Global/Local Hashed Perceptron Branch Prediction

Anthony S. Fong and C. Y. Ho
Department of Electronic Engineering, City University of Hong Kong

Abstract

This paper introduces combining local history hashing and global history hashing in perceptron branch prediction. The proposed perceptron predictor utilizes self-history as well as global history in indexing different weights of a perceptron. The simulation results show that our proposed perceptron predictor is more accurate than the one using either global history hashing or local history hashing alone. Our proposed perceptron predictor improves the misprediction rate by up to 26.9% over path-based neural predictor, and 17.2% over hashed perceptron predictor. A reducing aliasing approach is employed to improve the accuracy of our proposed perceptron predictor by diminishing the impact of destructive interference, supported by the simulation results.

Keywords: *branch prediction, perceptrons, hashing, neural networks.*

1. Introduction

The penalty incurred by misprediction increases as the instruction issue rate widens as well as the pipeline depth deepens. Therefore, branch prediction accuracy is considered to be a critical design issue for modern microarchitectures.

In the past, most research works for branch prediction have focused primarily on 2-bit saturating up-down counter. Yeh and Patt [9] investigated exploiting two level branch histories to capture the behavior of branches. Pan et al. [10] and Sechrest et al. [11] examined the correlation between branches. Sprangle et al. [7] and Lee et al. [8] explored how to reduce the effect of negative interference between branches.

In recent years, perceptron is introduced as an alternative to the 2-bit saturating up-down counter for branch prediction. Jiménez and Lin [1, 2] proposed the original perceptron predictor. Jiménez [3, 4] investigated the path-based neural branch predictor. Tarjan and Skadron [5] explored merging path and gshare indexing in perceptron predictor.

The proposed solution attempts to exploit local history together with global history in hash indexing to select different weights of a perceptron. It benefits branches exhibiting repetitive behaviors as well as branches strongly correlated with previous neighboring branches. It is also able to learn linearly inseparable behaviors. This proposed predictor is presented in a previous paper [14]. We further our evaluation by comparison with other well-known perceptron predictors. In addition, we introduce a reducing aliasing approach to alleviate the impact of destructive interference resulting in improved prediction accuracy.

2. Perceptron branch predictors

Perceptron is the simplest model of an artificial neural network used for pattern classification. As depicted in figure 1, each perceptron keeps track of a weight vector w. The w_0 is a bias weight to which the corresponding input unit is always set to +1. The perceptron receives an input vector x. Each input unit x_i is associated with a weight w_i.

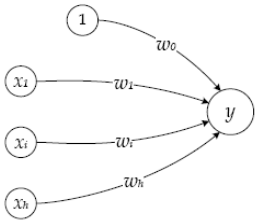

Figure 1. Perceptron

The output y, is the dot product of input vector and weight vector. The sign of output y serves to make prediction, that non-negative output indicates "*taken*" and negative output indicates "*not taken*".

$$y = w_0 + \sum_{i=1}^{h} x_i w_i$$

Once the actual branch outcome becomes known, the training algorithm is performed to modify the weight values. Let t be the actual branch outcome, and θ be the training threshold as a parameter to determine whether the perceptron needs further update or not. The following pseudocode is the training algorithm:

```
if sign(y) ≠ t or |y| ≤ θ then
        for j in 0..h do
                w_i := w_i + t x_i
        end for
end if
```

The weights are updated when the prediction outcome does not agree with the actual branch outcome or when the absolute value of y remains below the threshold value.

The drawback is the ability of learning linearly separable functions [7]. A decision boundary, hyperplane, separating two decision regions, is defined by the equation,

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.258

$$w_0 + \sum_{i=1}^{h} x_i w_i = 0$$

Intuitively, the two decision regions are the non-negative outputs and the negative outputs. A function is said to be "linearly separable" if given a set of input patterns, all of the non-negative outputs are sufficiently separated from the negative outputs by the hyperplane [7].

3. Global/Local hashed perceptron predictor

The proposed perceptron branch predictor combines local history XOR hashing with that of global history. The local branch history (or self-history) helps to predict the branches exhibiting repetitive behaviors.

3.1. Multiple indexing methods

A table of perceptrons can be interpreted as an n x (h+1) matrix W[0..n-1, 0..h], where n is the number of perceptrons and h the number of weights in a perceptron. For each column of the matrix, one weight is selected as a weight element of a perceptron. Different weights of a perceptron can be indexed by different methods. All of the indexed weights serve as a weight vector, [w_0,…,w_h], for calculation of dot product of a perceptron. The following three indexing methods are incorporated in perceptron branch predictor. Here *addr* is the current branch address. The *local* represents local branch history segment of the branch to be predicted and *global* represents global.

$index[j] = addr \bmod n$
$index[j] = (local[j] \oplus addr) \bmod n$
$index[j] = (global[j] \oplus addr) \bmod n$

Previous researches have investigated that different weights of a perceptron are selected by different mappings separately [5, 16]. However, they predict the direction of current branch based upon the global branch history alone.

3.2. Weight selection approach

Figure 2 illustrates the proposed weight selection approach. Some weights are selected by XOR'ing local branch history with branch address, and others are selected by global history. The bias weight w_0 is accessed by the branch address directly, and all input values to the selected weights are +1, which are 8-bit signed integers.

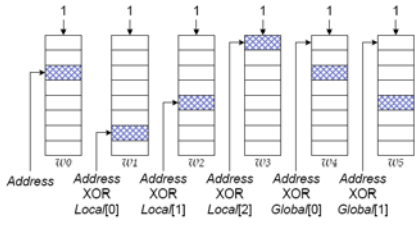

Figure 2. Weight selection of global/local hashed perceptron predictor

In this example, the number of perceptrons n is 8 and the number of weights in a perceptron h is 5, and the number of weights indexed by XORing local branch history with branch address is 3. The remaining weights of a perceptron (5 minus 3) are indexed by XORing global branch history with branch address.

3.3. Prediction algorithm

The following pseudocode shows the prediction algorithm. Here the definitions of W, n, h, *addr*, *local*, *global* are the same as those of previous subsection 3.1. Let *length* be the number of weights in a perceptron indexed by the local history XOR mapping. Then the number of weights indexed by the global history XOR mapping is (h minus *length*).

```
function prediction (addr: integer): {taken, not_taken}
begin
    index[0] := addr mod n
    y := W[index[0], 0]
    for j in 1..h do
        if j ≤ length then
            index[j] := (local[j-1] ⊕ addr) mod n
            y := y + W[index[j], j]
        else
            index[j] := (global[j-length] ⊕ addr) mod n
            y := y + W[index[j], j]
        end if
    end for
    if y ≥ 0 then
        prediction := taken
    else
        prediction := not_taken
    end if
end
```

The output y is the sum of all weight elements w_i. The prediction outcome is made according to the sign of computed output y. The branch is predicted as *taken* if the computed output is non-negative, whereas it is predicted as *not taken* if the computed output is negative.

3.4. Training algorithm

Once the actual outcome of the branch is resolved, the training algorithm is invoked to modify the set of weights. Below we give the pseudocode of the training algorithm.

The perceptron is trained when the actual outcome does not agree with the prediction result or the absolute value of computed output y remains below the threshold value θ. Here the threshold value is set to be [1.93*h + h/2], the same as that of hashed perceptron predictor [5]. The weight values are incremented by 1 if the actual outcome is taken, and decremented by 1 otherwise. The actual outcome is shifted left into the local history shift register of that branch as well as the global history shift register in the least significant bit position.

```
function training (index[], y: integer; prediction, outcome: {taken, not_taken})
begin
    if prediction ≠ outcome or |y| ≤ θ then
        for j in 0..h do
            if outcome = taken then
                W[index[j], j] := W[index[j], j] + 1
            else
                W[index[j], j] := W[index[j], j] – 1
            end if
        end for
    end if
    local := (local << 1) or outcome
    global := (global << 1) or outcome
end
```

3.5. Simulation results

In order to evaluate the prediction accuracy of global/local hashed perceptron predictor, we conduct a trace-driven simulation. We use 12 SPEC CPU 2000 integer benchmarks including *164.gzip*, *175.vpr*, *176.gcc*, *181.mcf*, *186.crafty*, *197.parser*, *252.eon*, *253.perlbmk*, *254.gap*, *255.vortex*, *256.bzip2*, *300.twolf* [12].

3.5.1. Different configurations

Figure 3 reveals the results of average misprediction rate with the number of weights in a perceptron ranging from 2 to 64. The performance increases with the number of perceptrons. However, as the number of perceptrons becomes larger, the impact of aliasing is relatively small, resulting in better performance. For any fixed number of weights in a perceptron, the global/local hashed perceptron predictor is more accurate as the number of perceptrons increases.

The performance improves when the number of weights in a perceptron ranges from 2 to 16. However, the predictor becomes less accurate when the number of weights in a perceptron further increases from 16 to 64. The best performance occurs at 16. This shows that having sufficient number of perceptrons, the number of weights in a perceptron is not necessary to be large to provide better prediction accuracy.

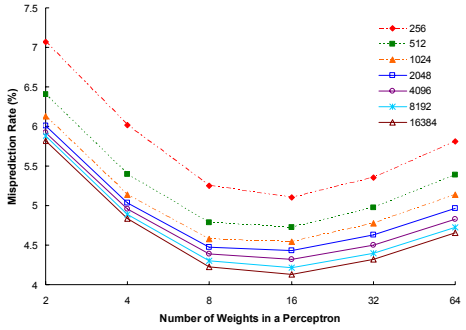

Figure 3. Misprediction rate of different configurations

3.5.2. Local history hashing vs. global history hashing

Figure 4 shows the results of average misprediction rate with increasing number of weights indexed by local history hashing. Too many or too few number of weights indexed by local history hashing as opposed to global history hashing will degrade the performance.

Note that the configuration with global history hashing alone is identical to the mapping scheme proposed by Tarjan and Skadron [5]. The simulation results shows that combining local history hashing with global history hashing gives more accurate prediction than either component alone.

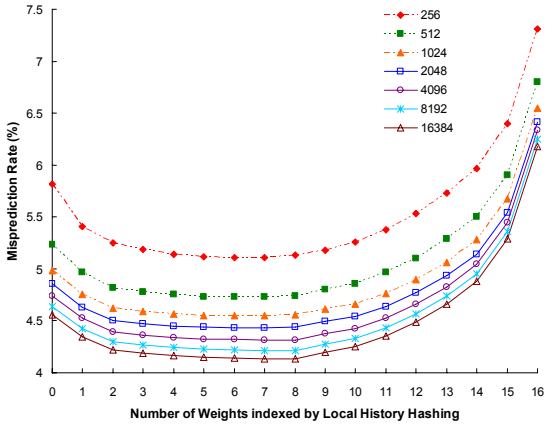

Figure 4. Misprediction rate against varied number of weights indexed by local history hashing

3.5.3. Comparison with well-known predictors

We measure the prediction accuracy according to benchmarks. We simulate the path-based neural perceptron predictor [3, 4], the global hashed perceptron predictor and the hashed perceptron predictor [5]. These predictors are the three of the most well-known perceptron predictors from the literature. They provide more superior performance than counter-based branch predictors. We present two sets of results: the first one is according to the number of perceptrons; the second one is according to the number of weights in a perceptron.

3.5.3.1 Accuracy according to Number of Perceptrons

All of the simulated predictors are configured with 16 weights in a perceptron. For the hashed perceptron predictor, since it merges path history with global history, we set the number of weights in a perceptron using path history to two. However, it is set to one when the number of weights in a perceptron is two. Figure 5 depicts the average misprediction rates against the number of weights in a perceptron.

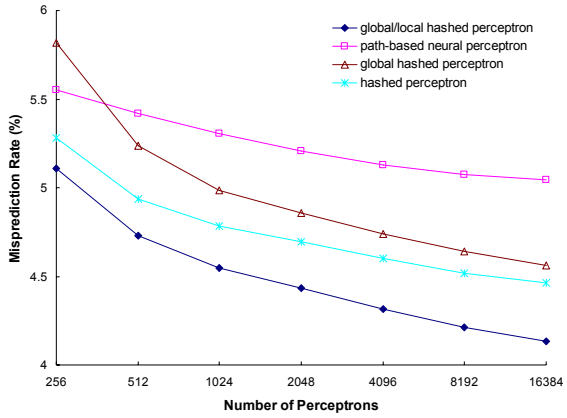

Figure 5. Comparison of misprediction rate between different perceptron predictors

In comparison with the global hashed perceptron predictor, our global/local hashed perceptron predictor makes the greatest improvement at 256 perceptrons, and our predictor improves the global hashed perceptron predictor by 12.22%.

3.5.3.2 Accuracy according to Number of Weight in a Perceptron

In this simulation, the number of perceptrons is kept to 8,192 for all of the predictors and find out the misprediction rate when number of weights in a perceptron varies from 2 to 64. The configuration of hashed perceptron predictor about the path history is the same as the previous subsection. As shown in Figure 6, our global/local hashed perceptron predictor is the most accurate one for all configurations varying from 2 to 64 weights in a perceptron.

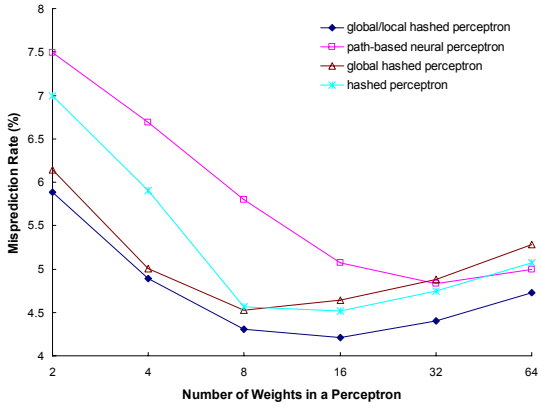

Figure 6. Comparison of misprediction rate between different perceptron predictors

4. Reducing the impact of destructive aliasing

Destructive aliasing has substantial impact on accuracy of branch prediction [9, 11, 14]. No matter that the predictors are using traditional 2-bit saturating up-down counters or artificial perceptrons, interference of branches exists among both of them. In this section, the interference of branches in perceptron branch predictors is discussed, and the impact of destructive aliasing in our global/local hashed perceptron predictor is explored.

4.1. Interference of branches in perceptron branch predictors

In counter-based branch predictors, interference is defined as unrelated branches are mapped to the same counter. The idea behind the Agree predictor [7], which is an efficient scheme on reducing the impact of destructive aliasing. The structure of Agree predictor is not complicated and it is feasible to be implemented in perceptron-based branch predictors.

The training algorithm exploited is to increment the weights by 1 if actual outcome is *taken*, and decrement the weights by 1 if actual outcome is *not taken*.

4.2. Structure of global/local hashed perceptron predictor with reducing aliasing approach

A biasing bit is assigned to each branch in BTB. Branches exhibit biased behaviors [6]. The biasing bit is used to capture the most likely tendency of direction of corresponding branch. It is set to the direction of that branch the first time it is introduced into to BTB. Figure 7 shows the structure of global/local hashed perceptron predictor with reducing aliasing approach. Instead of predicting the direction of a branch directly, the output y of the indexed perceptron predicts whether or not the direction of the branch is consistent with its biasing bit.

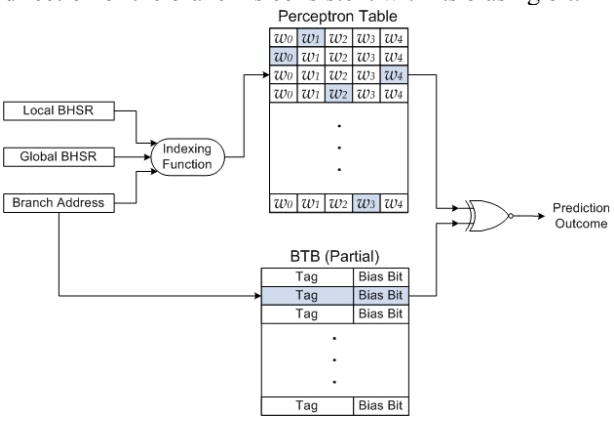

Figure 7. Global/local hashed perceptron predictor with reducing aliasing approach

4.3. Prediction algorithm

The weights of perceptron are fetched and output y of perceptron is calculated for branch instructions. If the output y of the perceptron is non-negative, the current branch is predicted to be the direction of its biasing bit. Otherwise, the opposite. Table 1 lists the possible

combinations of perceptron output *y*, biasing bit and the results of prediction outcome. The pseudocode shows the prediction algorithm. Here the *bias* represents the biasing bit of branch to be predicted.

Table 1. True table of prediction algorithm

Perceptron Output *y*	Biasing Bit	Prediction Outcome
Negative	Not Taken	Taken
Negative	Taken	Not Taken
Non-Negative	Not Taken	Not Taken
Non-Negative	Taken	Taken

```
function prediction (addr: integer): {taken, not_taken}
begin
    index[0] := addr mod n
    y := W[index[0], 0]
    for j in 1..h do
        if j ≤ length then
            index[j] := (local[j-1] ⊕ addr) mod n
            y := y + W[index[j], j]
        else
            index[j] := (global[j-length] ⊕ addr) mod n
            y := y + W[index[j], j]
        end if
    end for
    if y ≥ 0 then
        prediction := bias
    else
        prediction := not bias
    end if
end
```

4.4. Training algorithm

The update of weight values is based on the biasing bit as well as the output *y* of indexed perceptron. The weight values are incremented by 1 if the actual direction of the branch is the same as its biasing bit. Conversely, the weight values are decremented by 1. The following pseudocode describes the training algorithm.

```
function training (index[], y: integer; prediction, outcome: {taken, not_taken})
begin
    if prediction ≠ outcome or |y| ≤ θ then
        for j in 0..h do
            if outcome = bias then
                W[index[j], j] := W[index[j], j] + 1
            else
                W[index[j], j] := W[index[j], j] – 1
            end if
        end for
    end if
        local := (local << 1) or outcome
        global := (global << 1) or outcome
end
```

4.5. Advantages of reducing aliasing approach

To illustrate, we imagine two interfering branches having opposite biased behaviors, Branch A and Branch B. Branch A has the *taken* biased behavior, while Branch B has the *not taken* biased behavior. For most of the time, having reducing aliasing approach, Branch A and Branch B modify the weight values toward the same direction instead of opposite direction.

Alternatively, we illustrate the concept from the view of probability. Assuming the probability of occurrence of the most likely case to be 90% and the least likely case to be 10%,,without reducing aliasing, the global/local hashed perceptron predictor suffers from destructive aliasing when the two branches exhibit opposite direction of actual outcome. The first possible situation is that the actual outcome of Branch A and Branch B is *taken* and *not taken* respectively; while the second possible situation is reverse. The probability of destructive aliasing to occur is estimated as follows:

The probability of occurrence of destructive aliasing without reducing aliasing approach
= (Branch A's *taken* and Branch B's *not taken*) or
(Branch A's *not taken* and Branch B's *taken*)
= (90% * 90%) + (10% * 10%)
= 82%

With reducing aliasing approach, the destructive aliasing occurs if one of the two branches exhibit consistent behavior. The first possible situation is that Branch A's actual outcome is equal to its biasing bit but Branch B's actual outcome differs from its biasing bit, that is, (Branch A's *taken* and Branch B's *taken*). The second possible situation is Branch A's actual outcome does not agree with its biasing bit but Branch B's actual outcome does. The estimated probability of destructive aliasing is: (90% * 10%) + (10% * 90%) = 18%

4.6. Simulation results

4.6.1. Different configurations

The average misprediction rates of global/local hashed perceptron predictor with reducing aliasing approach are presented in Fig. 8. Compared with Fig. 3, having reducing aliasing approach the trend of overall performance is similar to the original global/local hashed perceptron predictor. The misprediction rates of all curve decrease when the number of weights in a perceptron varies from 2 to 16, whereas they increase between 16 and 64. The predictor performs the best with 16 weights in a perceptron at a fixed number of perceptrons. The performance is more sensitive to the number of perceptrons rather than the number of weights . The predictor continues to perform better as the number of perceptrons increases.

It appears that the vertical positions of all curves in Fig. 8 are slightly lower than that located in Fig. 3. This indicates that the prediction accuracy is enhanced among almost all configurations.

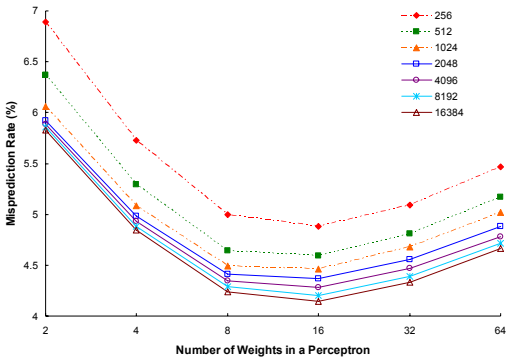

Figure 8. Misprediction rate of different configurations (with reducing aliasing approach)

4.6.2. Improvement in Prediction Accuracy

The improvement percentage of misprediction rate is revealed in Fig. 9. All of the curves have positive results except the one with 16384 weights in a perceptron.

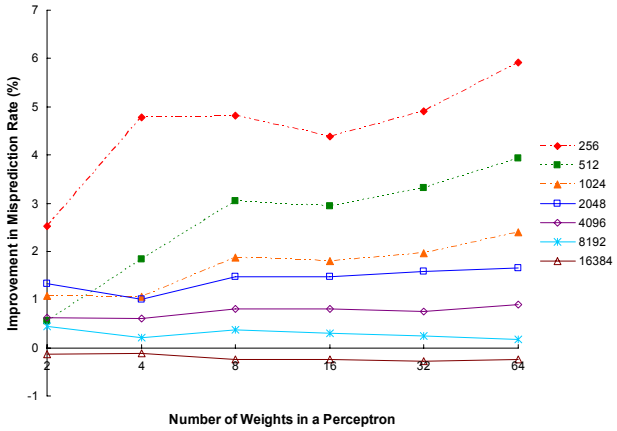

Figure 9. Improvement in misprediction rate by reducing aliasing approach

Figure 9 shows that the greatest improvement in prediction accuracy is induced when the number of perceptrons is 256. The improvement percentage decreases as the number of perceptrons increases. This behavior is consistent with what have been discussed before. The main idea of reducing aliasing is to alleviate the negative effect of interference among perceptrons.

5. Conclusions

We have presented a global/local hashed perceptron predictor, which combines local history XOR hashing and global history XOR hashing to fetch different weights of a perceptron. The simulation results prove that the prediction accuracy improves with the number of perceptrons. The ideal ratio of number of weights indexed by local history hashing to total number of weights in a perceptron should be approximately half.

In comparison with other well-known perceptron predictors, our predictor attains superior accuracy at all configurations. The simulation results show that the reducing aliasing approach improves the misprediction rate on our. The improvement is more significant when the predictor has smaller number of perceptrons.

6. Acknowledgement

The work described in this paper was partially supported by the City University of Hong Kong, Strategic Research Grant 7001847.

7. References

[1] D. A. Jiménez and C. Lin, "Dynamic Branch Prediction with Perceptrons", In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pp. 197-206, 2001.

[2] D. A. Jiménez and C. Lin, "Neural Methods for Dynamic Branch Prediction", *ACM Transactions on Computer Systems*, 20(4):369-397, 2002.

[3] D. A. Jiménez, "Fast Path-Based Neural Branch Prediction", In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 243-252, IEEE Computer Society, 2003.

[4] D. A. Jiménez, "Improved Latency and Accuracy for Neural Branch Prediction", *ACM Transactions on Computer Systems*, 23(2):197-218, 2005.

[5] D. Tarjan and K. Skadron, "Merging Path and Gshare Indexing in Perceptron Branch Prediction", *ACM Transactions on Architecture and Code Optimization*, 2(3):280-300, 2005.

[6] S. McFarling, "Combining Branch Predictors", *Western Research Laboratory Technical Note TN-36*, June 1993.

[7] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference", *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pp. 284-291, 1997.

[8] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge, "The Bi-Mode Branch Predictor", *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 4-13, 1997.

[9] T.-Y. Yeh and Y. N. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History", *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp. 257-266, 1993.

[10] S.-T. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation", *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 76-84, 1992.

[11] S. Sechrest, C.-C. Lee, and T. Mudge, "Correlation and Aliasing in Dynamic Branch Predictors", *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 22-32, 1996.

[12] SPEC CPU2000 Benchmarks, http://www.spec.org/cpu2000/

[13] T. Li, L. K. John and R. H. Bell, Jr, "Modeling and Evaluation of Control Flow Prediction Schemes Using Complete System Simulation and Java Workloads", In *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systemse*, pp. 391-400, 2002.

[14] C. Y. Ho and Anthony S. S. Fong, "Combining Local and Global History Hashing in Perceptron Branch Prediction", *Proceedings of the 6th IEEE International Conference on Computer and Information Science*, Melbourne, Australia, July 2007.