

# Adaptive Information Processing: An Effective Way to Improve Perceptron Branch Predictors

**Hongliang Gao**

**Huiyang Zhou**

*School of Computer Science  
University of Central Florida  
Orlando, Florida 32816-2362  
Voice (407) 823-5210  
FAX (407) 823-5419*

[HGAO@CS.UCF.EDU](mailto:HGAO@CS.UCF.EDU)

[ZHOU@CS.UCF.EDU](mailto:ZHOU@CS.UCF.EDU)

## Abstract

Perceptron branch predictors achieve high prediction accuracy by capturing correlation from very long histories. The required hardware, however, limits the history length to be explored practically. In this paper, an important observation is made that the perceptron weights can be used to estimate the strength of branch correlation. Based such an estimate, adaptive schemes are proposed to preprocess history information so that the input vector to a perceptron predictor contains only those history bits with the strongest correlation. In this way, a much larger history-information set can be explored effectively without increasing the size of perceptron predictors. For the distributed Championship Branch Prediction (CBP-1) traces, our proposed scheme achieves a 47% improvement over a g-share predictor of the same size<sup>1</sup>. For SPEC2000 benchmarks, our proposed scheme outperforms the g-share predictor by 35% on average.

## 1. Introduction

Given its great impact on performance, branch prediction has been extensively studied and various branch predictors have been proposed in the literature. In this paper, we attack the problem from a different perspective: *rather than devising another new prediction algorithm, we propose to process the inputs to branch predictors*. The insight behind our idea is that many existing branch prediction algorithms are very powerful to exploit correlation from their inputs such as local/global branch history, and many mispredictions are actually due to the lack of such correlation from the inputs to branch predictors. If the inputs can be configured to maximize the correlation, much higher prediction accuracy can be achieved without adding extra complexity onto the branch predictors.

As presented in [2], dynamic branch prediction schemes can be described using a generic conceptual system model, shown in Figure 1. The source information, including branch addresses (PC), local/global histories (LHR/GHR), along with other run-time information, is gathered when a program executes. The information processor extracts a subset of the source information and forms an information vector. Then, the predictor processes the information vector and makes a prediction. Traditionally, various Markov Finite State Machines (FSMs) are used as the predictor [2]. Recently, predictors based on perceptrons [5], [6] have been proposed and shown to have superior prediction accuracy to the widely used FSM-based predictors such as g-share [7] and two-level

---

<sup>1</sup> Our proposed scheme achieves the highest prediction accuracy for the un-disclosed CBP-1 traces and won the Champion of the first Championship Branch Prediction (CBP-1) contest [1].

predictors [9]. One main reason is that the cost of a perceptron predictor scales linearly rather than exponentially, thereby enabling it to explore correlation from much longer information vectors. In a recent study [8], the accuracy of perceptron predictors is further improved with the following extensions: using pseudo-tag to reduce aliasing impact, skewing perceptron weight tables to improve table utilization, and introducing redundant history to handle linearly inseparable data sets. The nonlinear redundant history also leads to a more efficient representation, Multiply-Add Contributions (MAC), of perceptron weights and a simpler hardware implementation [8].

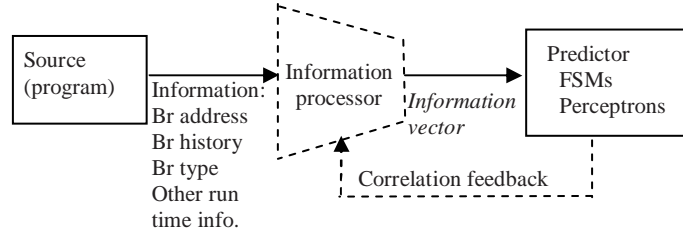


Figure 1: A conceptual system model for branch prediction [2].

Most branch predictors employ an implicit information processor, simply extracting some number of bits from GHR, LHR or PC. The novelty of this paper is a new correlation-based information processor, which extracts the information vector from information source adaptively based on the strength of branch correlation. As highlighted in Section 2, a key observation of our approach is that *perceptron weights can be used as a quantitative measure of correlation between a branch and the input information vector*. With such a measure, the information vector can then be re-assembled accordingly to maximize the branch correlation. Additionally, since the adaptation is performed at a coarse grain, e.g., at program phase boundaries or at an interval of a certain number of branches, the adaptation logic is *not* latency critical.

The remainder of the paper is organized as follows. A micro-benchmark is used in Section 2 to illustrate the correlation exploitation and motivate our main ideas. The proposed adaptive information processing schemes are contained in Section 3. Section 4 presents the overall predictor design including the special handling of loop and bias branches to minimize their adverse impact on the perceptron predictor. The experimental results using CBP-1 and SPEC2000 benchmarks are discussed in Section 5. Finally, Section 6 concludes the paper.

## 2. Exploiting Correlation with Perceptron Branch Predictors

Many branch predictors, including two-level and g-share predictors, achieve high prediction accuracy by exploiting correlation from history information. Perceptron predictors, moreover, have an additional capability to estimate the strength of such correlation, as illustrated from the example shown in Figure 2.

The micro-benchmark in Figure 2 contains four conditional branches ( $B1$ ,  $B2$ ,  $B3$ , and  $B4$ ):  $B1$  is determined by two random values,  $B2$  correlates with  $B1$  as both branches share the same random variable  $r1$ ,  $B3$  is simply the inverse of  $B1$ , and  $B4$  correlates with both  $B2$  and  $B3$  using an XOR function. To predict these four branches, a 4-entry perceptron predictor (i.e., each branch has its own perceptron) with 8-bit global branch history (i.e., an 8-bit GHR) [6] is used. After simulating 100M instructions, the perceptron weights ( $w1$ - $w8$ ) for these branches are reported and the misprediction rates are 50.09%, 25.45%, 0%, and 19.11% respectively for the branches  $B1$ - $B4$ .

Several important observations can be made from Figure 2. First, perceptron weights embody the correlation information and the absolute values of perceptron weights provide an estimate of the correlation strength. As the branch *B1* finds no correlation from previous branch history, it features with small random perceptron weights. The branch *B2* correlates to *B1*. So, its perceptron weights contain a high  $w_1$  (i.e., strong correlation with GHR[0]) and small random  $w_2$ - $w_8$ . The branch *B3* shares the same condition (inverse) as *B1*. Therefore, it has a singular large weight,  $w_2$  (i.e., strong correlation with GHR[1]). The branch *B4*, due to its nonlinear correlation with previous branches, has more relatively large perceptron weights.

Source Program	misprediction rate (MR) and perceptron weights							
while (1) {								
r1 = Rand(0, 1000); r2 = Rand(0, 1000);								
c1 = r1 > r2; c2 = r1 > 500; c3 = r1 <= r2;								
if (c1)	B1: MR=50.09%							
count1++;	w1-w8: -1 -1 5 7 -3 -5 -1 -1							
if (c2)	B2: MR=25.45%							
count2++;	w1-w8: 26 0 0 2 2 2 0 0							
if (c3)	B3: MR=0%							
count3++;	w1-w8: -1 -31 1 1 -1 -1 -3 -1							
if (c2 ^ c3)	B4: MR=19.11%							
count4++;	w1-w8: 12 22 -12 10 10 -2 0 -2							
}								

Figure 2: A micro benchmark to illustrate that perceptron weights can be used as a quantitative measure of branch correlation.

Secondly, based on the correlation strength, the input to a perceptron predictor can be re-assembled to utilize resource more efficiently. For the example in Figure 2, we know that GHR[0] and GHR[1] carry the most correlation. Therefore, we can reduce the perceptron size to only exploit correlation from a 2-Bit GHR instead of an 8-bit GHR and we found that the perceptron predictor with the shorter history achieves similar or slightly better prediction accuracies (misprediction rates as 50.03%, 25.02, 0%, and 18.21% for *B1-B4* respectively). The improvement is due to the reduced noise effect in the perceptron training process. Note that such correlation-based input/history re-assembling is more flexible than history length adjustment [5]. For example, if it is determined that GHR[0] and GHR[7] carry the most correlation, we can still use a perceptron predictor with two weights to capture correlation from the 2-bit history formed with GHR[0] and GHR[7]. Dynamic history length adjustment, on the other hand, would fail to exploit such correlation patterns.

Thirdly, if the perceptron size is fixed (e.g., with 3 weights), the input can be re-configured to maximize branch correlation by exploring a much larger history-information set and replacing weak correlation bits with stronger ones. For example, rather than a 3-bit GHR, we can form the input vector to the perceptron predictor with GHR[1:0] and a redundant bit (GHR[0]^GHR[1]) since the redundant history is helpful to handle linearly inseparable data sets [8]. For the example in Figure 2, the misprediction rate of the branch *B4* drops to 0% with this redundant information bit.

In summary, we can see that with correlation strength effectively measured, there are new opportunities to further improve the performance of branch predictors, either upon resource efficiency or upon prediction accuracy. In the next section, we present our design effort to enhance prediction accuracy with a given prediction table budget for CBP-1. In the rest of the paper, an MAC perceptron predictor [8] is used as the baseline predictor although the proposed idea does not depend on any particular type of perceptron predictors.

### 3. Enhancing Branch Correlation Using Adaptive Information Processing

Based on the principles developed in Section 2, we propose two types of adaptive schemes to extract information bits to maximize branch correlation. The first is based on static workload profiling. The second is based on dynamic examination of perceptron weights and serves as a safe-net for workload misdetections or phase variations inside a workload.

In a MAC predictor, the perceptron weights are distributed in many weight tables. For each table, 4-bit information data are selected and used as the column address for the corresponding weight. The 4-bit inputs to all the weight tables simply form the information vector, as shown in Figure 3. The shaded units in Figure 3 are introduced for our proposed adaptation schemes, as described in detail next.

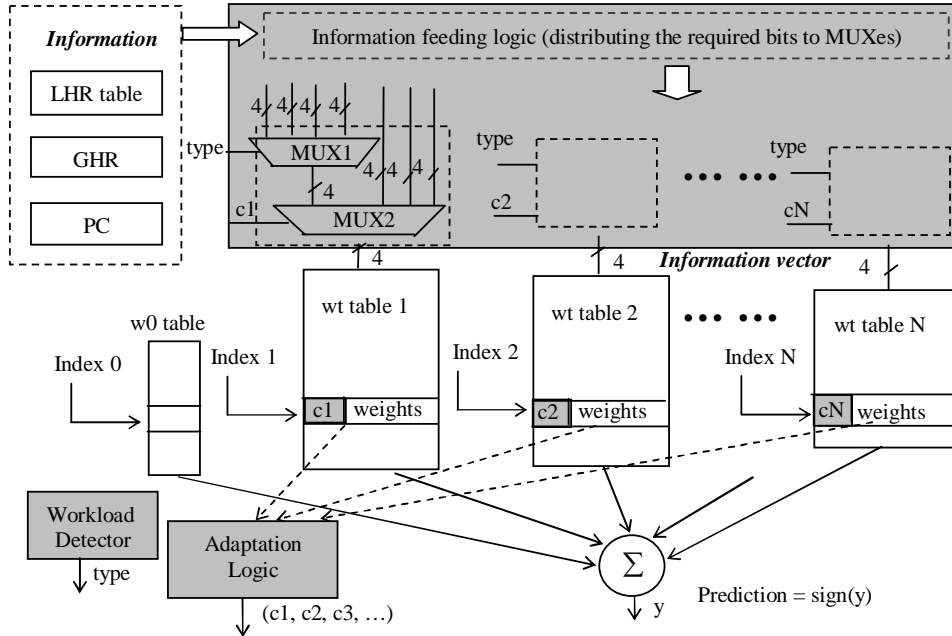


Figure 3: An MAC perceptron predictor with adaptive information processing (the shaded units are introduced for adaptive information processing).

#### 3.1. Profile-directed Adaptation

The distributed CBP traces are grouped in four categories: floating point (FP), integer (INT), multi-media (MM), and server benchmarks (SERV). Each type of workload exhibits different behavior and the same information bits, such as PC, LHR, or GHR, carry different correlation strength for different types of workloads. For example, SERV benchmarks have a large number of static branches and their PC bits carry strong correlation when multiple branches share the same perceptron. FP workloads, in comparison, have much fewer static branches and stronger correlation is found from global/local history than from PC bits.

Because of this workload-dependent behavior, we propose to use static profiling to determine appropriate configurations of the input vector for each type of workloads. Then, at run-time, a workload detector determines the workload type and selects a predetermined configuration accordingly. As this adaptation is based on profiling, it is called *profile-directed adaptation*. In our proposed implementation as shown in Figure 3, the inputs to the first multiplexer (MUX1) of

each perceptron weight table are the pre-determined configuration and the control of the multiplexer is the workload type figured out by the workload detector. Based on CBP traces, the predetermined configurations for each type of workloads are shown in Figure 4. Taking the weight table *I* (labeled ‘*wt table I*’) as an example, the inputs to its MUX1 are 4-bit local history (LHR[0:3]) for FP workloads and 4-bit PC (PC[0:3]) for other workloads.

For CBP traces, the workload detector uses the following detection criteria: (a) SERV benchmarks feature with a large number of static branches; (b) a small/medium number of static branches, a high number of floating point operations, and a high/medium number of instructions using XMM registers imply FP/MM workloads; and (c) the remaining benchmarks are treated as default INT workloads. More details can be found in the publicly distributed source code [1].

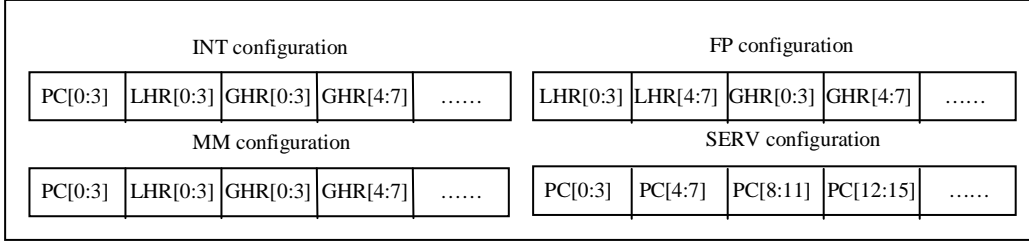


Figure 4: Pre-determined input configurations for different types of workloads.

### 3.2. Correlation-directed Adaptation

Profile-directed adaptation exploits coarse-grain workload behavior (i.e., one configuration for the entire run after the workload type is determined). If a workload exhibits significant phase behavior<sup>1</sup>, such a global configuration would fail to fit the changing requirements. To overcome such inefficiency, we propose an additional adaptation scheme based on dynamical correlation examination and it is named *correlation-directed adaptation*.

In correlation-directed adaptation, the strength of correlation is examined at a certain execution interval (or a program phase) and the inputs with weak correlation will be replaced with those potentially having stronger correlation. For example, if some LHR bits are found with weak correlation while GHR bits show stronger correlation, replacing those LHR bits with additional GHR bits would be an appealing choice.

As shown in Figure 3, besides the MUX1, which is used to support profile-directed adaptation, there is another multiplexer (MUX2) for each table. The MUX2 fine-tunes the pre-determined workload-dependent information vector based on the strength of branch correlation. For each weight table, the inputs to its MUX2 include the 4-bit output from MUX1, 4-bit GHR, 4-bit LHR, and 4-bit PC. The control bits of MUX2, *ci*, are initialized so that the 4-bit output from MUX1 is selected. Then, after a certain execution interval (or a program phase), the strength of branch correlation is examined for the 4-bit information data and those with weak correlation will be replaced. In order to measure correlation strength of the 4-bit input to weight table 1, for example, all the weights in the row selected by the *index1* are read and the summation of each weight (the absolute value) will be used as a quantitative measure of the correlation between the 4 information bits and the current branch instruction. If it turns out that the GHR/PC/LHR bits carries the strongest correlation, the 4 information bits with the minimum correlation will be replaced with 4 more GHR/PC/LHR bits by setting the control bits of MUX2 of the corresponding table. For example, for a branch in a MM benchmark, if its 4 PC bits (e.g., the

<sup>1</sup> The distributed CBP traces show little phase behavior given its limited trace length.

input to weight table 1) have the strongest correlation while its LHR bits (e.g., the input to weight table 2) have the weakest correlation, the input to weight table 2 will be configured as 4 more PC bits (e.g., PC[4:7]), replacing the LHR bits.

In the design shown in Figure 3, each entry in a weight table has its own control signal  $ci$  so that the input vector can be tuned for each entry. For example, the perceptron corresponding to the first entry of the weight table 1 may choose to exploit correlation from 4 PC bits while the one corresponding to the second entry exploits correlation from 4 GHR bits. Such flexibility requires more hardware resource (2-bit control for each entry) and more complex information feeding logic. A more efficient design is to tune the input configuration at the table level so that all the entries in the same table will share the same input configuration (i.e., only one  $ci$  for each table). We call these two adaptation designs as per-entry and per-table correlation-directed adaptation and examine their performance impact in Section 5.

#### 4. Overall Branch Predictor Structure

For conditional branches with simple taken/not taken patterns, a perceptron predictor is not a good candidate due to its complexity and potential latency and power consumption issues. In addition, such simple branch patterns could pollute perceptron weights when the same weights are used to predict other branches. Therefore, we propose a bias branch predictor and incorporate a loop branch predictor to handle these special branches, as shown in Figure 5.

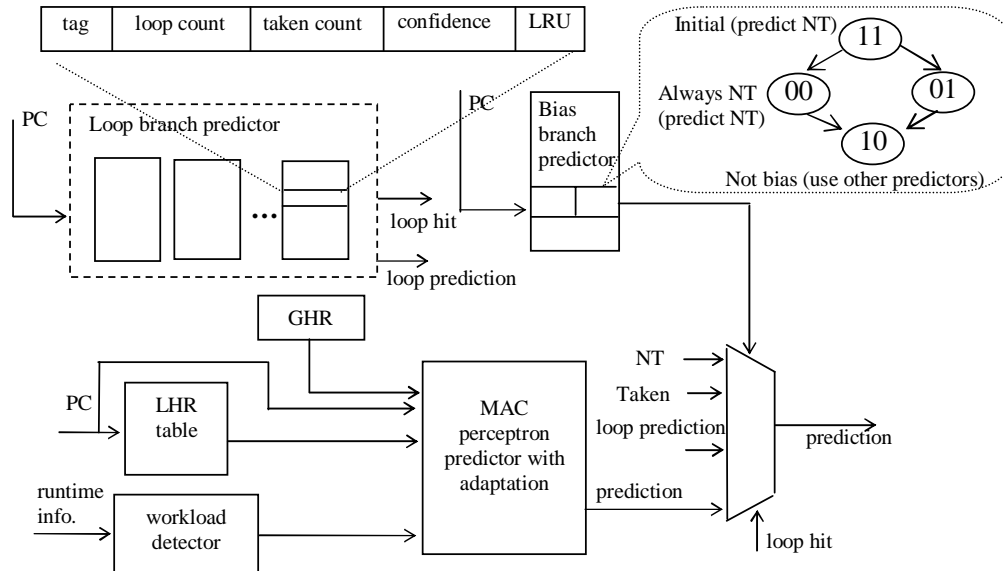


Figure 5: The overall branch predictor structure.

The bias predictor is an array of 2-bit FSMs indexed with PC. Each FSM transits from the initial state '11' into the state '01' or '00' depending on whether the branch is taken or not taken respectively. The state '01' implies the branch is always taken while the state '00' means the branch is always not taken. An FSM transits to the state '10' whenever the actual outcome disagrees with the bias state. In this case, the final prediction will be made by other predictors.

The loop branch predictor is a set-associative cache structure and each entry consists of loop count, current iteration count (taken count), tag, confidence (increased/decreased when the loop

count matches/mismatches), and LRU bits. A branch hits in the loop branch predictor only if there is a tag match and the confidence is high.

The overall prediction priority order is the bias prediction if a branch is biased (i.e., the bias state is not '10'), the loop branch prediction if the branch hits in the loop predictor, and then the prediction from the MAC perceptron predictor with adaptive information processing.

During update, the bias predictor is updated first. Only if the bias state is 10 (i.e., not biased), the loop branch predictor and the MAC perceptron predictor are examined. If the branch hits in the loop branch predictor, the MAC perceptron predictor is not updated. In this way, the adverse impacts from bias and loop branches upon the MAC perceptron predictor, including weight pollution and un-necessary updates, are effectively eliminated.

#### 4.1. Hardware budget requirements

With the branch predictor configuration that we proposed for CBP-1, the hardware budget requirements are summarized in Table 1.

Table 1: Hardware budget.

COMPONENT	CONFIGURATION	COST
Bias branch predictor	2293 entries	$2293 \times 2 = 4586$ bits.
Loop branch predictor	24 entries, 8-way set associative	1344 bits
Information	PC, GHR: 100 bits, LHR: 8 bits, 63 entries	$32 + 100 + 8 * 63 = 636$ bits
MAC perceptron predictor with adaptive information processing	W0 table: 61 entries, 8 bits each Other tables' sizes: 63, 55, 53, 53, 51, 49, 43, 41, 41, 39, 37, 37, and 35. Total MAC entries: 597. Each MAC entry has 16 weights. Each weight has 6 bits. Control bits: 2 bits each entry	$61 * 8 + 597 * 16 * 6 + 597 * 2 = 58994$ bits
Adaptation Logic	Interval : 100000 conditional branches	22 bits
Workload Detector	Interval: 10000 instructions and 10000 conditional branches	82 bits
Total Cost: 65649 bits		

## 5. Results

### 5.1. CBP-1 traces

Figure 6 shows the prediction accuracy of different branch predictors with the same 8KB prediction table size, including a g-share predictor, an MAC perceptron predictor, and our proposed predictor with different adaptation schemes, profile-directed adaptation (labeled 'profile'), profile-directed with per-table correlation-directed adaptation (labeled 'profile+per-table'), and profile-directed with per-entry correlation-directed adaptation (labeled 'profile+per-entry'). Compared to the g-share predictor, our proposed adaptive schemes reduce the misprediction rate by 47% on average. Compared to the fine-tuned baseline MAC perceptron predictor, the improvement from adaptation is close to 10%. As the distributed CBP traces do not exhibit strong phase behavior, profile-directed adaptation reaps the most benefit (2.847 mispredictions per 1K instructions) and correlation-directed adaptation only show small

incremental improvements (2.846 and 2.822 mispredictions per 1K instructions for per-table and per-entry correlation-directed adaptation respectively).

Since our overall predictor design contains multiple predictors handling different branches, we investigate the contribution of each predictor in Figure 7, which shows the percentage of predictions (both correct and incorrect) made by each predictor. It can be seen that the simple bias predictor and the loop predictor filter out a significant amount of dynamic branches and effectively reduces the aliasing and update cost of the MAC perceptron predictor. On average, among all the dynamic predictions, 17.61% (and 0.03%) are correct (and incorrect) predictions made by the bias predictor, 16.56% (and 0.03%) are correct (and incorrect) predictions made by the loop predictor, and 63.13% (and 2.64%) are correct and (incorrect) predictions made by the MAC perceptron predictor with adaptation.

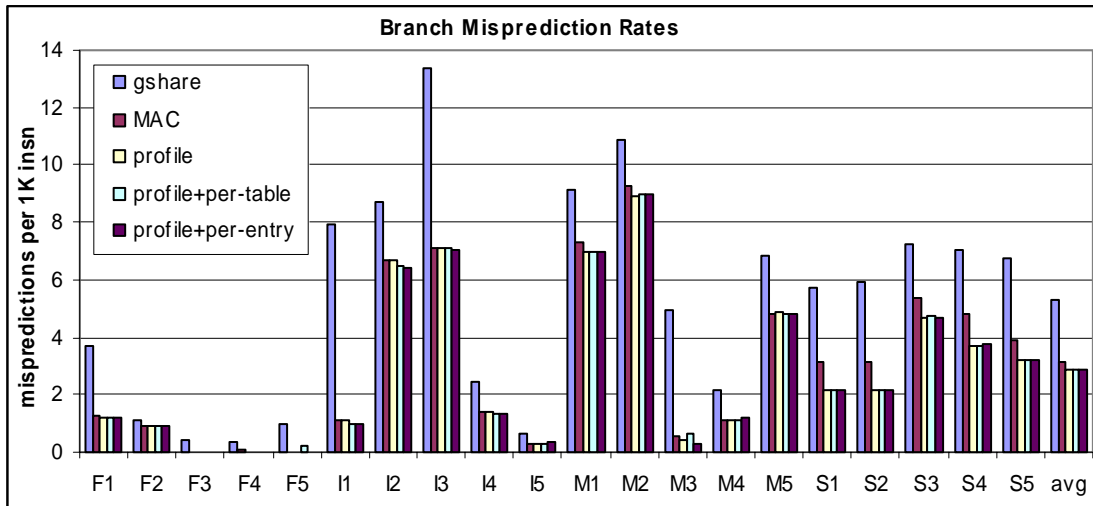


Figure 6: Branch misprediction rates on CBP-1 traces.

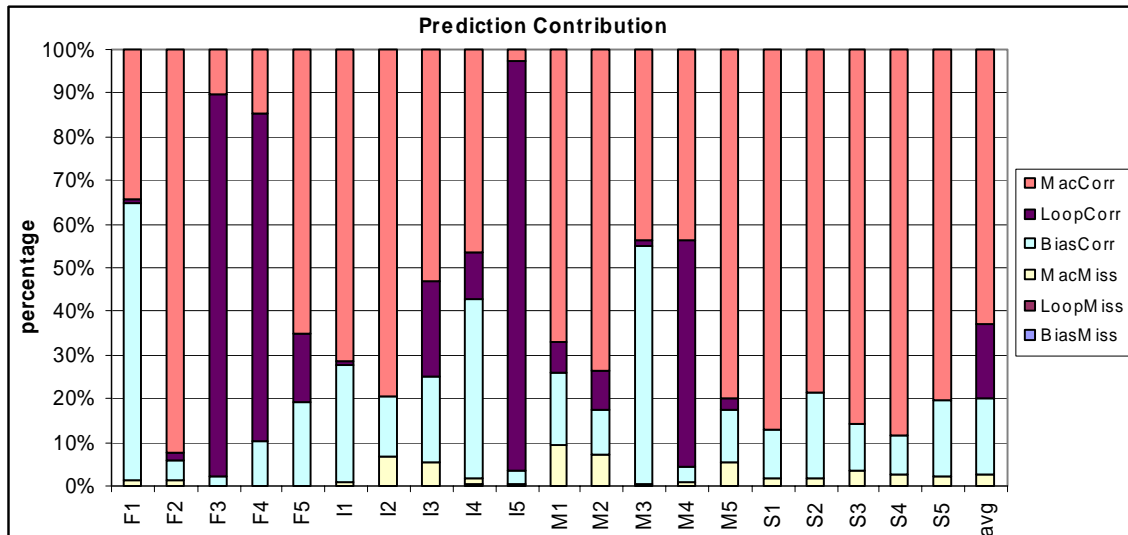


Figure 7: Prediction contribution from individual branch predictors in the overall design.

### 5.2. SPEC2000 benchmarks

In this experiment, we examine the prediction accuracy of the proposed scheme on SPEC 2000 CINT and CFP benchmarks [3]. In our simulation, the reference inputs are used. We fast-forward the first 500M instructions and simulate the next 1000M instructions. We also assume that the workload type (FP or INT) can be detected correctly and the FP/INT configurations profiled based on the CBP traces are then used for FP/CINT benchmarks.

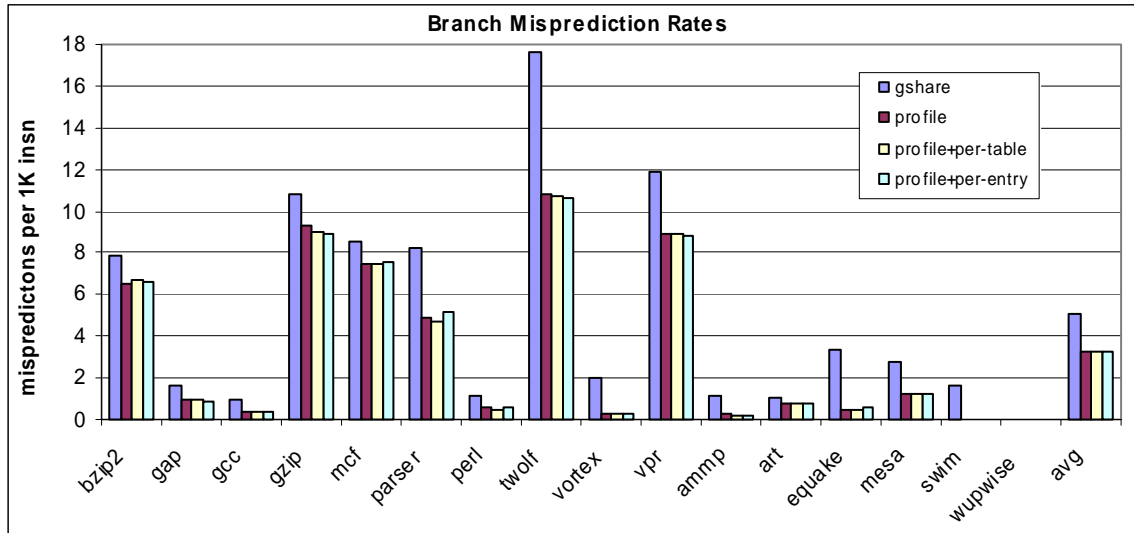


Figure 8: Branch misprediction rates on SPEC 2000 benchmarks.

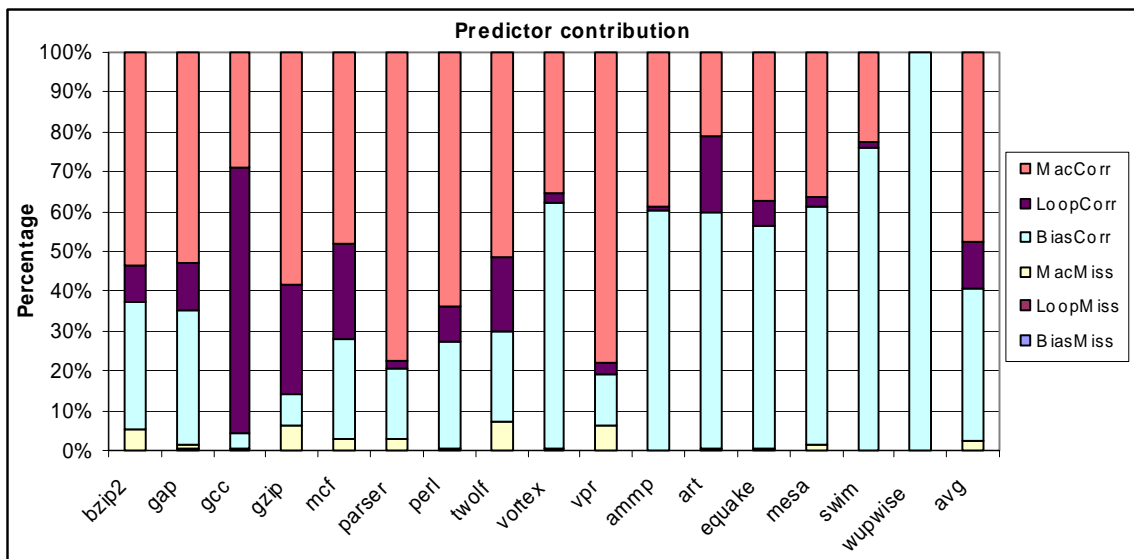


Figure 9: Prediction contribution from individual branch predictors in the overall design.

From the misprediction rates that are reported in Figure 8, it can be seen that our proposed predictor outperforms the g-share predictor by 35% on average. Another interesting observation is that the FP and INT configurations profiled using the CBP traces work well for SPEC benchmarks and only little improvements are from the additional per-table and per-entry

correlation-directed adaptation. This result is promising since profile-directed adaptation is much simpler to implement than correlation-directed adaptation.

Individual contributions from the bias, loop, and MAC perceptron predictor with adaptation are examined in Figure 9 and it can be seen that the bias and loop predictors accurately predicts a significant amount of dynamic branches for SPEC 2000 benchmarks, similar to CBP traces.

## 6. Conclusions

This paper takes an untraditional perspective to attack the branch prediction problem: rather than a new prediction scheme to better exploit branch correlation, we propose to process the inputs to branch predictors to maximize such correlation. An important observation is made from perceptron branch predictors that the perceptron weights provide an estimate of the correlation strength. Based on such correlation strength, novel adaptive information processing schemes are proposed to dynamically re-assemble the input information vector so that it contains only those history bits with the strongest correlation. The experiments using CBP traces and SPEC 2000 benchmarks show that our proposed approach achieves significant improvement on prediction accuracy over the competing branch prediction schemes.

## References

- [1] The official CBP-1 website: <http://www.jilp.org/cbp/>.
- [2] I. K. Chen, J. T. Coffey, and T. N. Mudge, "Analysis of branch prediction via data compression", *Proc. of the 7th Int. Conf. on Arch. Support for Programming Languages and Operating Systems (ASPLOS-VII)*, 1996.
- [3] J. Henning, "SPEC2000: measuring CPU performance in the new millennium", *IEEE Computer*, July 2000.
- [4] T. Juan, S. Sanjeevan, and J. Navarro, "A third level of adaptivity for branch prediction", *Proc. of the 25<sup>th</sup> Int. Symp. on Comp. Arch (ISCA-25)*, 1998.
- [5] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons", *Proc. of the 7th Int. Symp. on High Perf. Comp. Arch (HPCA-7)*, 2001.
- [6] D. Jimenez and C. Lin, "Neural methods for dynamic branch prediction", *ACM Trans. on Computer Systems*, 2002.
- [7] S. MacFarling, "Combining branch predictors", Technical Report, DEC, 1993.
- [8] A. Sez nec, "Revisiting the perceptron predictor", Technical Report, IRISA, 2004.
- [9] T.-Y. Yeh and Y. Patt, "Alternative implementations of two-level adaptive branch prediction", *Proc. of the 22<sup>nd</sup> Int. Symp. on Comp. Arch (ISCA-22)*, 1995.