A.Jimenez and Calvin Lin, "Dynamic branch prediction with perceptrons," *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, pp.197-206, Jan. 2001.

[3] David Tarjan and Kevin Skadron, "Revisiting the Perceptron Predictor Again," *Technical Report CS-2004-28, University of Virginia Department of Computer Science*, Sept.2004.

[4] Daniel A. Jimenez, "Fast path-based neutral branch prediction," *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.43-52, Feb. 2002.

[5] A.Seznec, "Redundant History Skewed Perceptron Predictor: pushing limits on global history branch predictors," *Technical Report 1554, IRISA*, Sept. 2003.

[6] S.McFarling, "Combining Branch Predictors," *Technical Report TN-36, Digtial Western Research Laboratory, June 1993*.

[7] T.Y-.Yeh and Y.N.Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp.257-266, 1993.

[8] Oswaldo Cadenas, Graham Megson and Daniel Jones, "A New Organization for a Perceptron-Based Branch Predictor and Its FPGA Implementation, " *IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI'05)*, pp. 305-306, 2005.

[9] Andr´e. Seznec, Stephen Felix, Venkata Krishnan, and Yiannakakis Sazeides. "Design tradeoffs for the Alpha EV8 conditional branch predictor," In *Proceedings of the 29th InternationalSymposium on Computer Architecture*, May 2002.

[10] Doug Burger and Todd M. Austin., "The SimpleScalar tool set version 2.0," *Technical Report 1342, Computer Sciences Department, University of Wisconsin*, June 1997.