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

Hongliang Gao Huiyang Zhou
*School of Computer Science*
*University of Central Florida*
*{hgao, 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 effective history length to be explored, which in turn undermines the potential performance. In this paper, we propose an adaptive approach to dynamically reconfigure the input vector to a perceptron predictor to facilitate correlation exploitation. In this way, a much larger information set can be explored without increasing the size of a perceptron predictor. Along with carefully designed predictor parameters, the proposed scheme achieves significant improvements on prediction accuracy.*

## 1. Introduction

As presented in [1], dynamic branch prediction schemes can be described using a general conceptual system model, shown in Figure 1. The source information, including branch addresses, local/global histories, 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 (FSM) are used as the predictor [1]. Recently, predictors based on perceptrons [2],[3] have been proposed and shown to have superior prediction accuracy to the widely used FSM-based predictors such as g-share [4] and two-level predictors [6]. 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 [5], 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 [5].

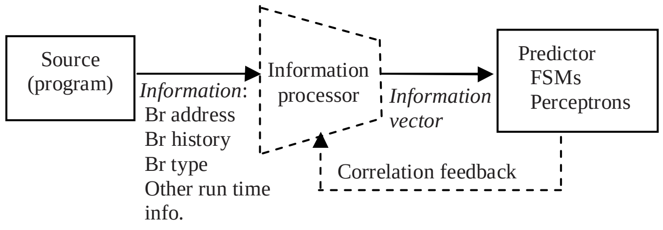

**Figure 1. A conceptual system model for branch prediction [1].**

In this paper, we use a MAC perceptron predictor with pseudo-tagging and skewing as our base predictor and propose a novel adaptive information processing scheme to extract the information vector. Our scheme is based on a *key observation* that perceptron weights can be used as a quantitative measure of correlation between the current prediction and the information vector. Therefore, we can adaptively re-assemble the information vector to maximize the correlation without increasing the size of perceptron predictors. In addition, since such adaptation is performed at a coarse grain, e.g., at program phase boundaries or at a fixed interval of a large number of branches, the adaptation logic is *not* latency critical.

The rest of the paper is organized as follows. Section 2 presents the detailed architecture design, the adaptation algorithm, and the hardware budget requirement. Section 3 briefly discusses the results and Section 4 concludes the paper.

## 2. Predictor Design

### 2.1. A MAC perceptron predictor with adaptive information processing

The structure of a MAC perceptron predictor with adaptive information processing is shown in Figure 2. In the predictor, perceptron weights in MAC representation are distributed in many weight tables. For each table, 4-bit information data are selected and used as the column address for the corresponding

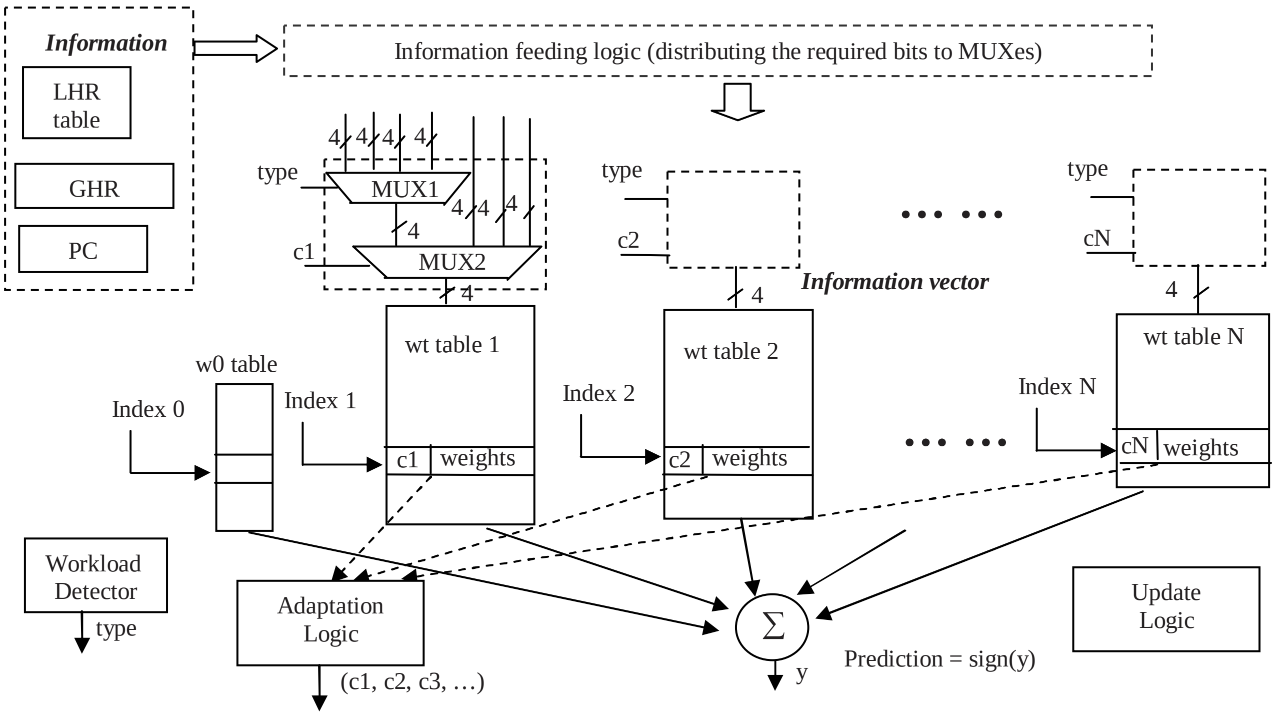

**Figure 2. A MAC perceptron predictor with adaptive information processing.**

weight. The 4-bit inputs to all the weight tables form the information vector.

The information vector is assembled using multiplexers. For each table, there are two multiplexers (MUX1 and MUX2) controlled by '*type*' and '*ci*' respectively. The control signal of MUX1 is determined by a workload detector and the adaptation logic controls MUX2. The inputs to MUX1 are configured based on profiling information or static analysis, providing a workload-dependent way to extract the information vector. In our design, we pre-determine the configurations for four types of workloads: floating point (FP), integer (INT), multi-media (MM), and server benchmarks (SERV), as shown in Figure 3. Taking the weight table 1 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. The default workload type is set as INT and the workload detector uses dynamic execution information to identify the workload type and select one of the pre-coded configurations. As this adaptation is based on profiling, it is called *profile-directed adaptation*.

The MUX2, on the other hand, fine-tunes the pre-determined workload-dependent information vector based on the strength of correlation and serves as a safe-net for incorrect workload detections. 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 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 (absolute value) will be used a quantitative measure of the correlation between

INT configuration

| | | | | |
|---|---|---|---|---|
| PC[0:3] | LHR[0:3] | GHR[0:3] | GHR[4:7] | …… |

FP configuration

| | | | | |
|---|---|---|---|---|
| LHR[0:3] | LHR[4:7] | GHR[0:3] | GHR[4:7] | …… |

MM configuration

| | | | | |
|---|---|---|---|---|
| PC[0:3] | LHR[0:3] | GHR[0:3] | GHR[4:7] | …… |

SERV configuration

| | | | | |
|---|---|---|---|---|
| PC[0:3] | PC[4:7] | PC[8:11] | PC[12:15] | …… |

**Figure 3. The pre-determined information-vector configuration for each type of workloads.**

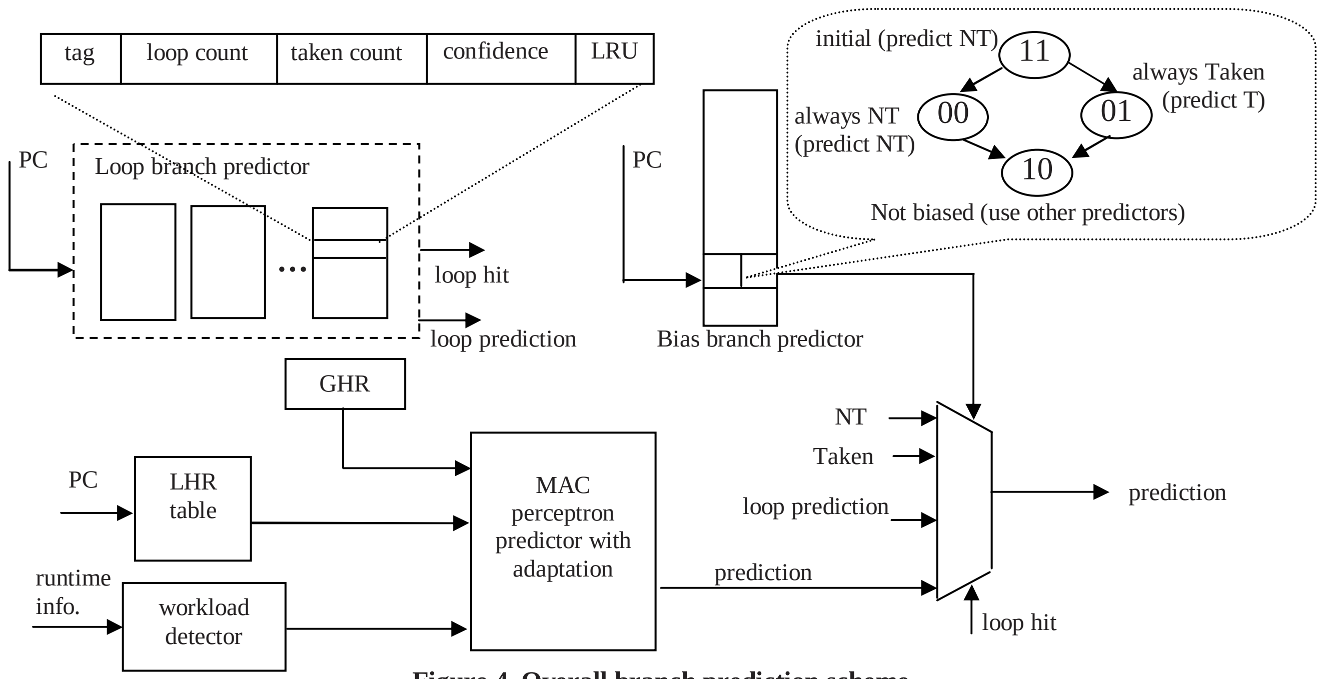

**Figure 4. Overall branch prediction scheme.**

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 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. Since this adaptation is based on dynamic correlation, it is called *correlation-directed adaptation*.

## 2.2. Overall branch predictor structure

In our design, the overall predictor consists of a bias branch predictor making predictions for those branches that are *always* taken or not-taken, a loop branch predictor handing loop-back branches, and a MAC perceptron predictor with adaptation predicting the remaining branches. The overall predictor structure is shown in Figure 4.

The bias branch predictor is an array of 2-bit FSMs. For each FSM, from the initial state '11', it transits into the state '01' or '00' depending on whether the branch is taken or not. The state '01' implies the branch is *always* taken while the state '00' means the branch is *always* not taken. The 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 either the loop branch predictor or the MAC perceptron predictor. The loop branch predictor is a set-associative cache structure and each entry, shown in Figure 4, 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 the branch is biased (i.e., the bias state is not '10'), the loop branch prediction if the branch hits in the predictor, and then the prediction from the MAC perceptron predictor.

During the update, the bias predictor is updated first. Only if the bias state is 10 (i.e., not biased), the loop branch predictor and 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 aliasing impacts from the bias and loop branches are effectively eliminated.

The workload detector used in profiled-directed adaptation examines dynamic instruction information to detect the workload type. It counts the number of floating-point instructions and the number of instructions using XMM registers. It also periodically consults (at a relatively large interval) the bias predictor to estimate how many static branches (i.e., not biased ones) are accessing the perceptron predictor. The detection criteria include: (a) server 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 floating-point/multi-media workloads; and (c) the remaining benchmarks are treated as default integer workloads.

### 2.3. Hardware budget requirements

The hardware storage requirements of our predictor can be calculated as follows based the configuration used in our experiments.

- Bias branch predictor. 2293 entry x 2 bits per entry = 4586 bits.
- Loop branch predictor. 24 entry, 8-way set associative. For each entry, 9-bit loop count + 9-bit current iteration count + 3 bit LRU + 32 bit tag + 3-bit confidence = 56 bits. 24-entry requires 1344 bits.
- Local branch histories. 63-entry table with an 8-bit history for each entry. 63 x 8 = 504 bits.
- Global branch history. 100 bits
- PC. 32 bits
- MAC perceptron branch predictor. It contains a w0 table (61 entries with 8 bits for each entry = 488 bits), 13 perceptron weight tables. Their entry sizes are: 63, 55, 53, 53, 51, 49, 43, 41, 41, 39, 37, 37, and 35 respectively. For each entry, it contains one 16 6-bit weights and 2-bit multiplexer (MUX2) control information. So, 58994 bits in total.
- Adaptation module. Correlation-directed adaptation can be implemented using combinational logic. For profile-directed adaptation, pre-determined controls can be implemented as fixed connections. For adaptation interval (every 100000 conditional branches), we use a 17-bit counter. For multiple adaptation intervals, another 5-bit counter is used. So, total storage budget for the adaptation module is 22 bits.
- Workload detection module. It needs the following information to determine when to make detection: the number of instructions executed (a 14-bit counter), number of floating-point instructions for two consecutive phases (two 11-bit saturating counters), number of static branch instructions (a 10-bit saturating counter) obtained by consulting the bias predictor periodically, number of dynamic conditional branches (a 14-bit saturating counter), and number of instructions using XMM registers (two 11-bit saturating counters). The rest is combinational logic. So, 82 bits in total.
- In summary, the hardware budget for our *overall* predictor is 65664 bits (less than 64 x 1024 + 256 = 65792 bits).

## 3. Results

The detailed results for each benchmark are reported in a separate result table. The overall average miss prediction rate is significantly reduced compared to a g-share predictor of the same size. Compared to the fine-tuned baseline MAC perceptron predictor, the improvement from adaptation is close to 10%.

## 4. Conclusions

In this paper, an adaptive information processing scheme is proposed to further enhance the prediction accuracy of perceptron-based branch predictors. Our adaptation scheme dynamically assembles the input information vector based on quantitative measure of correlation information and achieves significant improvement on prediction accuracy.

### References

[1] 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.

[2] D. Jimenez and C. Lin, “Dynamic branch prediction with perceptrons”, *Proc. of the 7th Int. Symp. on High Perf. Comp. Arch* (HPCA-7), 2001.

[3] D. Jimenez and C. Lin, “Neural methods for dynamic branch prediction”, *ACM Trans. on Computer Systems*, 2002.

[4] S. MacFarling, “Combining branch predictors”, *Technical Report*, DEC, 1993.

[5] A. Seznec, “Revisiting the perceptron predictor”, *Technical Report*, IRISA, 2004.

[6] T.-Y. Yeh and Y. Patt, “Alternative implementations of two-level adaptive branch prediction”, *Proc. of the 22nd Int. Symp. on Comp. Arch* (ISCA-22), 1995.