An Energy-Efficient Branch Prediction with Grouped Global History

Mingkai Huang\textsuperscript{1}, Dan He\textsuperscript{1}, Xianhua Liu\textsuperscript{1}, Mingxing Tan\textsuperscript{2}, and Xu Cheng\textsuperscript{1}

\textsuperscript{1} Microprocessor Research & Development Center, Peking University, Beijing, China
Email: \{huangmingkai, hedan, liuxianhua, chengxu\}@mprc.pku.edu.cn
\textsuperscript{2} School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
Email: mingxing.tan@cornell.edu

\textbf{Abstract}—Branch prediction has been playing an increasingly important role in improving the performance and energy efficiency for modern microprocessors. The state-of-the-art branch predictors, such as the perceptron and TAGE predictors, leverage novel prediction algorithms to explore longer branch history for higher prediction accuracy. We observe that as the branch history is becoming longer, the efficiency of global history is degraded by the interference of different branch instructions.

In order to mitigate the excessive influence of the branch history interference, we propose the Grouped Global History (GGH) based branch predictor, a lightweight yet efficient branch predictor. Unlike existing branch predictors that make use of a unified global history for prediction, GGH divides the global history into a set of subgroups such that the interference resulted by frequently executed branch instructions could be restricted. With subgroups of global history, GGH also enables us to track even longer effective branch correlation without introducing hardware storage overhead. Our experimental results based on SPEC CINT 2006 workloads demonstrate that our approach can significantly reduce the branch mispredictions per kilo instructions (MPKI) by 4.76 over the baseline perceptron predictor, with a simple control logic extension.

\textbf{I. INTRODUCTION}

Energy efficiency has become a first-order design constraint of modern integrated circuits because of diminished benefits of technology scaling [1]. As modern microprocessors usually employ branch prediction and speculative execution to exploit instruction-level parallelism, a branch misprediction will invalidate all speculative instructions and even flush the entire pipeline, resulting in significant degradation of performance and energy efficiency. Therefore, improving the accuracy of branch prediction has been becoming increasingly important for modern microprocessors in achieving high performance and energy efficiency.

Branch prediction has been studied extensively in the last three decades [2], [3], [4], [5]. Among various previous branch predictors, two-level branch predictor [3] has been recognized as an important seminal work and also the most commonly used branch predictor in high-performance microprocessors. Two-level branch predictor makes use of the correlation between different branch instructions to predict the branch direction (i.e., \textit{taken} or \textit{untaken}). Such branch correlation serves as the foundation of all history-based dynamic branch predictors. Following this idea, there have been a large number of research on improving the efficiency of branch history. For example, [6] studied different types of branch history including the global branch history and local branch history; [7] proposed to combine the global branch history and local branch history to improve branch prediction accuracy. Recent studies [8], [9] have demonstrated that longer branch history can potentially improve the accuracy of branch prediction by tracking more distant branch correlations. Unfortunately, longer branch history tends to cause an exponential increase in the hardware storage of most two-level branch predictor alternatives, and thus hinders its adoption in conventional branch predictors.

Recently, a number of novel branch predictors, including the state-of-the-art perceptron predictor [4] and TAGE predictor [5], are proposed to pursue longer branch history for higher branch predictor accuracy. Inspired by the widely used perceptron machine learning algorithm, the perceptron predictor makes use of the weighted sum of global history to predict the direction of each branch instruction, and is capable of tracking longer branch history with linear increased hardware storage overhead. Similarly, the TAGE predictor features partial tagged predictor components indexed with distinct history lengths forming a geometric series. The TAGE predictor is able to track very-long-distance history with logarithmic hardware storage increase, and thus enables significantly higher prediction accuracy. Besides the novel training and prediction algorithm, both the perceptron predictor and the TAGE predictor take the advantage of longer branch history to improve the prediction accuracy [9]. In particular, the perceptron predictor can achieve much higher prediction accuracy with relatively simple hardware implementation, thus has been implemented in recent commercial microprocessors such as AMD Piledriver [11].

We observe that perceptron predictor is still fundamentally restricted by the interference of the branch history for different branch instructions. In particular, the history of the frequently executed branches in a loop will flush the useful history of the branches outside, which would significantly degrade the efficiency of the longer global history, and thus decrease the prediction accuracy. In this paper, we introduce the novel Grouped Global History (GGH) to mitigate the
excessive influence of the branch history interference. Unlike existing branch predictors that make use of a unified global history for prediction, GGH divides the global history into a number of subgroups such that the interference resulted by frequently executed branch instructions will be restricted. By avoiding flushing effective branch history, GGH further enables us to track longer effective branch correlation without introducing additional hardware storage. In general, our approach can be combined with any global history based branch predictors. In this paper, we evaluate our approach based on the state-of-the-art perceptron branch predictor using SPEC CINT 2006. Experimental results demonstrate that our approach can significantly reduce the branch mispredictions per kilo instructions (MPKI) from 18.11 to 13.35, a reduction of 4.76, and improve the performance by 5.20% over the state-of-the-art perceptron predictor, without large storage and complex logic.

The primary contributions of this work include:

1) We systematically study the efficiency of distant global history in branch prediction. Our study demonstrates that longer branch history tends to offer opportunities to achieve higher prediction accuracy, but branch history interference can greatly impact the prediction accuracy in the presence of long branch history.

2) We propose a novel branch predictor, GGH perceptron predictor, to effectively capture distant global history by mitigating the branch history interference of different branch instructions.

3) We demonstrate that our proposed branch predictor can significantly reduce MPKI and improve the performance by minimizing the interference.

The rest of the paper is structured as follows: Section II provides the motivation for our GGH perceptron predictor; Section III describes the hardware structure and cost of GGH perceptron predictor; Section IV presents our experimental methodology; Section V demonstrates the experimental results; Section VI reviews the previous work, followed by conclusions in Section VII.

II. MOTIVATION

Global history based branch correlation has been serving as the foundation for most of modern dynamic branch predictors [3], [4], [5]. Recent state-of-the-art branch predictors, such as the perceptron predictor [4] and the TAGE predictor [5], tend to make use of longer global history to track long-distance branch correlation by employing novel branch prediction algorithms. However, we observe that the efficiency of the global history is inherently limited by the interference between different branch instructions. Especially, a frequently executed branch instruction may flush all useful history information from other instructions, and thus significantly degrading the efficiency of the global history.

The literature shows that longer history brings better performance [4], [5], [8], [9], but the work [16] shows that the outcomes of only a very small number of previous branches are needed to make an accurate branch prediction, as long as the most important branches can be identified. However, sometimes the useful history bits are far away from the latest history bit, i.e., useful history bits are flushed away by other useless history bits.

Figure 1 shows a code snippet with six different branch instructions (A – F) from 403.gcc in SPEC CINT 2006. In this example, Branch C has strong correlation with Branch A. Unfortunately, before Branch C is executed, the loop repeatedly executes Branch D and thus flushes the history of Branch A. Without the history of Branch A, Branch C becomes hard to predict.

Figure 2(a) illustrates the detailed interference of branch history for this example. Here, we use A0, C1, D4 and F1 to denote the corresponding history bits of Branch A, C, D and F. For the sake of simplicity, we assume a 4-bit global history register is used to track history information. Before D4 is committed, A0 is captured by the global history register; however, after D4 is committed, it will be shifted into the global history register and A0 will be evicted out of the global history register. We observe that such interference brings two problems: (1) Useful branch history (e.g. A0) is flushed out because of the interference, resulting in low prediction accuracy for the depending branch (e.g. Branch C); (2) The limited global history register is occupied by useless history information from the same Branch D, which reduces the efficiency of the branch history information. These two problems are inherent for unified global history,
and most of existing global history based branch predictors, such as perceptron or TAGE predictor suffer from these problems.

To reduce the inherent interference in conventional unified global history, we propose to divide the global history into subgroups. Branch history in different groups is independent, and each new branch history can only evict the history in the same group. Figure 2(b) illustrates our idea of Grouped Global History (GGH) for the same example shown in Figure 1. We divide the branches into four groups based on the lower-order bits of their instruction addresses, e.g., A/B, the lower-order bits of which are 00, are in Group 00; C, the lower-order bits of which are 01, is in Group 01; D, the lower-order bits of which are 10, is in Group 10, and E/F, the lower-order bits of which are 11, are in Group 11. Each group stores only 1-bit history, so the total storage is the same to the previous global history register. As Branch A and D are in different groups, there is no interference between them. Therefore, when D4 is committed, it only evicts the history D3 in the same group. For each branch prediction, we concatenate the stored history bits from different groups to form the final history, and apply the same prediction algorithm as the original predictor using this grouped history information. As shown in Figure 2(b), the grouped history information consists of A0, C0, D4, F0 from four different groups after D4 is committed. By dividing the unified global history into subgroups, we are able to: (1) keep the useful history A0 in the history regardless how many times Branch D is executed, so Branch C can be predicted with high accuracy; (2) remove useless history D2 and D3 from the global history, and thus improving the efficiency of the global history.

In addition, our GGH approach can also capture more distant history by keeping long-distance branch history information (e.g., A0) in the limited history storage. In Figure 2(a), the underlined history bits in global history sequence are the history bits kept by GGH. We use equivalent global history length to denote the max distance from these history bits to current branch. As shown in this figure, our GGH approach can effectively increase the equivalent global history length compared to conventional unified global history approach. In fact, we have evaluated the average equivalent global history length of GGH using SPEC CINT 2006 as shown in Figure 3. Assuming the total length of history is 32 bits, we study different configurations for number_of_groups \times bits_per_group = 32, where number_of_groups varies in 2, 4, 8, 16 and 32. Clearly, as the number of groups increases, the equivalent global history length dramatically increases. As pointed out by previous study [9], such longer equivalent global history length usually leads to higher prediction accuracy.

### III. GROUPED GLOBAL HISTORY PERCEPTRON PREDICTOR

In this section, we will describe our GGH approach by combining it with a state-of-the-art Perceptron predictor. However, it is worth mentioning that our approach can be also combined with any other global history-based branch predictors, such as O-GEHL [20] and TAGE predictor [21].

#### A. The baseline Perceptron Predictor

The perceptron predictor [4] uses one of the simplest possible neural networks, the perceptron, to learn positive and negative correlations between branch outcomes in the global history and the branch being predicted. In the perceptron predictor, each branch instruction address is mapped to a single perceptron. A perceptron is a vector whose elements are the weights. The output is the dot product of the weights vector, \( w_0...n \), and the input vector, \( x_0...n \) (\( x_0 \) is always set to 1, providing a bias input). The inputs to perceptrons are decided by the branch history, that is, each \( x_i \) is either -1, meaning untaken or 1, meaning taken. The output \( y \) of a perceptron is computed as the following formula:

\[
y = w_0 + \sum_{i=1}^{n} x_i w_i
\]

A negative output is interpreted as predicting untaken. A non-negative output is interpreted as predicting taken.

Once the branch outcome has been computed, the following algorithm is used to train the perceptron. Let \( t \) be -1 if the branch was untaken, or 1 if it was taken, and let \( \theta \) be the threshold, a parameter to the training algorithm used to decide when enough training has been done.

\[
\text{if } \text{sign}(y_{out}) \neq t \text{ or } |y_{out}| \leq \theta \text{ then}
\]

\[
\text{for } i := 0 \text{ to } n \text{ do}
\]

\[
w_i := w_i + tx_i
\]

\[
\text{end for}
\]

\[
\text{end if}
\]

Since \( t \) and \( x_i \) are always either -1 or 1, this algorithm increments the \( i^{th} \) weight when the branch outcome agrees with \( x_i \), and decrements the weight when it disagrees.
Intuitively, when there is mostly agreement, i.e., positive correlation, the weight becomes large. When there is mostly disagreement, i.e., negative correlation, the weight becomes negative with large magnitude. In both cases, the weight has a large influence on the prediction. When there is weak correlation, the weight remains close to 0 and contributes little to the output of the perceptron.

B. GGH Structure

GGH is the concatenation of several groups of global history divided in accordance with the lower-order bits of branch instruction addresses. Each group of GGH keeps the local information, and all the groups form the history containing the global information. This technique derives from our consideration of branch correlation, which comes from the correlation between static branches [16]. Actually, the latest history often contributes most to the prediction, so we hope to store only the latest several history bits of every branch. However, this is unrealistic. For feasibility, we use only the lower-order bits of branch addresses, forming only several groups. The evaluation results in Section V-C show that more groups may worsen the performance, so the performance of storing the latest history bits of every branch may be poor because of too much noise. The history kept by this group way is more efficient than that of global history, in which too much history of frequently executed branches may flush the useful latest history of rarely executed branches.

We argument the baseline Perceptron predictor with our GGH approach, i.e., dividing the global history into subgroups, to reduce the interference between history bits of different branches. Basically, the global history is grouped by the lower-order bits of branch instruction addresses, in which way the most recent history bits of branch groups are kept. Figure 4 illustrates the table storing GGH. Each entry contains the history of branches with the same lower-order bits of branch instruction addresses, and all the history bits in the entries are concatenated to form the final history. The concatenation order is not important, because all the groups are equal. In GGH, the history of a branch occupies one entire entry at most. From the positive perspective, the most useful part of history is retained. Nonetheless, from the negative perspective, it limits the local information kept, so GGH may be worse than global history and local history in some particular cases.

The number of bits per entry multiplied by the number of entries is the total length of history. Once the total length of history is fixed, the number of bits per entry and the number of entries can be adjusted. When more entries are used, the history conflicts of different branches less occur, but the local information of each branch group is more limited. When fewer entries are used, more local information of each branch group can be kept, but the history conflicts will increase. If there is only one entry, GGH fades to global history. The best configuration, which has the best balance for the common situation, can be obtained by the experiment. If different situation can utilize different configuration, more specialized configuration can be adopted to obtain better performance. The configurable ability shows the flexibility of GGH.

The GGH table is much like local Branch History Table (BHT). However, local history uses only one entry of local BHT, while GGH uses all the entries, so the information contained in GGH is global. The storage cost of GGH table is equal to that of global Branch History Register (BHR), which is much smaller than that of local BHT.

C. GGH-Based Perceptron Predictor

While applying GGH to the perceptron predictor to reduce the history interference, we just replace the global history by GGH, forming Grouped Global History Perceptron (GGHP) predictor. The overall structure of GGHP is showed in Figure 5. It mainly contains three components:

- The branch history is on the left side. It contains the previous branch direction outcomes. This part of GGHP is our primary focus. In comparison with the original perceptron predictor, GGH is used instead of global history.
- The branch queue is in the middle. It contains some information, such as the used GGH and branch address, which is needed in the updating process when the branch outcome is obtained. This part is the same as the original perceptron predictor.
- The table of perceptron weights is on the right side. It contains a series of perceptrons, each of which includes several weights. The perceptron weights express the correlations between the branch history and the branch direction being predicted. This part is also the same as the original perceptron predictor.

The predicting and updating process of GGHP is much like the original perceptron predictor, except the logic needed to deal with the history.

The predicting process is illustrated using the solid line with an arrowhead in Figure 5. During the predicting process, the history bits in all the entries of GGH table are read out. They are concatenated to form a history series. The corresponding weights indexed by the lower-order bits of the branch instruction address are read out of the table of
perceptron weights. Then the sum of history bits weighted by the perceptron weights is computed. The sign of the computation result decides the prediction of current branch direction. If the result is negative, the prediction is untaken. Otherwise, the prediction is taken. The formed history is also pushed into the branch queue, and then used in the updating stage.

The updating process is illustrated using the dash line with an arrowhead in Figure 5. During the updating process, the branch outcome, the selected perceptron weights and the recorded GGH are used to compute the new perceptron weights which are stored back later. At the same time, the branch outcome is shifted in the indexed entry in GGH table. If the prediction is wrong, the instructions following this branch are flushed, and the instruction fetch restarts.

D. Hardware Cost

The original perceptron predictor contains a table of perceptron weights, a global BHR, a branch queue, and some computing and control logic. Compared to the original perceptron predictor, GGHP only revises the history information used, so we only discuss the hardware cost and latency of history. When using GGH, only the global BHR is replaced with the GGH table and accessing logic of GGH table is added. Because we control the total length of GGH to the same as that of global history, so the storage cost of GGH table is the same as that of global BHR, and no extra storage is needed. Assuming 32-bit history is used, both global BHR and GGH table cost 32-bit storage. When predicting, all history bits are read without extra delay. When updating, addressing GGH table brings about some extra latency. Fortunately, updating is not on the critical path, and as the table is small, the latency is short.

E. Hybrid History

GGH partitions the history storage to store different branch history groups, so it eliminates the interference between groups, but at the same time, it has potential drawbacks that it shortens the maximum storage each group can occupy, so the performance of GGH may be worse than that of global history and local history in some conditions. Hence, we present several history mixture strategies. We consider mixing in global history or local history. The mixture ways include XOR and concatenation. Concatenation causes no information confusion, but leads to shorter history per type. XOR does not shorten the history length per type, but gives rise to information confusion.

While using the hybrid history, extra hardware components are required, such as global BHR or local BHT, as well as the hash function unit. This will lead to more hardware storage cost. For example, supposing 32-bit history is used, when using GGH, 32-bit storage is needed, but when using concatenation of 16-bit GGH and 16-bit local history indexed by 10-bit address, $16 + 16 \times 2^{10} = 16400$ bits storage is needed.

The hybrid history brings extra hardware cost and complexity, but if it provides significant benefit, the extra hardware cost and complexity are worth affording. If the hybrid history provides little performance improvement, the advantage of GGH is further highlighted. In Section V-D, we will evaluate which situation is true.

IV. EXPERIMENTAL METHODOLOGY

We employ gem5 [12] running in x86_64 Out-of-Order full-system mode to evaluate GGHP. We use the original perceptron predictor as the base branch predictor, which we call Base. Table I shows the parameters of processors using Base and GGHP. To simplify the implementation, the latencies of Base and GGHP are set to one cycle, as both branch predictors have the same latency. The total length of history is 32 bits and a 1K-entry perceptron table is used. Firstly, we evaluate the perceptron predictor with only GGH. Later, we compare different mixtures of GGH, global history and local history. Our workload is SPEC CINT 2006 with the reference input. Each benchmark runs 100M instructions.

V. RESULTS AND ANALYSIS

A. Basic Results

When the total length of history is fixed, we adjust the number of entries and the number of bits per entry to get different configurations. It is an extreme that each entry contains only one bit in which way the utilization of local information is the most limited. Another extreme is that only one entry is used, which is exactly global history. We keep the total length of history to 32 bits and evaluate the effects of GGH. Figure 6(a) compares MPKI of Base with GGHP of different configurations. All configurations reduce the average MPKI compared to Base. When using one bit
per entry, GGH reduces MPKI for SPEC CINT 2006 from 18.11 to 13.96, a reduction of 4.15. When the configuration is between the two extremes, better result is achieved. Perhaps the intermediate configuration has a better trade-off of reducing interference between groups and keeping longer history per group. On average, 8×4 is the best configuration, which decreases the MPKI by 4.76. The best configurations of different benchmarks are not the same, showing that different benchmarks have different characteristics.

The MPKI variation of h264 program is different from other benchmarks. When using the 32×1 configuration, the MPKI is increased by 1.10. The benchmark h264 is a video compression program implementing H.264 standard, which contains lots of loops. The branch prediction for loops needs a long local history. The 32×1 configuration limits the recorded local history, thus hurting the branch prediction. When fewer groups are used, longer local history is kept, so the negative impact becomes weaker. When using the 4×8 configuration, the MPKI is reduced by 0.28. In Section V-D, we evaluate the hybrid history. Mixing local history and GGH improves the performance of h264 a little more, which verifies our explanation.

Figure 6(b) shows the branch prediction miss rate comparison. On average, the 8×4 configuration reduces the miss rate from 10.14% to 7.02%, a reduction of 30.76% compared to Base. Figure 6(c) shows the performance improvement. Corresponding to the MPKI and miss rate reduction, the 8×4 configuration improves the performance of Base by 5.20%.

B. Energy

GGHP improves the performance of Base while it needs little extra hardware cost and complexity, so it should reduce the energy consumption of the branch predictor. We present the energy result in this section. The basic power parameters of the branch predictor are derived from the CACTI-6.5 tool [13] using the 32nm technology. The detailed comparisons between GGHP and Base are illustrated in Figure 7. On average, the 8×4 configuration reduces the energy consumption of the branch predictor by 4.71% over Base, which is similar to the performance improvement.

As the power of the overall processor is hard to model, we do not evaluate the total energy impact of GGHP on the processor. Nonetheless, we make no change to other parts of the processor, so the energy impact on other parts of the processor may be similar to the performance improvement. As a result, the energy impact of GGHP on the whole processor may also be similar to the performance improvement.

C. Length Variation

In order to explore the potential of GGH, we compare the performance variation as the history length varies between 16 bits and 128 bits. The configurations include GH (Global History), 2-entry, 4-entry, 8-entry and 1-bit (one bit per entry). Among them, we care more about GH, 8-entry and one bit per entry. GH is the history used in our base predictor, one bit per entry is the extreme configuration and 8-entry uses the number of entries which performs the best in the results of Section V-A. The results are showed in Figure 8. The performance is the best when the history length reaches some degree, beyond which the performance gets worse. We believe that distant history has less information, and may confuse useful information if not suitably utilized. The best lengths of GH and one bit per entry are 32 bits, and the best length of 8-entry is 64 bits. When using 64-bit history, the advantage of 8-entry over GH is larger. GH cannot properly utilize the extra history bits, so the extra history bits do more harms to GH than goods. However, longer history brings longer local history for 8-entry, which supplements the weakness of GGH, so the performance can be further improved. The drop trend of one bit per entry is obvious than GH. When too many entries are used, some of the entries keep stable, so the history bits in them act...
like bias history bits and outweigh other important history bits. This result suggests us that too many entries are not appropriate.

D. Hybrid History

According to the analysis in Section III-E, we see that global history, local history and GGH have different advantages, and are suitable for different situations. If we mix them, we may get both advantages to reach better performance. Therefore, we evaluate several configurations of the hybrid history, showed in Table II. G means global history, L means local history, P means GGH, - means concatenation, and X means XOR. We still fix the history length to 32 bits. In order to facilitate adjusting the configuration of the GGH part in the hybrid history, when concatenation is used, two 16-bit history parts are used. We evaluate various configurations of the GGH part in the hybrid history, and the configurations in Table II are the best.

The experimental results in Figure 9 show that the effect of XOR is bad. Maybe it is because perceptron weight is suitable for the history containing single meaning, but XOR mixes in multiple history. Concatenation can bring little performance improvement. The best configuration is LP, which improves the performance by 0.22% over 8×4. The improvement is negligible. Considering the hybrid history brings extra hardware cost and complexity, GGH is a good trade-off between simplicity and efficiency. With the help of local history, LP is the best configuration of h264. It improves the performance of h264 by 1.64% over global

**Figure 6.** Comparison between Base and GGHP.
Table II: Hybrid History Configuration

<table>
<thead>
<tr>
<th>Name</th>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>16G-16×1P</td>
<td>16-bit global history are concatenated with 16×1 GGH</td>
</tr>
<tr>
<td>LP</td>
<td>16L-8×2P</td>
<td>16-bit local history are concatenated with 8×2 GGH</td>
</tr>
<tr>
<td>GLXP (16G+16L)-8×2P</td>
<td>16-bit global history are XORed with 16-bit local history and then concatenated with 8×2 GGH</td>
<td></td>
</tr>
<tr>
<td>GPX</td>
<td>32G×8×4P</td>
<td>32-bit global history are XORed with 8×4 GGH</td>
</tr>
<tr>
<td>LPX</td>
<td>32L×32×1P</td>
<td>32-bit local history are XORed with 32×1 GGH</td>
</tr>
<tr>
<td>GLPX (32G×32L×2×16P)</td>
<td>32-bit global history, 32-bit local history and 2×16 GGH are XORed together</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>32L</td>
<td>32-bit local history</td>
</tr>
<tr>
<td>GL</td>
<td>16G-16L</td>
<td>16-bit global history are concatenated with 16-bit local history</td>
</tr>
</tbody>
</table>

This verifies our speculation in Section V-A. In Figure 9, we also show the evaluation result of L and GL [7]. GL is better than L and outperforms Base by 4.33%. Compared to 8×4, GL has less performance improvement and cost a large local BHT, so 8×4 is more efficient.

E. Comparison with the State-of-the-Art

In order to make our experiments more comprehensive, we compare GGHP with the state-of-the-art TAGE predictor from [5]. The performance results in Figure 10 show that the TAGE predictor outperforms the original perceptron predictor by 7.83%. The excellent performance of the TAGE predictor comes from that it can capture the long-distance branch correlation. Owing to GGH, the performance gap between the perceptron predictor and the TAGE predictor is narrowed from 7.83% to 2.63%.

VI. RELATED WORK

Branch prediction is a key technique of modern multi-issue multi-stage microprocessors. James E. Smith [2] proposed bimodal, which is based on a finite state machine. Considering the correlation between branches, researchers developed the two-level branch predictor [14], [15], [3], which utilizes the history of previously executed branch to predict the direction of current branch. The gshare predictor [3] is the most representative one. It usually tracks short distance correlation using 10–15 bits global BHR. Marius Evers, Sanjay J. Patel, Robert S. Chappell, Yale N. Patt [16] analyzed the source of such correlation in source code, pointing out that a branch may depend on other branches or the previous outcomes of the branch itself. Global history and local history [6] are commonly used history information. In order to reduce the interference between history information, researchers proposed Bi-Mode [17], YAGS [18] et al. Kevin Skadron, Margaret Martonosi, Douglas W. Clark [7] proposed alloying global history and local history together in a two-level branch predictor structure to attack wrong-history mispredictions, while we present hybrid history in Section III-E and Section V-D, which alloying global history, local history and GGH together.

Dynamic branch prediction relies on the branch history. The longer branch history is stored, the more information is kept, resulting in a higher hit rate. However, longer branch history means larger storage cost. Therefore, new techniques need to be developed to utilize longer branch history.

Some works utilize longer history by changing the history utilizing strategy, such as the perceptron predictor [4], [19], the O-GEHL predictor [20], and the TAGE predictor [21], [5]. The perceptron predictor uses perceptrons to train the correlation between branches. The storage cost of perceptron predictor is linear in the history length, so it can utilize longer history than the two-level branch predictor. In [4], the
perceptron predictor works best with a history length of 62 bits at 64KB hardware budget. The O-GEHL predictor and the TAGE predictor rely on several predictor components indexed with different history lengths forming a geometric series. The storage cost is logarithmic in the history length, so even longer history can be used, leading to higher performance of processors. In [5], a history length of up to 2000 bits is used in the TAGE predictor at 64KB hardware budget.

Other works [22], [23], [8], [9], [10], [24], [25], [26] focus on the efficiency of branch history. They filter less useful history to leave room for more useful history, noticing that the usefulness of different parts of history is different. Our work is in this domain. Po-Yung Chang, Marius Evers and Yale N. Patt [22] filter strong-biased branches, so as to avoid their influence to PHT. Recently, Dibakar Gope and Mikko H. Lipasti [25] filter completely biased branches, and the work is based on the more complex perceptron predictor and the TAGE predictor. Another technique proposed by [25] filters the older history of a static branch, which is much like our idea. The basic difference between [25] and our work is that they organize the history by the branch instruction address while we organize the history by lower-order bits of the branch instruction address to form groups. Because of this difference, [25] uses associative search for the branch instruction address, while we use table lookup, which is less complicated but more flexible. [25] stores only one history bit per static branch, while we can adjust the number of history bits per group more easily, which may lead to better performance according to the experimental results in Section V-A. Others [23], [8], [9], [24] noticed that the history in the loop or the function call is less useful for the branches behind them, and can be treated as a kind of pollution information. They increase the branch prediction hit rate by decreasing such kind of influence on successive prediction. [26] aims at utilizing only the critical history in the presence of nested loops.

While previous works deal with the interference inside the single thread, some works [27], [28], [29] focus on history interference between threads. [27] uses hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. [28] alleviates the destructive impact of user/kernel branch interference by separating the history of user code and kernel code. [29] aims at the history interference between short threads in the multi-core processors, and proposed technique that set the global history register of the spawned thread to the initial value of the program counter.

VII. CONCLUSION

In this paper, we systematically studied the efficiency of global history in the context of state-of-the-art branch predictors. Our study demonstrated that global history interference between different branch instructions is a key limitation in pursuing higher prediction accuracy. To overcome this limitation, we proposed a lightweight yet efficient approach, called GGH, by intelligently dividing the unified global history into a set of subgroups based on branch instruction addresses. We demonstrated that our approach can significantly reduce the interference and further enable the underlying predictor to track much longer effective history without adding additional hardware storage. Our experiments on SPEC CINT 2006 benchmark suites provide further evidence that our technique can effectively improve the branch prediction accuracy and achieve higher performance for a variety of workloads compared to the state-of-the-art perceptron predictor.

ACKNOWLEDGEMENTS

We are particularly grateful to Teacher Dong Tong for his great suggestions on this work. We would like to thank the anonymous reviewers for their constructive feedback. This work was partially supported by the National Science and Technology Major Project of China under grant 2009ZX01029-001-002 and National Science Fund of China under grant 6130004.
REFERENCES


