Bias-Free Branch 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 and Geometric History Length-based TAGE predictors has shown significant improvements in branch prediction accuracy by exploiting correlations in long branch histories. However, not all branches in the long branch history provide useful context. Biased branches resolve as either taken or not-taken virtually every time. Including them in the branch predictor's history does not directly contribute any useful information, but all existing history-based predictors include them anyway.**

In this work, we propose Bias-Free branch predictors that are 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. With a 64KB storage budget, the Bias-Free predictor delivers 2.49 MPKI (mispredictions per 1000 instructions), improves by 5.32% over the most accurate neural predictor and achieves comparable accuracy to that of the TAGE predictor with fewer predictor tables or better accuracy with same number of tables. This eventually will translate to lower energy dissipated in the memory arrays per prediction.

***Keywords*-branch filtering, branch correlation.**

I. Introduction

Modern high-performance deeply pipelined, wide-issue microprocessors rely on sophisticated branch predictors to continuously supply the core with right-path instructions. Prior research in neural-based perceptron predictors has been very successful in considerably increasing the branch prediction accuracy [1]–[3] 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. Furthermore, if two correlated branches are separated by a function call containing many branches, a longer history is likely to capture the correlated branch that appeared in the dynamic execution stream prior to the function call.

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 in the global history register 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 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 length global history register 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.

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

Since the last few years, the TAGE predictor [5]–[7] is considered the most accurate branch predictor in the academic literature. TAGE relies on several predictor tables indexed through progressively longer global history. Such a sizable number of table accesses every processor cycle can potentially lead to considerable power consumption per prediction. Employing the Bias-Free approach enables a TAGE-style predictor with fewer tables without significantly impairing its accuracy.

To summarize, the key contributions are:

1. Filtering biased branches from the history: Bias-Free predictor is structured to learn correlations only with non-biased branches. It detects the non-biased branches on the fly using a cost-effective hardware and proposes hardware solutions to mitigate issues that arise from the dynamic detection.

2. Filtering multiple instances of branches from the history: Many branch instructions repeat in the history and the additional instances often provide little or no useful context. Bias-Free predictor introduces a structure called recency stack to track only the most recent occurrence of a branch in the history.

3. Bias-Free Neural Predictor (BF-Neural): While state-of-the-art neural predictors scale poorly with the history length, BF-Neural can reach very deep into the history (of the order of 2000 branches) with a modest history length of 64 branches. A 64KB BF-Neural predictor achieves 2.49 MPKI, improves by 5.32% over the most accurate neural predictor, OH-SNAP [8].

4. Bias-Free TAGE Predictor (BF-TAGE): BF-TAGE consistently provides better accuracy than conventional TAGE [4] for small to moderate number of predictor tables. A 10 tagged table BF-TAGE closely matches the accuracy of a 15 tagged table TAGE for most of the long history-sensitive traces.

The remainder of this paper is organized as follows. Section II motivates our proposal. Section III presents the idea of Bias-Free prediction. Section IV details the design and implementation of the BF-Neural predictor. Section V applies Bias-Free prediction to a TAGE-style predictor. Section VI presents accuracy results and associated analysis. Section VII discusses related work and Section VIII concludes the paper.

II. Motivation

Figure 1 shows a control flow graph (CFG) in which Branch A and E are found to be non-biased branches, whereas Branch B, C and D are biased branches. We are interested in predicting the Branch E, the last branch in the CFG. Clearly in this example Branch A and B serve as histories for younger Branch E on the execution path (A-B-E), whereas Branch A, C and D serve as histories on the execution path (A-C-D-E). Furthermore, it is observed that Branch E exhibits the opposite behavior if the execution path leading up to the Branch E changes from A-B to A-C-D. Since the biased branches B or C and D execute only on one of the program paths to the Branch E, the weights associated with Branches B or C and D along the two program paths will develop strong correlations to influence the prediction decision of the Branch E [1]. However, it is not difficult to observe that it is the non-biased Branch A that steers the control flow through either the Branch B or the Branches C and D that subsequently leads up to the Branch E. As a result, the change in the direction of the Branch A provides sufficient evidence to cause a change in the direction of the Branch E. In other words, the control flow through either the Branch B or the Branches C and D in the two program paths merely reinforces the prediction decision of the Branch E that can independently be established by correlating only with the non-biased branch A.

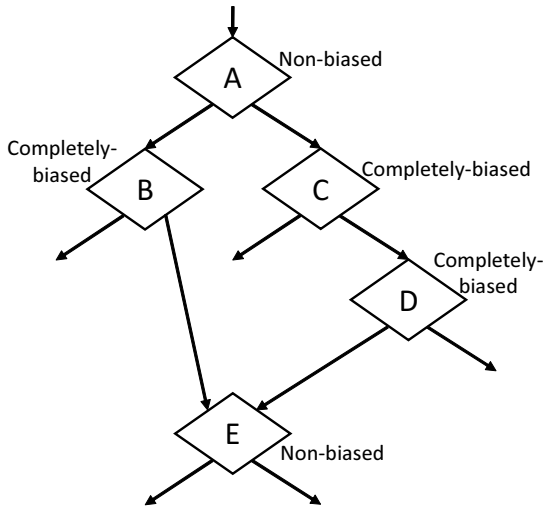


Figure 1: An Example Control Flow Graph.

Our work builds on this observation and learns the correlations only with non-biased conditional branches to predict a branch. Figure 2 demonstrates the presence of biased branches across the traces provided for the 4$^{\text{th}}$ Championship Branch Prediction [9].

III. Bias-Free Prediction

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 as well as the required structural modifications to the branch predictor's logic.

A. Filtering biased branches from the history

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

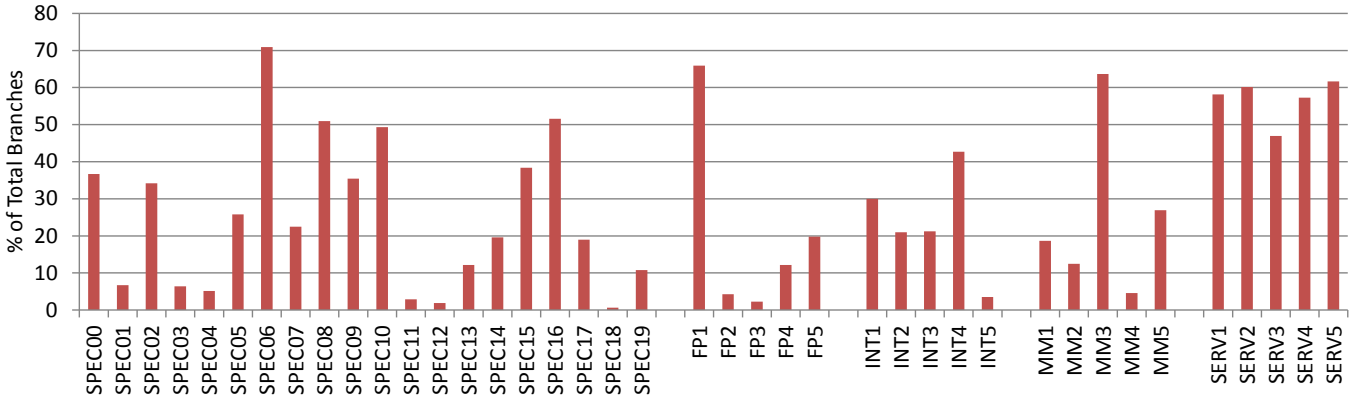


Figure 2: Biased Branches. SPEC: SPEC2006, FP: Floating-Point, INT: Integer, MM: Multi-media, SERV: Server.

B. Filtering multiple instances from the history

The Bias-Free 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, Bias-Free 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 Bias-Free predictor introduces a recency-stack-like (RS) structure to track the most recent occurrence of a branch in the history. Figure 3 shows a possible implementation of a 4-entry RS structure. Let PC_x, PC_y and PC_z be the PCs of the three most recent branches present in the RS. 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. The associated OR gate of the hitting entry guarantees downstream flip-flops to be clock gated. This results in downstream flip-flops to retain the most recent outcome of other non-biased branches. In case of no entry is found with PC_{nb}, the RS acts like a conventional shift register.

C. Positional History

Although including multiple instances of a non-biased branch in the global history register often provides little or no useful context, however a single instance of the branch captured in the RS can sometimes influence a following branch in both directions. Figure 4 demonstrates a example code pattern which seems to be common in our experimental

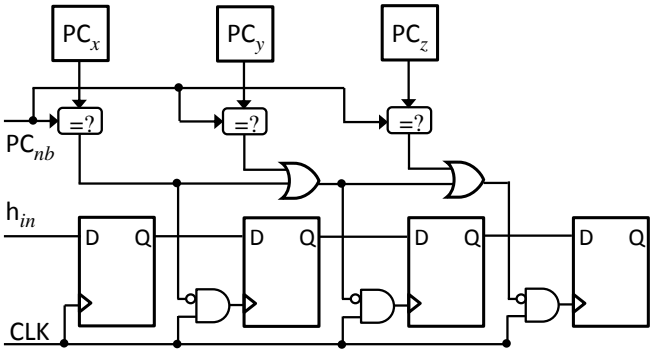


Figure 3: Recency Stack (RS) Design.

workloads. Here in this example, only one instance of the branch X across all the iterations of the loop L is strongly correlated with the branch A. In remaining iterations, it is not. Since the RS keeps only one occurrence of non-biased branches A and L, the Bias-Free predictor ends up with observing the same filtered global history across all instances of the branch X during the course of the loop execution. This results in mispredicting the instance of the branch X that exhibits the unlikely direction.

Hence, 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 its positional history, *pos_hist* along with its recent outcome in the RS and uses that during prediction and training. Its *pos_hist* conveys the absolute distance of the non-biased branch from the current branch in the past global history.

IV. BF-Neural Predictor

In this section we present an idealized version of the BF-Neural predictor without paying attention to detecting biased branches at runtime. We then derive a practical implementation of this idealized predictor. Furthermore, biased branches are predicted with their behavior and excluded from the

```
#define p 10
#define loop_count 100
int variable, array[loop_count];
.....
if (variable == 0)          //Branch A
   array[p] = 1;
for (i = 0; i < loop_count; i++)
{                           //Branch L
   if (array[i] == 1)       //Branch X
   {
      .....
   }
}
```

Figure 4: Example Code Illustrating the Requirement of Positional History.

perceptron prediction and thus prevented from training and possibly aliasing with other weights.

The following variables are used by the BF-Neural 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 GHR.

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 GHR.

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

Algorithm 1 shows the idealized BF-Neural prediction function. For each non-biased branch captured in A, the function hashes the branch address, the address of the non-biased branch and its distance in the history recorded in P to select a row and uses its 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 its recorded distance in the history. The correlations computed in this way for each component of the current path are aggregated to make a prediction.

Training: As branches are committed, the weights used to predict a non-biased branch are updated according to conventional perceptron learning [2]. 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.

Algorithm 1 BF-Neural 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

A. Folded Global History

In order to compute the indexes for accessing the correlating weights, prior studies on perceptron-based prediction [1], [8] 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.

For instance, let us consider a branch B which can be reached by two different paths corresponding to history H and history H' respectively. Furthermore the behavior of the branch B is influenced by the prior branch C present in the n^{th} position in A in both paths, but in opposite directions. This kind of phenomenon not only increases the destructive aliasing on the perceptron table entries associated with recent histories, but also sometimes ends up with accumulating significant noises from branches in distant path histories. Hence the impact of this kind of aliasing on predictor accuracy is particularly important.

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.

B. 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.

Algorithm 2 BF-Neural Prediction {Practical Implementation}

function *prediction* (*pc*: **integer**) : { *taken*, *not_taken* }
if $BST[pc \bmod m] == Notfound$ **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
  $accum \leftarrow W_{\text{b}}[pc \bmod n]$ /* n is the number of entries in bias weight table W_{b} */
  for $i \leftarrow 1$ .. h_{t} **do in parallel**
    $row\_index \leftarrow hash(pc \text{ xor } A[i] \text{ xor } folded\_hist[i]) \bmod n$
    $accum \leftarrow accum + W_{\text{m}}[row\_index,\ i] * GHR_{\text{unfiltered}}[i]$/*$n$ is the number of rows in 2-dim weight table W_{m}*/
  end for
  for $i \leftarrow 1$ .. $h - h_{\text{t}}$ **do in parallel**
    $table\_index \leftarrow hash(pc \text{ xor } RS[i].A \text{ xor } RS[i].P \text{ xor } folded\_hist[RS[i].P]) \bmod p$
    $accum \leftarrow accum + W_{\text{rs}}[table\_index] * RS[i].H$ /* 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

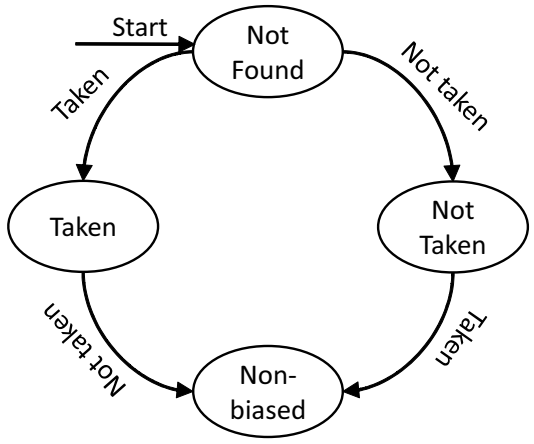


Figure 5: Biased Branch Detection Finite State Machine.

1) Biased Branch Detection: The biased branch detection logic in our work is controlled by a simple finite state machine (FSM) as shown in Figure 5. This operates in one of four possible states: *Not found*, *Taken*, *Not taken* or *Non-biased*. Encoding four possible states requires 2-bit counters.

Until a conditional branch is encountered for the first time, the FSM relating to its 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 correlations 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.

Algorithm 3 Training {Practical Implementation}

function *training* (*pc*:**integer**, *branch_direction*:**boolean**)
if $BST[pc \bmod m] == Notfound$ **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 $GHR_{\text{unfiltered}}$

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.

Probabilistic Counters: In this work, we use simple 2-bit counters in the BST to conduct a feasibility study of our bias-free approach. However, for incorporating such a predictor into a commercial processor, we advocate using probabilistic 3-bit counters [10] in the BST. The probabilistic

counter update mechanism can classify branches into different categories according to the frequency with which they exhibit a particular direction and can revert from non-biased to biased as application changes phase.

Note that the state-of-the-art perceptron-based predictors [1], [8] as well as the idealized version of BF-Neural 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 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 its 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.

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, its absolute depth in the history and the folded global history leading up to the current branch as discussed in Section IV-A. Since the previously detected non-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.

BF-Neural 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 BF-Neural approach cannot outweigh the bias weight to produce the unlikely predictions. As a result, during the training phase BF-Neural 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 to outweigh the bias weight and avoid some mispredictions during the training phase. Furthermore, BF-Neural predictor sometimes fails to predict loops with constant number of iterations. The loop count (LC) predictor is used to predict these loops.The LC predictor used in this work features only 64 entries and is 4-way skewed associative.

3) Conventional Perceptron Predictor Component: It accumulates the correlations over few recent unfiltered history bits and combines that with the correlations obtained from the one-dimensional weight table as discussed in Section IV-B2 to predict a branch.

Algorithm 2 presents the BF-Neural 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. $GHR_{\text{unfiltered}}$ is the global history register containing the outcomes of all branches.

V. BF-TAGE Predictor

The TAGE predictor is currently the most accurate conditional branch predictor in the academic literature [4]. It can capture correlations with very old branches (up to 2000 branches as shown in [4]).

A. Conventional TAGE Predictor Overview

Figure 6 presents the organization of a conventional TAGE predictor. The TAGE predictor comprises a base predictor T0 in charge of providing a basic prediction and a set of (partially) tagged predictor tables T_i. These tagged predictor tables T_i, $1 \leq i \leq M$ are indexed using different global history lengths that form a geometric series, i.e, $L(i) = (int)(\alpha^{i-1} * L(1)+0.5)$ as introduced for the OGEHL predictor [11]. At prediction time, the base predictor and the tagged tables are accessed simultaneously. The base predictor provides a default prediction. The tagged tables provide a prediction only on a tag match. In the general case, the longest matching history is used to make a prediction. The base predictor is a simple PC-indexed 2-bit counter bimodal table. An entry in a tagged table consists of a 3-bit signed counter whose sign provides the prediction, a (partial) tag and an useful bit. The useful bit locks entries in the tagged tables and thus prevents evicting useful entries from the predictor.

A $(n + 1)$ table TAGE consists of n tagged tables and a base bimodal predictor. With a 64KB storage budget, TAGE achieves best accuracy using 15 tagged tables [4]. The techniques proposed in the recent work [6] reduces the implementation complexity, the silicon footprint and the energy consumption of TAGE significantly, thus opening up the opportunity for incorporating such a branch predictor into a commercial processor. Nevertheless, such a sizable

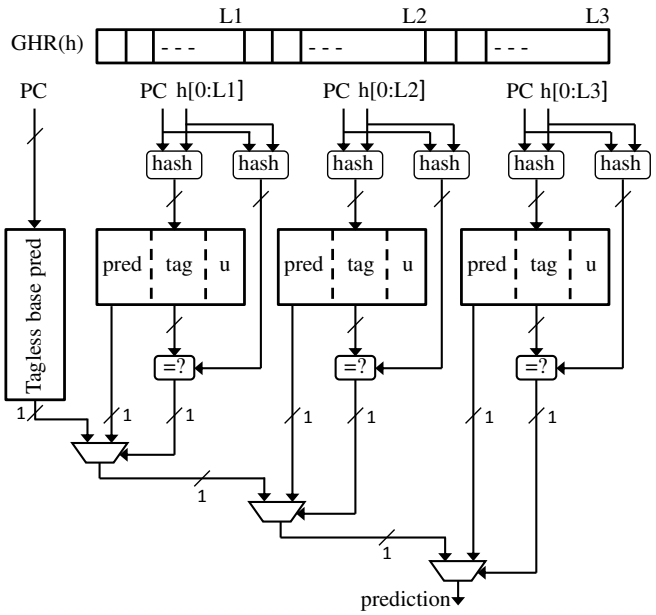


Figure 6: A 4-table TAGE predictor synopsis: a base predictor is backed with several tagged predictor tables indexed with increasing history lengths.

number of table accesses every processor cycle can potentially lead to a considerable power consumption. BF-TAGE demonstrates the potential to closely match the accuracy of a 15 tagged table TAGE with fewer tables, thus reducing the power consumption of the predictor even further.

B. BF-TAGE Predictor Design

In this section, we detail the design of the BF-TAGE predictor that incorporates the bias-free filtering approach into the global history register to realize a TAGE-like accuracy but with fewer tagged tables.

1) Organization: The BF-TAGE predictor is structured to consider only the non-biased branches in the global history register. Furthermore, RS-like management policy captures only the most recent occurrence of a non-biased branch in the global history register. However, for a few workloads, there exist a small set of branches that exhibit repeated occurrences throughout the program execution. Since the RS keeps only the most recent occurrence in the history, it can not include much older context in those cases to predict future branches more accurately. Furthermore, in many workloads, there exist branches whose behaviors correlate well with their own local history. This necessitates the RS structure to include few additional recent occurrences of a non-biased branch in addition to the most recent occurrence in the global history register. However this results in a sizable increase in the size of the RS. The associative search of the RS makes it impractical to scale beyond some small number of entries.

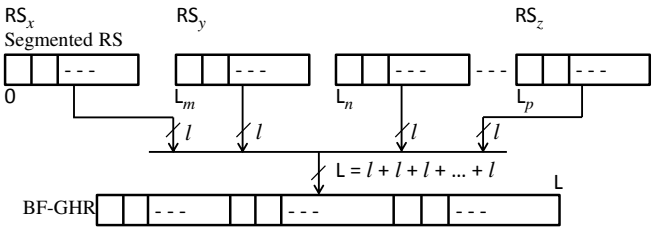


Figure 7: Bias-Free History Generation for TAGE Predictor.

The BF-TAGE predictor solves this issue by dividing the long global history into non-overlapping segments, each of which can be handled by a small, implementable RS. This leads to the design of the bias-free global history register (BF-GHR) as shown in Figure 7. The BF-GHR is constructed from M distinct RSs RS$_i$ that covers non-overlapping segments of the global branch history. Each of these individual RSs has a size of l. The size of the non-overlapping segments form a geometric series as introduced for the OGEHL [11] and later used in the TAGE [4]. Note that history segmentation does not change the overall size of the BF-GHR in terms of number of entries and comparators. However, the segmented RS has lower latency and access power than a monolithic RS, since the associative lookups are now localized to each segment. Each of the segmented RSs includes only a single instance of a non-biased branch from the corresponding history segment. In Figure 7, the RS$_y$ captures the recent instance of non-biased branches from the segment that covers past global histories from depth of L_m to L_n. The size of the per-segment RS (RS$_x$, RS$_y$, ..., RS$_z$ in Figure 7) is much smaller than the history segments that they cover, resulting in capturing long global histories (up to 2000 branches in our experiments) in about $150 - 200$ bits of the BF-GHR. This necessitates the tagged tables T$_i$, $1 \leq i \leq M$ to be indexed using modified history lengths that take into account this compression in the BF-GHR. This clearly differs from the history lengths used to index the tagged tables in conventional TAGE [4]. The branch address, the BF-GHR and a (limited) 16-bit path history consisting of 1 address bit per branch is hashed together to index into the tagged tables.

2) Correlation Redundancy: BF-TAGE draws the idea of history segmentation from prior work [12]. [12] partitions the branch history register into smaller segments, each of which can be easily handled by a small pattern history table (PHT). In contrast to that, we leverage history segmentation to mitigate the associative search problem in the RS structure. The small size of the per-segment RS may preclude the inclusion of all non-biased branches from the associated history segment. We argue that several non-biased branches in the global history are strongly cross-correlated. The presence of few of those non-biased branches in a small RS should capture correlations conveyed by the non-biased

branches not included in the RS. As a result, if some of the non-biased branches from a history segment fail to fit into the associated RS, it may not impair the predictability of the branch as other correlated branches still exist in that RS.

3) Prediction Computation: At prediction time, the per-segment RSs in increasing depth of histories acts together as the global history register. The remaining mechanism of the prediction computation stays the same as in [4].

4) Predictor Update: When a branch B commits, it is inserted into the $GHR_{\text{unfiltered}}$ along with its bias status and the hashed address. It keeps on moving deeper into the history as new branches commit. When B reaches a depth of L_m (Figure 7) in $GHR_{\text{unfiltered}}$, if it is non-biased, its hashed address is inserted into the RS_y, any other entry with the same hashed address is evicted from RS_y. Later when B reaches a depth of L_n in $GHR_{\text{unfiltered}}$, it falls out of RS_y and is considered for the next RS in Figure 7. The prediction and the useful counters are updated similarly as in [4].

VI. Experimental Framework and Results

Our evaluation is divided into three subsections. In subsection VI-B, we present the prediction accuracy results of BF-Neural and compare that with other recent state-of-the-art predictors. Subsection VI-C demonstrates the prediction accuracy results of BF-TAGE and compare that with conventional TAGE [4]. Finally, subsection VI-D provides detailed per-benchmark analysis of BF-TAGE predictor's impact on the number of tagged tables accesses per prediction.

A. Methodology

We use the trace-driven evaluation framework provided for the 4^{th} Championship Branch Prediction [9]. We present branch predictor accuracy as number of mispredictions per 1000 instructions (MPKI). The benchmark set comprises 40 traces: 20 long traces (approximately 15-30 million conditional branches) derived from SPEC2006 benchmark suite and 20 short traces (approximately 3-5 million conditional branches) derived from various categories: floating-point (FP), integer (INT), multi-media (MM) and server (SERV) workloads.

We compare BF-Neural with the following predictors:

1) ISL-TAGE [4]: This was ranked 1st at the 3rd CBP [13]. We use the original author's code available from the CBP-3 website [13].

2) OH-SNAP [8]: This is currently the most accurate neural predictor [13]. OH-SNAP builds on piecewise linear branch predictor [1] and introduces dynamic weight adaptation along with few optimizations. We leverage the original author's code available from the CBP-3 website [13].

B. BF-Neural Predictor

Figure 8 compares the performance of BF-Neural predictor with other predictors. The storage budget considered for the BF-Neural and the baseline predictors are approximately 64KB. The BF-Neural predictor features a BST with 16384 entries, a 2-dimensional weight table with 1024 rows and 16 columns, a 1-dimensional weight table with 65536 entries and a RS of depth 48. Note that the results shown in Figure 8 include same sized loop count predictor for BF-Neural and baseline TAGE. The baseline TAGE predictor in Figure 8 does not include the statistical corrector (SC) and the immediate update mimicker (IUM) components from the ISL-TAGE [4]. BF-Neural with loop predictor in Figure 8 requires the same storage space to that of the ISL-TAGE without SC and IUM components.

OH-SNAP achieves an average (arithmetic mean) MPKI of 2.63, whereas BF-Neural delivers a MPKI of 2.49. TAGE obtains 2.445 MPKI. So BF-Neural improves the accuracy by 5.32% over OH-SNAP and provides accuracies comparable to that of TAGE. ISL-TAGE with all its components delivers a MPKI of 2.39. For a 32KB storage budget, BF-Neural provides an accuracy of 2.73 MPKI [14].

Figure 9 demonstrates the contributions of individual optimizations to accuracy. The leftmost bar shows the accuracy achieved by a piecewise-linear-like conventional perceptron predictor [1] with a history length of 72. We choose this history length to fit in 64KB budget. The next three bars illustrate the breakdown of improvement in MPKI with BF-Neural predictor as we gradually apply the optimizations. All three bars use folded global history (*fhist*) to index the perceptron counters. The first among these three shows the accuracy obtained 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 3.28 to 2.67. Clearly traces such as SPEC02, SPEC06, SPEC09 and SERV3 benefit significantly from the sizable presence of biased branches (Figure 2). Interestingly, traces such as SPEC03, SPEC04, SPEC11, SPEC12 and SPEC18 achieve significant improvement in spite of very few biased branches (Figure 2). The benefit in those traces is primarily attributed to the *fhist* optimization that reduces aliasing significantly on the perceptron table entries associated with recent histories. The next bar reflects the improvement when BF-Neural predictor does not include biased branches in the global history register i.e. learn correlations only with non-biased branches. SPEC02, SPEC06, SPEC08, SPEC10, SPEC14, SPEC15, FP1, INT1, INT4 and MM3 clearly benefit from this due to the significant presence of biased branches. This optimization improves the MPKI from 2.67 to 2.59. The rightmost bar shows the improvement with the RS-like management policy for the global history register. Traces such as SPEC03, SPEC14 and SPEC18 that have few biased branches and hence can not reach very deep into the history with the last optimization; RS assists those to make up for that and proves to be the most valuable optimization. This optimization improves the MPKI from 2.59 to 2.49.

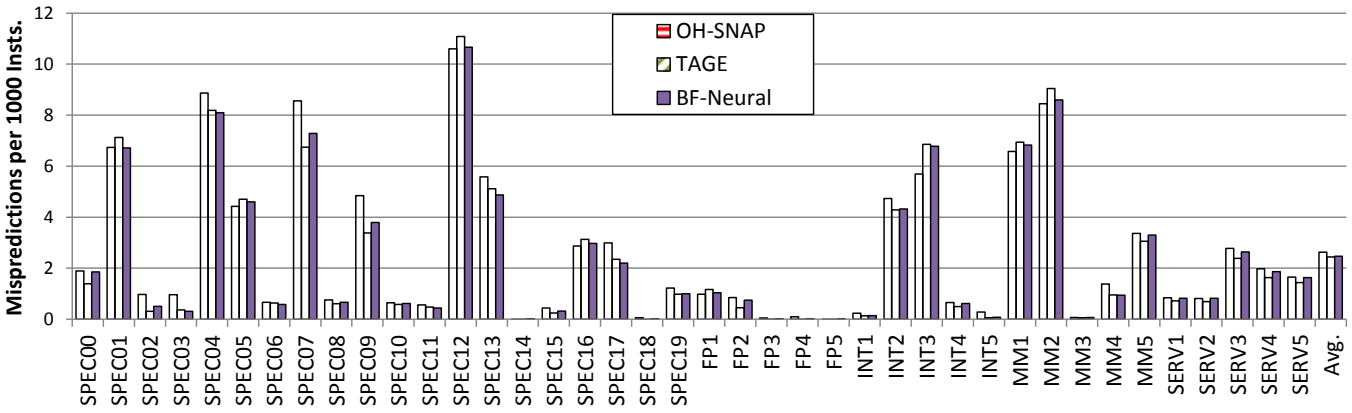


Figure 8: MPKI Comparison between Various Predictors.

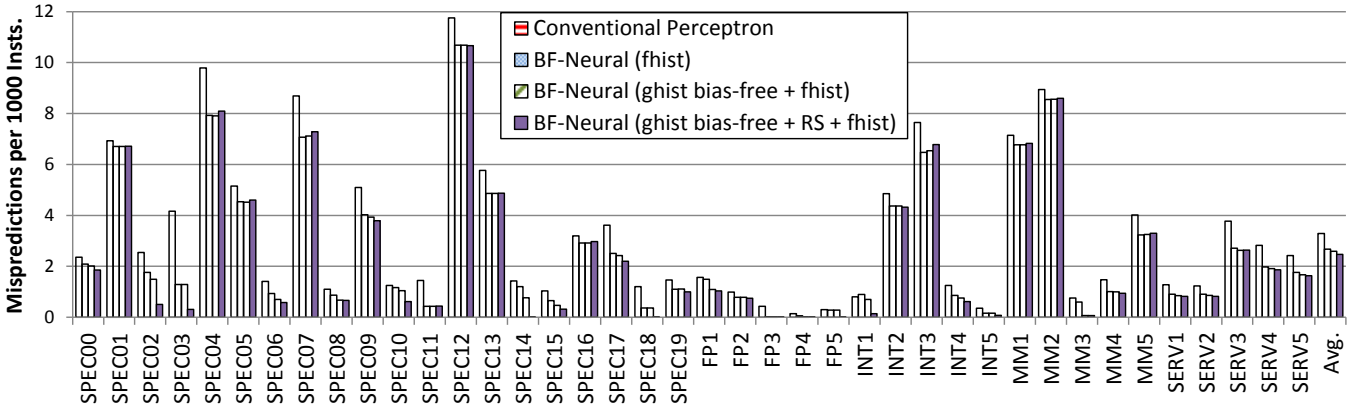


Figure 9: Contribution of Optimizations for the BF-Neural Predictor.

C. BF-TAGE Predictor

In order to minimize the perturbations caused by the dynamic detection of non-biased branches, BF-TAGE does not filter biased branches from the 16 recent global history bits. History segmentation divides the long global history into following non-overlapping segments such as {16, 32, 48, 64, 80, 104, 128, 192, 256, 320, 416, 512, 768, 1024, 1280, 1536, 2048}. The size of the per-segment RS considered is 8. The best set of history lengths found for a 10 tagged table BF-TAGE in our experiments is {3, 8, 14, 26, 40, 54, 70, 94, 118, 142}[2]. Clearly our BF-TAGE attempts to include older and richer context in 142 history BF-GHR.

Figure 10 compares the performance of our BF-TAGE against the baseline ISL-TAGE [4] for different number of tagged tables. BF-ISL-TAGE inherits the SC and the IUM components from the ISL-TAGE. BF-ISL-TAGE with 4 to 10 tagged tables in Figure 10 is sized to fit into the storage budget required in the baseline ISL-TAGE with corresponding number of tables. Each bar presents the accuracy in terms of arithmetic mean MPKI over all 40 traces.

[2]History lengths used by a 15 tagged table ISL-TAGE [4] is {3, 8, 12, 17, 33, 35, 67, 97, 138, 195, 330, 517, 1193, 1741, 1930}

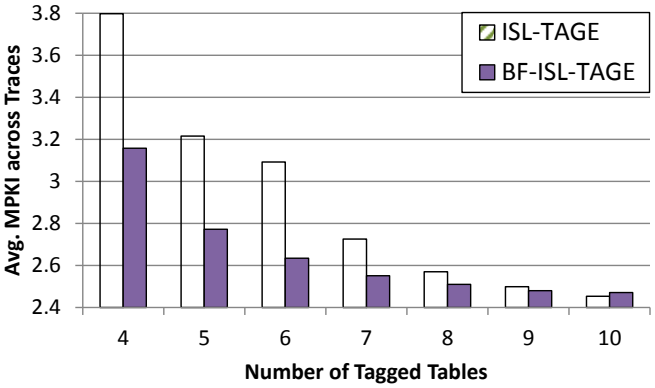


Figure 10: MPKI Comparison for Different Number of Tables.

For small to moderate number of tables, BF-TAGE consistently provides better accuracy. A BF-TAGE with 7 tagged tables achieves a MPKI of 2.57 in comparison to 2.73 obtained using conventional TAGE. Note that BF-TAGE and conventional TAGE both index the 7th tagged table using about 70 history bits. However BF-TAGE provides reasonable improvement with 7 tables, confirming the fact

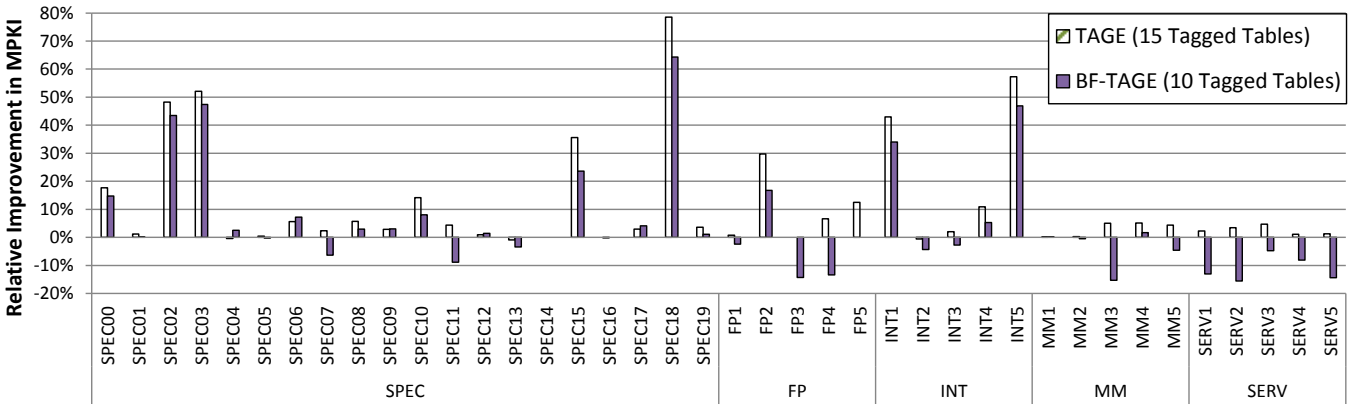


Figure 11: Relative Improvement in MPKI w.r.t a Conventional TAGE with 10 Tagged Tables.

that BF-GHR can offer much richer context to the TAGE predictor in contrast to the unfiltered history bits.

TAGE achieves best accuracy using 15 tagged tables [4]. However considering the fact that in recent mobile processors, such as ARM's Cortex-A15, branch predictor power accounts for $12-15\%$ of all core energy [15], a smaller TAGE with few tables is more likely to make inroads into a next-generation commercial processor. In view of the fact that BF-TAGE outperforms conventional TAGE significantly for small number of tables, it stands as a suitable choice for embedded and mobile processors.

Note that although BF-TAGE with 10 tagged tables closely matches the accuracy of baseline TAGE with 15 tagged tables for most of the long history-sensitive traces (subsection VI-D), however this does not result in the average MPKI improvement for BF-TAGE with 10 tables in Figure 10. This is primarily attributed to the poor performance of SPEC07, MM5 and the short traces, predominantly the MM and SERVER traces. The reason behind their poor performance is described in the subsection VI-D.

D. Reduced Number of Tagged Table Accesses Per Prediction

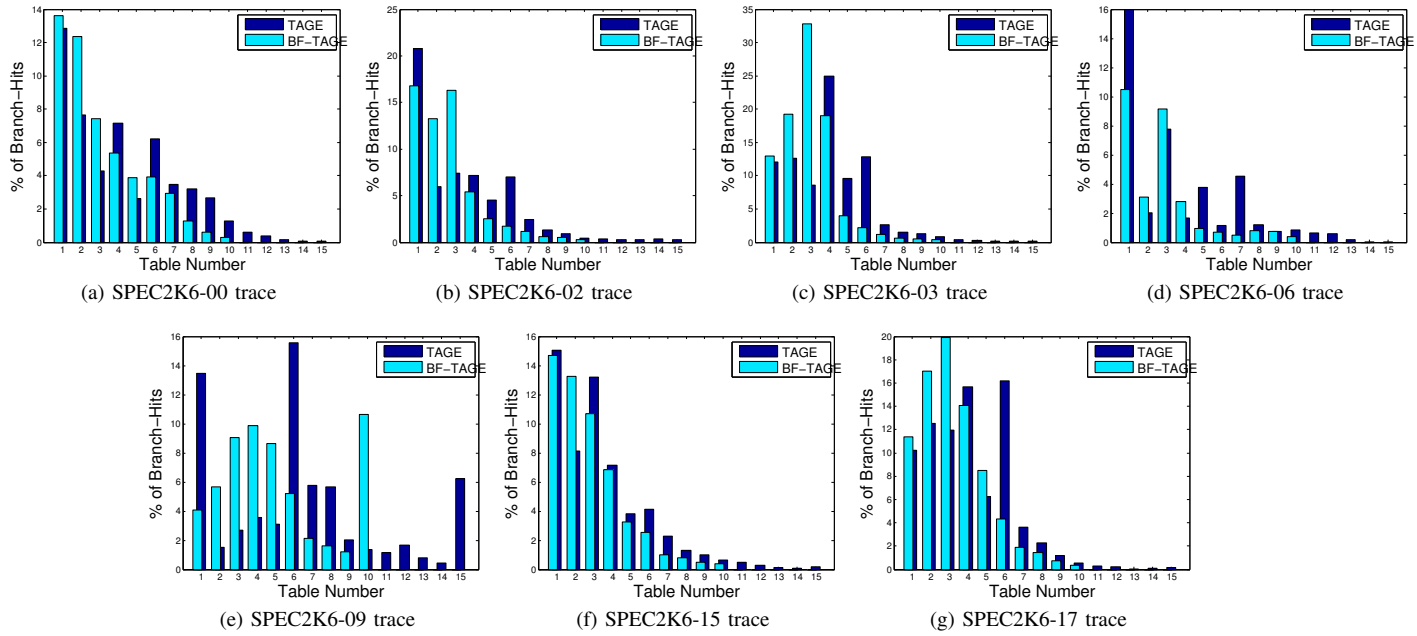
While analyzing the workloads with the 15 tagged table baseline TAGE [4], we observe that traces such as SPEC00, SPEC02, SPEC03, SPEC06, SPEC09, SPEC10, SPEC15, SPEC17, FP2, INT1, INT4 and INT5 exhibit a gradual decrease in mispredictions with increase in number of tagged tables from 10 to 15 i.e. increase in history lengths from 195 to 1930. SPEC05, SPEC08, SPEC11, SPEC19 and SERV3 observe marginal improvement with 15 tagged tables when compared to the 10 tagged table TAGE.

Figure 11 shows the incremental improvement in prediction accuracy that the baseline TAGE [4] with 15 tagged tables can obtain in comparison to 10 tagged tables. This figure also demonstrates the BG-TAGE predictor's potential to closely track that same accuracy but with only 10 tagged tables. Traces such as SPEC00, SPEC02, SPEC03, SPEC06, SPEC09, SPEC10, SPEC15, SPEC17, INT1, INT4 and INT5 closely match the accuracy of 15 tagged table TAGE with only 10 tagged tables of BF-TAGE.

Although SERVER traces have a large fraction of biased branches, they suffer significantly from the dynamic detection of non-biased branches throughout the execution. SERV3 trace suffers the most among them. A static profile-assisted classification of branches is found to improve the MPKI from 2.62 to 2.44 and restore the accuracy of SERV3 in the 10 tagged table BF-TAGE. FP1 and MM5 also suffer a marginal accuracy loss due to the same reason. The static profile-assisted detection was found to outperform the 15 tagged table TAGE on FP1 and MM5. On the other hand, the dynamic detection does not impair the accuracy of the long traces of the SPEC2006 workloads, as BF-TAGE gets enough time to recover the losses from this dynamic detection. Repeated executions of the short traces are found to achieve accuracy that are comparable to the TAGE with 15 tagged tables.

The performance drop in SPEC07 and FP2 with our 10 tagged table BF-TAGE in comparison to the 15 tagged table TAGE is attributed to only few branches in the trace. Those branches are intrinsically better predicted through the use of local history than through the use of a global history. The unfiltered history in 15 tagged table TAGE captures that useful local context interleaved with long global history. However recency-stack-like management policy fails to provide the useful local context that can assist in capturing different correlations for instances exhibiting opposite directions. As a result those instances repeatedly hash-conflict to the same set of entries causing mispredictions.

Figure 12 analyzes the BF-TAGE predictor's potential to achieve comparable accuracy to that of a 15 tagged table TAGE with fewer tagged tables. The set of traces exhibits almost similar number of mispredictions with a 15 tagged table TAGE and a 10 tagged table BF-TAGE. The bars represent the different tagged tables, illustrating the distribution of branches satisfied by each table. In other



(a) SPEC2K6-00 trace (b) SPEC2K6-02 trace (c) SPEC2K6-03 trace (d) SPEC2K6-06 trace

(e) SPEC2K6-09 trace (f) SPEC2K6-15 trace (g) SPEC2K6-17 trace

Figure 12: Histograms illustrating the shift in branch's distributions from longer-to-shorter-history tables.

Table I: Total storage for BF-TAGE with 10 tagged tables

Source	Quantity of bits
Kentries (T0-T10)	16, 2, 2, 2, 4, 4, 4, 2, 2, 1, 1
Tag width (T1-T10)	7, 7, 8, 9, 10, 11, 11, 13, 14, 15 bits
Storage (T0-T10)	2560, 2816, 2816, 3072, 6656, 7168, 7680, 3840, 4352, 2304, 2432 bytes
BST	8192 entries × 2-bits/entry = 2048 bytes
RS	142 entries × 16 bits/entry = 284 bytes
Unfiltered Hist.	1536 entries × (14 + 1 + 1) = 3072 bytes
	14-bit hashed PC, 1 bit T/NT, 1 bit bias status
Total	51100 bytes

words, they convey the percentage of branches that find a tag match in a table with the longest history. Clearly BF-TAGE observes the expected shift in the branch's distributions from longer-to-shorter-history tables. It validates our initial motivation and confirms that BF-TAGE can capture much older and much richer context with fewer tables.

Table I shows the storage budget used for BF-TAGE with 10 tagged tables. It requires virtually same storage to that of ISL-TAGE [4] with 10 tagged tables (51072 bytes without its Loop, SC and IUM components). Similarly BF-TAGE with 4 to 9 tagged tables in Figure 10 sizes the tagged tables appropriately to include the required RS and unfiltered history in the same storage as required by the baseline TAGE with corresponding number of tables.

VII. Related Work

There are decades of research in branch prediction [16]–[21]. The Filter predictor [22] uses the BTB to identify highly biased branches and prevent them from using the pattern history table (PHT), thus it reduces interference in the PHT. In contrast to that, Bias-Free predictor restricts the predictor from learning correlations with biased branches.

Two recent prior works [23], [24] attempt to identify phases (loop exit, return from function calls) in the program control flow with little correlation to prior branches and artificially modify the global history register to reduce or eliminate unnecessary noisy correlations. These approaches do not attempt to track distant branch correlations, rather reduces noisy correlations around loops and function calls. Nevertheless, Bias-Free predictor can benefit further by using these simple heuristics.

Thomas *et al.* [25] makes use of register-dataflow technique to identify correlated branches from a large global history. However, the associated hardware overhead and energy consumption make it impractical to scale beyond 64 past branches. Recent work on hashed perceptron [26] chooses few strided samples from the large global history to expand the effective reach of a fixed-length history register.

VIII. Conclusion

In this work, we propose Bias-Free predictor that is structured to learn correlations only with non-biased branches. The benefit of filtering biased branches is to expand the effective reach of a fixed-length history so that correlations from more distant branches can still be captured. The BF-Neural predictor is a conceptual design that demonstrates the potential of this bias-free approach. In this work, we present a practical implementation of this idea by applying it to the TAGE predictor. A detailed pipelined implementation

of BF-Neural is left for future work. That implementation will utilize the ahead-pipelining technique as proposed in [1] in conjunction with not including the branch PC in row index computation.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their insightful feedback which has improved the content and presentation of this paper. 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 and C. Lin, "Dynamic Branch Prediction with Perceptrons," in *International Symposium on High Performance Computer Architecture*, January 2001.

[3] D. A. Jimenez, "Fast Path-Based Neural Branch Prediction," in *International Symposium on Microarchitecture*, December 2003.

[4] A. Seznec, "A 64 Kbytes ISL-TAGE Branch Predictor," in *The 3rd Championship Branch Prediction, http://www.jilp.org/jwac-2*, June 2011.

[5] ——, "A 256 Kbits L-TAGE Branch Predictor," in *The 2nd Championship Branch Prediction*, May 2007.

[6] ——, "A New Case for the TAGE Branch Predictor," in *International Symposium on Microarchitecture*, December 2011.

[7] ——, "Storage-Free Confidence Estimation for the TAGE Branch Predictor," in *International Symposium High Performance Computer Architecture*, February 2011.

[8] D. A. Jimenez, "An Optimized Scaled Neural Branch Predictor," in *International Conference on Computer Design*, October 2011.

[9] "The Championship Branch Prediction (CBP-4)," in *The Journal of Instruction-Level Parallelism 4th JILP Workshop on Computer Architecture Competitions (JWAC-4), http://www.jilp.org/cbp2014/*, June 2014.

[10] N. Riley and C. Zilles, "Probabilistic Counter Updates for Predictor Hysteresis and Stratification," in *International Symposium on High-Performance Computer Architecture*, February 2006.

[11] A. Seznec, "Analysis of the O-GEometric History Length Branch Predictor," in *International Symposium on Computer Architecture*, June 2005.

[12] G. H. Loh, "A Simple Divide-and-Conquer Approach for Neural-Class Branch Prediction," in *International Conference on Parallel Architectures and Compilation Techniques*, September 2005.

[13] "The Championship Branch Prediction (CBP-3)," in *The Journal of Instruction-Level Parallelism 2nd JILP Workshop on Computer Architecture Competitions (JWAC-2), http://www.jilp.org/jwac-2/*, June 2011.

[14] D. Gope and M. H. Lipasti, "Bias-Free Neural Predictor," in *The 4th Championship Branch Prediction, http://www.jilp.org/cbp2014*, June 2014.

[15] "NVIDIA Tegra 4 Family CPU Architecture, NVIDIA, Tech. Rep., 2013. [Online]. Available: http://www.nvidia.com/docs/IO/116757/NVIDIA_Quad_a15_whitepaper_FINALv2.pdf."

[16] A. N. Eden and T. Mudge, "The YAGS Branch Predictor," in *International Symposium on Microarchitecture*, December 1998.

[17] M. Evers, P. Y. Chang, and Y. N. Patt, "Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches," in *International Symposium on Computer Architecture*, May 1996.

[18] P. Michaud, A. Seznec, and R. Uhlig, "Trading Conflict and Capacity Aliasing in Conditional Branch Predictors," in *International Symposium on Computer Architecture*, June 1997.

[19] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design Tradeoffs for the EV8 Branch Predictor," in *International Symposium on Computer Architecture*, June 2002.

[20] G. H. Loh and D. S. Henry, "Predicting Conditional Branches with Fusion-based Hybrid Predictors," in *International Conference on Parallel Architectures and Compilation Techniques*, September 2002.

[21] P. Michaud, "A PPM-like, Tag-based Predictor," in *The Journal of Instruction-Level Parallelism, http://www.jilp.org/vol7*, April 2005.

[22] P. Y. Chang, M. Evers, and Y. Patt, "Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference," in *International Conference on Parallel Architectures and Compilation Techniques*, October 1996.

[23] L. Porter and D. M. Tullsen, "Creating Artificial Global History to Improve Branch Prediction Accuracy," in *International Conference on Supercomputing*, June 2009.

[24] Z. Xie, D. Tong, and X. Cheng, "An Energy-Efficient Branch Prediction Technique via Global-History Noise Reduction," in *International Symposium on Low Power Electronics and Design*, September 2013.

[25] R. Thomas, M. Franklin, C. Wilkerson, and J. Stark, "Improving Branch Prediction by Dynamic Dataflow-based Identification of Correlated Branches from a Large Global History," in *International Symposium on Computer Architecture*, May 2003.

[26] D. A. Jimenez, "Strided Sampling Hashed Perceptron Predictor," in *The 4th Championship Branch Prediction, http://www.jilp.org/cbp2014*, June 2014.