Bias-Free Neural Predictor

Dibakar Gope and Mikko H. Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
gope@wisc.edu, mikko@engr.wisc.edu

Abstract

Prior research in neurally-inspired perceptron predictors has shown significant improvements in branch prediction accuracy by exploiting correlations in long branch histories. However, systems with moderate hardware budgets typically restrict such perceptron predictors from correlating beyond 64 to 128 past branches and limit their capability to learn distant branch correlations, such as on the order of 1024 to 2048 branches deep. In this work, we propose Bias-Free Neural predictor that is structured to learn correlations only with non-biased conditional branches, aka. branches whose dynamic behavior varies during a program's execution. This, combined with a recency-stack-like management policy for the global history register, opens up the opportunity for a modest history length to include much older and much richer context to predict future branches more accurately. Bias-Free Neural predictor achieves 2.73 MPKI (mispredictions per 1000 instructions) for a 32KB storage budget and 2.1 MPKI for an unlimited budget.

1. Introduction

Prior research in neural-based perceptron predictors has been very successful in considerably increasing the branch prediction accuracy [3, 1] by correlating a branch's outcome with previously executed branches. However, a moderate hardware budget of 32-64KB restricts such state-of-the-art perceptron predictors to rely on the correlations found with only 64 to 128 recent branches in the dynamic execution stream to predict a branch. For a branch under prediction, some of the correlated branches may have appeared at a large distance, such as on the order of 512 to 1024 branches apart, in the dynamic execution stream. This can happen, for instance, if two dynamic instances of a branch observe the same recent histories but behave oppositely, then a longer history can potentially establish a correlation from these hard-to-predict branches with distinguishable distant branches. Prediction accuracy of the recently proposed ISL-TAGE predictor [4] further confirms that looking at much longer histories (of the order of 2000 branches) can provide useful information for prediction. However, scaling a state-of-the-art perceptron-based predictor [1] from 64KB to 1MB to track distant branch correlations results in long computational latency and high energy consumption in the large storage structures, which may prohibit the incorporation of such a branch predictor into a commercial processor. Furthermore, it causes a substantial increase in training time. In addition, all of the additional branches included may not be correlated and they preclude the inclusion of any highly correlated branches from deeper in the global history.

In this work, we propose Bias-Free Neural (BFN) predictor that utilizes the behavior of past non-biased conditional branches to predict a branch. Non-biased branches resolve in both directions whereas conditional branches that display only one behavior during the execution of a program are considered as "completely biased" [1] branches.

Our work builds on the observation that in order for a branch to establish an effective correlation with another branch, the change in the direction of one branch has to influence the direction of another. Since a biased branch is skewed towards one direction, the change in the direction of a non-biased branch can not establish any true correlation with that branch. As a result the prediction of a non-biased branch can not rely on the direction of a biased branch observed in the past history. A biased branch can sometimes merely reinforce a prediction decision already established by the correlation captured with another non-biased branch in the past global history.

Restricting the predictor to learn correlations only with non-biased branches enables a modest history length to reach very deep into the program's execution history to find correlated branches and provide highly-accurate branch predictions with a modest storage budget.

2. Key Idea

In this section we provide an overview of the two types of filtering used to collect older and richer context from the long global history and present an idealized version of the BFN predictor without paying attention to detecting biased branches at runtime. Biased branches are predicted with their behavior and excluded from the perceptron prediction and thus prevented from training and possibly aliasing with other weights.

2.1. Filtering biased branches from the history

Since biased branches provide virtually no useful context to the branch predictor's history, the BFN predictor only tracks a branch in the global history register if that branch is detected as non-biased at runtime.

[1] hereafter "completely biased" branches will be referred simply as biased branches

2.2. Filtering multiple instances from the history

The BFN predictor attempts to find branch correlations deeper into the global history within a limited hardware budget by filtering biased branches from the history. In order to capture even more distant branch correlations (of the order of 2000 branches deep) and improve the prediction accuracy further, BFN predictor only tracks the latest occurrence of a non-biased branch in the global history register and attempts to learn correlations with that occurrence. This optimization minimizes the footprint of a single non-biased branch in the path history of a branch and thus in turn assists in including any highly correlated branches from deeper in the global history within a modest length global history register.

The BFN predictor introduces a recency-stack-like (RS) structure to track the most recent occurrence of a branch in the history. When a non-biased branch PC_{nb} is committed, the RS structure is scanned to find the last occurrence of that branch. If the branch PC_{nb} hits in the RS, then it is moved to the top of the RS and updated with its recent outcome. The set of locations from the first position in the RS to the hitting entry are shifted by one position. In case of no entry is found with PC_{nb}, the RS acts like a conventional shift register.

Furthermore, in order to capture different correlations for different instances of a branch with the recent occurrence of a non-biased branch present in the RS, the non-biased branch includes it's positional history, *pos_hist* along with it's recent outcome in the RS and uses that during prediction and training. It's *pos_hist* conveys the absolute distance of the non-biased branch from the current branch in the past global history.

The following variables are used by the BFN prediction algorithm:

a) W_b, W_m: one-dimensional and two-dimensional arrays of integer weights respectively. W_b is the bias weight table, whereas W_m is the correlating weight table.

c) GHR: The global history register containing only the recent occurrence of non-biased branches as they are executed.

b) h: The size of the RS-like global history register.

d) A: An array of addresses of the non-biased branches in the past global history.

e) P: The absolute distance in the past global history of corresponding non-biased branches included in array A. In other words, P captures the *pos_hist* of the non-biased branches present in the RS.

f) accum: The dot-product of the weights vector chosen and the global history register.

In effect, the GHR in conjunction with the array A and the array P behaves as a RS.

Algorithm 1 shows the function predict that computes the Boolean prediction function. For each non-biased branch captured in the array A, the idealized version of BFN predictor hashes the branch address, the address of the non-biased branch and it's distance in the history recorded in P to select a row and uses it's depth in A to map to a column in W_m. That is, for every non-biased branch of every path, the predictor tracks the correlation of that branch in conjunction with it's recorded distance in the history. The correlations computed in this way for each component of the current path are aggregated to make a prediction.

Algorithm 1 BFN Prediction {Idealized version}

function *prediction* (*pc*: **integer**) : { *taken*, *not_taken* }
if *pc* is "completely biased" branch **then**
  $prediction \leftarrow bias\_direction$
else
  $accum \leftarrow W_b[pc \bmod n]$
  for $i \leftarrow 1 \,..\, h$ **do in parallel**
    $row\_index \leftarrow hash(pc \text{ xor } A[i] \text{ xor } P[i]) \bmod n$
    $accum \leftarrow accum + W_m[row\_index,\ i] * GHR[i]$
  end for
  $prediction \leftarrow (accum \geq 0)?\ taken : not\_taken$
end if

Training: As branches are committed, the weights used to predict a non-biased branch are updated according to conventional perceptron learning [3]. The weights are not updated if a biased branch commits. When a non-biased branch commits, the RS-like management policy updates the GHR, A and P.

2.3. Folded Global History

In order to compute the indexes for accessing the correlating weights, prior studies on perceptron-based prediction [1, 2] consider hashing the branch addresses in path history with the current branch to be predicted. However, sometimes in spite of being captured in the same relative depth in A and in the same absolute distance in the past global history, a non-biased branch can influence the prediction decision of the current branch differently if the execution paths from the non-biased branch to the current branch differ.

In order to limit this phenomenon, for each non-biased branch captured in A, the hash function outlined in Algorithm 1 to index the perceptron counters is augmented with global history bits from the non-biased branch leading up to the current branch. When the number of global history bits exceeds the number of bits used in the predictor index function, the global history is "folded" by a bit-wise XOR of groups of consecutive history bits and is hashed down to the required number of bits for the predictor index.

3. Implementation

In this section we present a simple hardware structure to detect the non-biased branches on the fly and describe the required structural modifications to the perceptron weight table to minimize the perturbations caused by the dynamic detection of non-biased branches as execution advances.

3.1. Biased Branch Detection

The biased branch detection logic is controlled by a simple finite state machine (FSM) that operates in one of four possible

Algorithm 2 BFN Prediction {Practical Implementation}

function *prediction* (*pc*: **integer**) : { *taken*, *not_taken* }
if $BST[pc \bmod m] == Not\,found$ **then** /* m is the number of entries in BST */
  $prediction \leftarrow taken/not\_taken$
else if $BST[pc \bmod m] == Taken/Not\,taken$ **then**
  $prediction \leftarrow BST[pc \bmod m]$
else
  $accum \leftarrow W_{\mathrm{b}}[pc \bmod n]$ /* n is the number of entries in bias weight table W_{b} */
  for $i \leftarrow 1 \,..\, h_{\mathrm{t}}$ **do in parallel**
    $row\_index \leftarrow hash(pc \text{ xor } A[i] \text{ xor } folded\_hist[i]) \bmod n$
    $accum \leftarrow accum + W_{\mathrm{m}}[row\_index,\ i] * GHR_{\mathrm{unfiltered}}[i]$ /* n is the number of rows in 2-dim weight table W_{m} */
  end for
  for $i \leftarrow 1 \,..\, h - h_{\mathrm{t}}$ **do in parallel**
    $table\_index \leftarrow hash(pc \text{ xor } RS[i].A \text{ xor } RS[i].P \text{ xor } folded\_hist[RS[i].P]) \bmod p$
    $accum \leftarrow accum + W_{\mathrm{rs}}[table\_index] * RS[i].H$ /* p is the number of entries in 1-dim weight table W_{rs} */
  end for /* RS is the Recency Stack. Each entry have A, P and H fields that contain the address, absolute distance and outcome of the latest occurrence of a branch */
  $prediction \leftarrow (accum \geq 0)?\, taken : not\_taken$
end if

Algorithm 3 Training {Practical Implementation}

function *training* (*pc*: **integer**, *branch_direction*: **boolean**)
if $BST[pc \bmod m] == Not\,found$ **then**
  $BST[pc \bmod m] \leftarrow branch\_direction$
else if $prediction \neq branch\_direction$ and $BST[pc \bmod m] == Taken/Not\,taken$ **then**
  $BST[pc \bmod m] \leftarrow Non\_biased$
  Update weights in W_{b}, W_{m}, W_{rs}
else if $BST[pc \bmod m] == Non\_biased$ and ($|accum| < \theta$ or $prediction \neq branch\_direction$) **then**
  Update weights in W_{b}, W_{m}, W_{rs}
end if
if $BST[pc \bmod m] == Non\_biased$ **then**
  Update RS
end if
Update $GHR_{\mathrm{unfiltered}}$

states: *Not found*, *Taken*, *Not taken* or *Non-biased*.

Until a conditional branch is encountered for the first time, the FSM relating to it's status stays in the *Not found* state. The status of a branch is identified by consulting a structure called the Branch Status Table (BST). The BST is a direct-mapped structure that records information relating to the past behavior of branches. When a prediction is to be made for a conditional branch detected in the *Not found* state, the aggregated correlation from the perceptrons is not considered. When this conditional branch is subsequently committed for the first time, the detection FSM transitions from the *Not found* state to one of two possible states: *Taken* or *Not taken* depending on the outcome of the branch. The *Taken* and *Not taken* state exists to record the biased direction of a previously unknown branch in the BST and used to predict the future instances of the branch. In the event a branch in either *Taken* or *Not taken* state executes in the opposite direction that differs from the recorded state, the detection FSM transitions to the *Non-biased* state. Any future instances of this branch are predicted using perceptron computation and contribute to the GHR, A and P arrays and thus assist other non-biased branches to establish correlations with that branch.

Note that the state-of-the-art perceptron-based predictors [1, 2] as well as the idealized version of BFN predictor outlined in Algorithm 1 use the depth of a captured branch in the RS to map to a column in the two-dimensional weight table $\mathrm{W_m}$. In our implementation, all branches begin being predicted considering as biased until they transition to the *Non-biased* state in the BST. Furthermore, until a branch is detected as *Non-biased*, it does not contribute to the history for future branches. In the event a branch is detected as *Non-biased* using the FSM transitions as described above, it starts placing it's path history into the RS, which results in shifting the relative depths of previously detected non-biased branches in the RS. This necessitates those previously detected non-biased branches to re-learn correlations in the new relative depths in the RS in spite of possibly being in the same absolute distances in the past global history, resulting in hurting the accuracy.

Our implementation solves this issue by making use of a one-dimensional correlating weight table; this eliminates perturbations induced by the occurrences of a newly detected non-biased branch in the history.

3.2. One-Dimensional Weight Table

Our implementation stores the correlations in a one-dimensional array of integer weights instead of maintaining those in a two-dimensional weight table as outlined in Algorithm 1. Now for each non-biased branch captured in the RS, the one-dimensional weight table is indexed using a hash function of the current branch to be predicted, the address of the non-biased branch, it's absolute depth in the history and the folded global history leading up to the current branch as discussed in Section 2.3. Since the previously detected non-

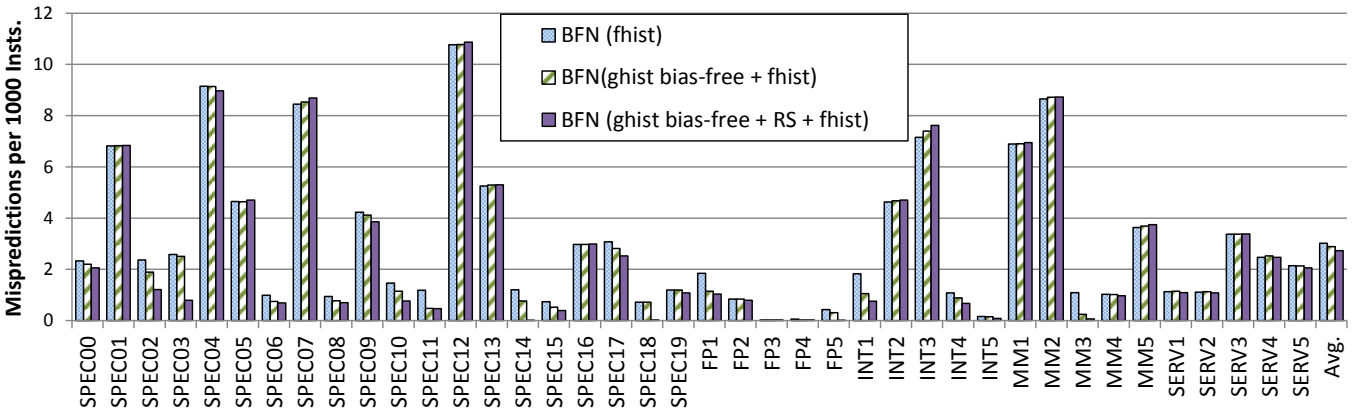

Figure 1: Contributions of Optimizations for the BFN Predictor.

biased branches do not depend anymore on the relative depths in the RS to index to columns in the correlating weight table, they do not require re-learning their correlations.

The BFN predictor is very effective in capturing very distant branch correlations. However it does not perform that well on some branches that have a very strong bias towards one direction, but do not find good correlations at remote histories. For these branches, until the set of non-biased branches present in the recent history develop strong correlations, the BFN approach cannot outweigh the bias weight to produce the unlikely predictions. As a result, during the training phase BFN predictor performs poorly than a conventional perceptron predictor for those branches and causes sizable number of mispredictions.

In order to address this perceptron predictor artifact and avoid the mispredictions caused by this class of branches, we incorporate a conventional perceptron predictor component that captures correlations for few recent unfiltered history bits. The presence of few recent unfiltered history bits essentially assists other non-biased branches in the global history register to outweigh the bias weight and avoid some mispredictions during the training phase. Furthermore, BFN predictor sometimes fails to predict loops with constant number of iterations. The loop count (LC) predictor is used to predict these loops.

Algorithm 2 presents the BFN Prediction function and Algorithm 3 outlines the Training used to update the BST and the weight tables. W_{b} is the array of bias weights, W_{m} is the two-dimensional conventional perceptron weight table, whereas W_{rs} is the one-dimensional weight table. h_{t} is the number of recent branches tracked using the conventional perceptron predictor component. $GHR_{\mathrm{unfiltered}}$ is the global history register containing the outcomes of all branches. Table 1 shows the computation of the size of the state used for the predictor.

4. Results and Conclusion

Figure 1 demonstrates the contributions of individual optimizations to accuracy. All three bars use folded global history (*fhist*) to index the perceptron counters. The leftmost bar shows the accuracy achieved with identifying the biased branches using BST and preventing them from using the weight tables. However, this does not restrict the biased branches from updating the global history register. This optimization improves the average MPKI from 3.02 to 2.89. The next bar reflects the improvement when BFN predictor does not include biased branches in the global history register i.e. learn correlations only with non-biased branches. The rightmost bar demonstrates the improvement with the RS-like management policy for the global history register. This optimization improves the MPKI from 2.89 to 2.73.

Table 1: Total predictor storage budget

Source	Quantity of bits
BST	8192 entries × 2-bits/entry = 16,384
W_{b}	1024 weights× 6-bits/weight = 6,144
W_{m}	1024 × 11 × 6-bits/weight = 67,584
W_{rs}	32768 weights × 5-bits/weight = 1,63,840
RS	36 depth × (15 + 11 + 1) = 972; 15-bit tag *RS*[ ].*A*, 11-bit *RS*.[ ]*P*, 1 bit T/NT *RS*[ ].*H*
Filtered Hist.	128 entries × 26-bits/entry = 3,328
LC Predictor	2,368
Total	260,620 bits = 31.81KB

In this work, we propose BFN predictor that learns correlations only with non-biased branches and enables a modest storage budget of 32KB and history length of 47 bits to reach very deep into the program's execution history.

Acknowledgments

This work was supported in part by NSF grants CCF-1116450 and CCF-1318298.

References

[1] D. A. Jimenez, "Piecewise Linear Branch Prediction," in *International Symposium on Computer Architecture*, June 2005.

[2] D. A. Jimenez, "An optimized scaled neural branch predictor," in *International Conference on Computer Design*, October 2011.

[3] D. A. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *International Symposium on High Performance Computer Architecture*, February 2001.

[4] A. Seznec, "A 64 kbytes ISL-TAGE branch predictor," in *3rd Championship Branch Prediction*, June 2011.