The Combined Perceptron Branch Predictor

Matteo Monchiero Gianluca Palermo

Report n. 2004.35

Politecnico di Milano – Dipartimento di Elettronica e Informazione
Via Ponzio, 34/5, 20133 Milan, Italy

{monchier, gpalermo}@elet.polimi.it

Abstract— **Previous works have shown that neural branch prediction techniques achieve far lower misprediction rate than traditional approaches. We propose a neural predictor based on two perceptron networks: the *Combined Perceptron Branch Predictor*. The predictor consists of two concurrent perceptron-like neural networks; one using as inputs branch history information, the other one program counter bits. We carried out experiments proving that this approach provides lower misprediction rate than state-of-the-art conventional and neural predictors.**

I. Introduction

Modern high-performance microprocessors rely on sophisticated and accurate branch predictors to efficiently exploit Instruction Level Parallelism (ILP). Complex front-ends, capable of filling large instruction windows, are required to provide high frequency of operations and aggressive parallelism. Branch prediction is a key element of such a system, providing correct fetch beyond branch boundary and, therefore, large throughput instruction deliver.

Traditional branch predictors [1] are based on one or more tables of saturating 2-bit counters. The simplest predictor, known as *Bimodal*, is composed of a table of 2-bit counters indexed by the least significative bits of the Program Counter (PC). More complex approaches, the *Two-Level Adaptive* predictors [2], use a local or global history, stored in a Branch History Register (BHR) to index into the 2-bit counter table, which represents the second level history. To get advantages of both local and global indexing, *hybrid* predictors, which combines two-level or bimodal predictors as components, [3] have been proposed. Several advanced branch predictors have been suggested so far in the literature. Most of them are 2-bit counter table based predictors and they are organized in order to minimize interference which may occur in the counter tables.

In this paper, we present an innovative branch predictor architecture, based on a neural approach, first proposed by Jimenez *et al.* in [4] (the *Perceptron* predictor). Our proposal features a novel mechanism, based on an additional perceptron, using some PC bits as inputs, to achieve superior accuracy with respect to a single perceptron approach.

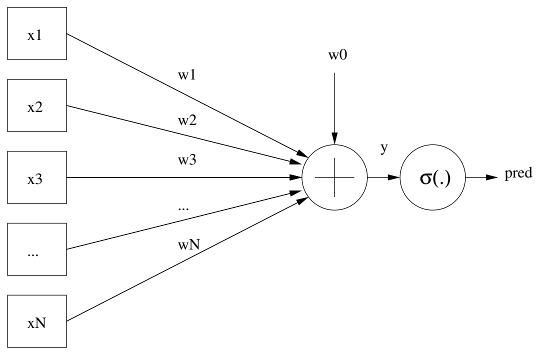

Fig. 1. Perceptron neural network

Section II introduces some background about neural branch prediction. In Section III, our proposal is presented. Section IV shows obtained experimental results. Finally Section V concludes the paper.

II. Background

Branch predictors based on neural methods have been recently studied [4], [5], [6], showing that they are the most accurate predictors in the literature. In fact, neural networks can exploit much longer histories than conventional branch predictors.

The simplest neural network is the *perceptron*, whose diagram is shown in Figure 1. The network output, $pred$, is a non-linear function (σ) of y, which is a linear combination of the network inputs, as stated in following equations:

$$pred = \sigma(y) \tag{1}$$

$$y = w_0 + \sum_{i=1}^{i=N} x_i w_i \tag{2}$$

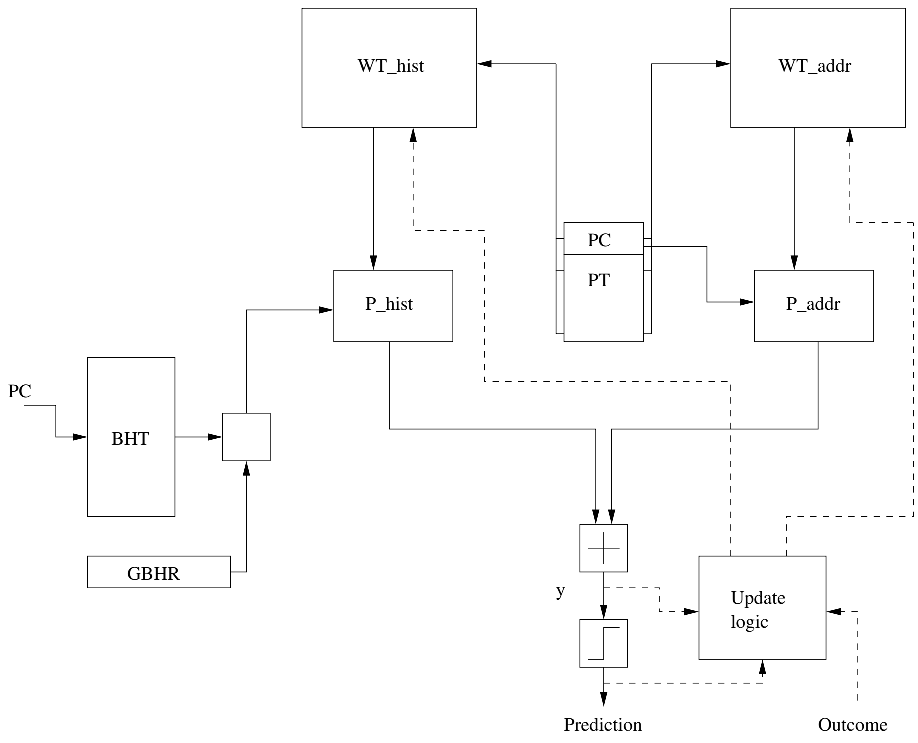

Fig. 2. Block diagram of the *Combined Perceptron Branch Predictor*

where w_i are $N+1$ weights and x_i are N inputs; w_0 is called bias weight. Function σ can assume various shapes for a generic neural network. Perceptron uses as σ a $step$ function, which is a natural choice, when dealing with branch prediction patterns. The $step$ function means taken when it is 1 and not-taken when 0. $\bar{w} = [w_0, w_1, \cdots, w_N]$ is said *weight vector* and specifies the perceptron. Weights can be dynamically trained, so that prediction run-time adapts to the real taken/not-taken branch pattern.

The *Perceptron* predictor, presented in [4], uses perceptrons to predict branch outcome. It is a history based predictor, since it maintains a Global Branch History Register (GBHR) and a set of local Branch History Registers (BHR), collected in a Branch History Table (BHT). A history register obtained concatenating local and global history is used as input of a perceptron network to perform the prediction.

Weights are 8-bit integers and they are selected in a $n \times (h+1)$ matrix, called *Weight Table* (WT). n and h are design parameters: n has the meaning of number of entries of the WT, while h is the size of the history register, which is the network input. Each row of the matrix is an $(h+1)$-length weight vector, which determines the perceptron. When the prediction is performed, the least significative bits of the PC are used to select the row corresponding to the weight vector to use. A fast adder provides to generate the summation of the weights, according to applied inputs (see Equation 2), and a comparator makes the prediction (see Equation 1).

III. Proposed Predictor Architecture

The *Combined Perceptron Branch Predictor*, proposed in the paper, is based on the idea to combine two different kinds of *Perceptron*: a *history-based* one and a *address-based* one. The *address-based Perceptron* has as inputs some bits of the PC. Its output is sensitive to the branch address and, if combined with the output of the *history-based Perceptron*, which is sensitive to branch history, it adds a contribution which significantly improves the prediction accuracy.

The two subpredictors (the *history-based* one and the *address-based* one), which compose the whole predictor are *Perceptron* predictors modified with respect to the ones described in Section II. As proposed in [6], we adopted, as components of our predictor, *path-based* perceptrons, which use branch path information to get superior accuracy. The path of a branch is composed of the past branch PCs and it is stored in a Path Table (PT). In a path-based perceptron, the i-th weight of the weight vector is the element of the i-th column of the Weight Table, indexed by the i-th element of the Path Table, as indicated by the following formula:

$$w_i = WT[PT[i]] \tag{3}$$

Algorithm 1 Prediction algorithm

```
/*Calculate output of history-based perceptron*/
y_hist=W1[PC][0];
for (j = 1; j <= history_length; j++)
{
    k = path[j-1];
    if (history_reg[j-1])
     y_hist += W1[k][j];
    else
     y_hist -= W1[k][j];
}

/*Calculate output of address-based perceptron*/
y_addr=W2[PC][0];
for ( j=1; j <= N_BITS; j++)
{
    k = path[j-1];
    if (PC[j-1])
y_addr += W2[k][j];
    else
      y_addr -= W2[k][j];
}
y = y_hist + y_addr;

if ( y >= 0 ) prediction = true;
else prediction = false;
```

Furthermore, σ function is moved from the output of the two single subpredictors to the output of the whole predictor. Single subpredictors behave as perceptron-like neural networks with a linear σ function.

Figure 2 shows the block diagram of the proposed predictor. We indicated as *WT_hist* and *P_hist* the Weight Table and the perceptron logic of the *history-based* subpredictor, while *WT_addr* and *P_addr* are related to the *address-based* one. Perceptron logic is substantially composed of an adder and some multiplexers which sums selected weights depending on the inputs.

The Path Table (PT) holds last PC and it is used to index into the Weight Tables. GBHR and a BHT store information related to branch outcome history and supply the history register which is the input of the *history-based* subpredictor. The *update logic* is the circuitry needed to update the predictor Weight Tables. Dashed lines represent data transfers needed by the update phase.

The prediction algorithm is shown, as C-like pseudocode, in Algorithm 1. Weight Tables of both predictors are concurrently accessed to get weight vectors. The outputs of the perceptrons are calculated and summed together. Finally a comparator decides the prediction whether the obtained value is greater or less than zero.

Update algorithm details are shown in Algorithm 2. The weights of the two subpredictors tables are modified on mispredictions or when the value of y is too small. A threshold is established for this purpose. Its value has been set, so that weight vectors are updated if y falls into a value range which is half of the weight range. The update is performed following the rule:

$$\Delta w = outcome \oplus input \tag{4}$$

A weight is incremented if the corresponding input has given positive contribution, otherwise it is decremented. The GBHR, the BHT and the PT are also updated in this phase.

The proposed architecture can be configured by setting the following parameters:

- W1_SIZE: number of entries of the Weight Table of the *history-based* subpredictor;
- HL: global history length;
- BHT_SIZE: BHT number of entries;

Algorithm 2 Update algorithm

```
if (last_prediction!=outcome ||
    (last_y <= THETA && last_y >= -THETA))
{
   /*update history-based perceptron*/
   if (outcome) weight_inc(W1[PC][0]);
   else weight_dec(W1[PC][0]);

   for (j = 1; j <= history_length ; j++)
   {
      k = path[j-1];
      if (outcome == hist[j-1])
        weight_inc(W1[k][j]);
      else
        weight_dec(W1[k][j]);
   }

   /*update address-based perceptron*/
   if (outcome)  weight_inc(W2[PC][0]);
   else weight_dec(W2[PC][0]);

   for (j = 1; j <= N_BIT; j++)
   {
     k = path[j-1];
     if (outcome == PC[j-1])
       weight_inc(W2[k][j]);
     else
       weight_dec(W2[k][j]);
   }
}
update_ghist();
update_lhist();
update_path();
```

TABLE I
PREDICTOR PARAMETERS

Parameter	Value
W1_SIZE	137
HL	25
BHT_SIZE	2048
LHL	4
W2_SIZE	254
N_BITS	11

TABLE II

PREDICTOR RESOURCE OCCUPATION

Resource	Symbolic expression [bit]	Resource occupation [bit]
WT_hist	$W1\_SIZE \times (HL + LHL + 1) \times 8$	32,880
WT_addr	$W2\_SIZE \times (N\_BITS + 1) \times 8$	24,384
BHT	$BHT\_SIZE \times LHL$	8,192
GBHR	HL	25
PT	$max\{HL + LHL, N\_BIT\} \times \lceil log_2(max\{W1\_SIZE, W2\_SIZE\}) \rceil$	232
Last y register		32
Last prediction register		1
Total		65,746

- LHL: local history length;
- W2_SIZE: number of entries of the Weight Table of the *address-based* subpredictor;
- N_BITS: number of bits to be used as inputs of the *address-based* subpredictor;

We explored design space parameters and we find the optimal predictor configuration, shown in Table I.

From these parameters resource occupation of the predictor can be easily calculated. We list the storage resources needed by the proposed predictor and the number of bits needed in Table II. *WT_hist* and *WT_addr* are the two Weight Tables. *BHT* and *GBHR* are the Branch History Table and the Global Branch History Register. *PT* is the Path Table. In particular, we can observe that Weight Table size is determined by the dimensions of weight vectors, other than the number of entries. Path Table size depends on the largest Weight Table to index, relatively both to the number of entries and to the number of bits of each element in the PT. Furthermore, registers for recording the last value of y and the last prediction is needed by the update logic.

IV. EXPERIMENTAL RESULTS

We report experimental results, obtained using the CBP framework on the 20 benchmark traces provided. Table III shows the misprediction rate (expressed in mispredicted branches per 1000 instructions) achieved by our predictor. An average value of 3.486 can be observed, 34% better than the baseline GShare predictor (15-bit history length). We conducted experiments comparing the *Combined Perceptron* to a state-of-the-art perceptron predictor. Our approach results in a much lower misprediction rate.

TABLE III

MISPREDICTION RATE [NUMBER OF MISPREDICTED BRANCHES PER 1000 INSTRUCTIONS]

Benchmark	Misprediction Rate
DIST-FP-1	1.858
DIST-FP-2	1.098
DIST-FP-3	0.425
DIST-FP-4	0.195
DIST-FP-5	0.183
DIST-INT-1	3.578
DIST-INT-2	6.588
DIST-INT-3	8.065
DIST-INT-4	1.584
DIST-INT-5	0.325
DIST-MM-1	7.548
DIST-MM-2	9.460
DIST-MM-3	1.408
DIST-MM-4	1.444
DIST-MM-5	5.046
DIST-SERV-1	3.337
DIST-SERV-2	3.510
DIST-SERV-3	5.204
DIST-SERV-4	4.737
DIST-SERV-5	4.118
Average	3.486

V. CONCLUSIONS

An innovative branch predictor architecture, based on a neural approach, has been presented. We carried out experiments on the benchmarks provided. The proposed predictor achieves 34% lower misprediction rate than the baseline GShare predictor and lower misprediction than a state-of-the-art *Perceptron* predictor.

REFERENCES

[1] M. Evers and T.-Y. Yeh. Understanding branches and designing branch predictors for high performance microprocessors. *Proceedings of the IEEE*, 89(11):1610–1620, November 2001.

[2] T.-Y. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *Proc. MICRO-24*, pages 51–61. ACM Press, 1991.

[3] S. McFarling. Combining branch predictors. Technical Report TN-36, Western Research Laboratory, June 1993.

[4] D. A. Jimenez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, 20(4):369–397, November 2002.

[5] L. N. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *Proceedings of the International Joint Conference on Neural Networks*, July 1999.

[6] D. Jimenez. Fast path-based neural branch prediction. In *Proceedings of MICRO-36*, 2003.