Designing an Advanced Simulator for Unbiased Branches' Prediction

Adrian Florea[1], Ciprian Radu[1], Horia Calborean[1], Adrian Crapciu[1], Arpad Gellert[1] and Lucian Vintan[1]

[1] *"Lucian Blaga" University of Sibiu, Computer Science Department, E. Cioran Street, No. 4, Sibiu-550025, ROMANIA, Tel./Fax: +40-269-212716, E-mail: adrian.florea@ulbsibiu.ro, {radu_ciprianro, horia.calborean, adrian.crapciu}@yahoo.com, arpad.gellert@ulbsibiu.ro, lucian.vintan@ulbsibiu.ro*

***Abstract* — In this paper we continue our work on detecting and predicting unbiased branches. We centered on two directions: first, based on a simple example from Perm – Stanford benchmark, we show that extending context information some of branches in certain contexts became fully biased, thus diminishing the frequency of unbiased branches at benchmark level. Second, we use some state-of-the art branch predictors to predict the unbiased branches. Following our aims, we developed the ABPS tool (Advanced Branch Prediction Simulator), an original useful simulator written in Java that performs trace-driven simulation on 25 benchmarks from Stanford and SPEC suites.**

***Keywords*—** unbiased branches, neural predictors, trace-driven simulation, benchmarking.

I. INTRODUCTION

The branch prediction becomes a challenge problem for processors' designers. Without performing branch prediction (BP) it won't be possible to aggressively exploit program's instruction level parallelism. All present branch prediction techniques are limited in their accuracy. An important limitation cause is given by the used prediction contexts (global and local histories respectively path information). Using these dynamic contexts, some branches are unbiased and non-deterministically shuffled, therefore unpredictable. The percentages of these branches represent a fundamental prediction limitation. One of our goals is to demonstrate the insufficiency of global correlation information (many times the global history is too short and doesn't keep the really correlated branches with the predicted one, retaining quite enough noise). If the context would permit it could be seen a correlation between branches situated at a large distance in the dynamic instruction stream. Also, the local correlation reduces the noise included in global history. Another aim of our work is to use some state-of-the art neural branch predictors (simple and fast path-based perceptron) to predict the unbiased branches.

The organization of the rest of this paper is as follows. In section II we review related work in the field of branch prediction. Section III shows, based on a simple example from *Perm* – Stanford benchmark, the influence of different length context information (global and local) on unbiased branches. In section IV is presented the software design of ABPS. Section V includes simulation methodology and experimental results obtained using the developed ABPS simulator. Finally, section VI suggests directions for future works and concludes the paper.

II. RELATED WORK

The nowadays processors use hybrid prediction structures, combining two (or more) two-level adaptive predictors [1], one correlated with local history of the predicted branch (PAg predictor) and other correlated with global history of the predicted branch (GAg predictor). The selection between two predictions is made using a confidence table that records the dynamic behavior of each predictor. The processor Alpha 21264 embeds a hybrid predictor having a local predictor with 1024 entries (keeping a local history of 10 bits) and a global predictor with 4096 entries reaching to almost 95% of prediction accuracy [2].

The most accurate single-component branch predictors in the literature are neural branch predictors [2, 3, and 4]. Their main advantages consist in possibility of using longer correlation information at linear cost. The *Perceptron* predictor – the simplest neural branch predictor – keeps a table of *weights vectors* (small integers that are learned through the *perceptron* learning rule) [2]. As in global two-level adaptive branch prediction, a shift register records a global history of outcomes of conditional branches, recording *true* for *taken*, or *false* for *not taken*. To predict a branch outcome, a weights vector is selected by indexing the table with the branch address modulo the number of weights vectors. The dot product of the selected vector and the global history register is computed, where *true* in the history represents 1 and *false* represents -1. If the dot product is at least 0, then the branch is predicted taken, otherwise it is predicted not taken. Once the perceptron output has been computed, the training algorithm starts: it increments the *i*-th correlation weight when the branch outcome agrees with the *i*-th bit from global branch history shift register and decrements the weight otherwise. Unfortunately, the high latency of the perceptron predictor and impossibility to predict the linearly inseparable branches makes it impractical yet for hardware implementation. In order to reduce the prediction latency, the *Fast Path-based Perceptron* [3] chooses its weights for generating a prediction according to the current branch's path, rather than according to the branch's PC and history register. The prediction latency is hidden due to the speculative calculation of the perceptron's output. *Intel Co* includes the perceptron predictor in one of its IA-64 simulators for researching future microarchitectures [2]. The piecewise linear branch predictors [4] use a piecewise-linear function for a given branch, exploiting in this way different paths that lead to the same branch in order to predict – otherwise linearly inseparable – branches.

In [5] the authors proposed a hybrid scheme that employs two Prediction by Partial Matching (PPM) Markovian predictors, one that predicts based on local branch histories and one based on global branch histories. The two independent predictions are combined using a simple hardware feasible perceptron.

Vintan et al. proved that a branch in a certain dynamic context is difficult-to-predict if it is unbiased and the outcomes are shuffled [6]. In other words, a dynamic branch instruction is unpredictable with a given prediction information if it is unbiased in the considered dynamic context and the behavior in that certain context cannot be modeled through Markov stochastic processes of any order.

III. UNBIASED BRANCH PREDICTION – A CHALLENGE PROBLEM FOR PROCESSORS' DESIGNERS

A. Analyzing Branch Prediction Contexts Influence

In this section we analyze the present day branch prediction used contexts (global and local histories respectively path information) from the point of view of their limits in predicting unbiased branches. The main idea is: in a perfect dynamic context all branch instances should have the same outcome. If the outcome is not the same a first solution might consists in extending the context information. We vary the contexts length and observed that some of dynamic contexts remained unpredictable despite of their length.

In the following we present partially the C and assembly code of Stanford *Perm* benchmark that generates a suite of permutations. We detect unbiased branches and we focused on two of the most important branch instructions (having PC=35 and PC=58 after compiling process).

```
/***********************************************/
Permute (int n){
 int k;
 pctr = pctr+1;
 if(n != 1) # the first branch instruction analyzed (PC=35)
 {
  Permute(n-1);
  for( k = n-1; k >= 1; k--)# the second branch instruction
                                    analyzed (PC=58)
  {
       Swap(&permarray[n], &permarray[k]);
       Permute(n-1);
       Swap(&permarray[n], &permarray[k]);
  };
 }
}
/***********************************************/
_Permute:
       SUB SP, SP, #128
       ST 0(SP), RA
       ST 8(SP), R17
       ST 12(SP), R18
       ST 16(SP), R19
       ST 20(SP), R20
       MOV R20, R5
       LD R13, _pctr
       ADD R13, R13, #1
       ST _pctr, R13
       EQ B1, R20, #1
       BT B1, L8 (#0)   # after compiling process this
                   branch has the address 35 (PC=35)
       ADD R17, R20, #-1
       MOV R5, R17
       BSR RA, _Permute (#0)
       MOV R18, R17
       LES B1, R18, #0
       BT B1, L8 (#0)
       ASL R13, R20, #2
       MOV R7, #_permarray
       ADD R19, R13, R7
       ASL R13, R18, #2
       ADD R17, R13, R7
L12:
       MOV R5, R19
       MOV R6, R17
       BSR RA, _Swap (#0)
       ADD R5, R20, #-1
       BSR RA, _Permute (#0)
       MOV R5, R19
       MOV R6, R17
       BSR RA, _Swap (#0)
       ADD R17, R17, #-4
       ADD R18, R18, #-1
       GTS B1, R18, #0
       BT B1, L12 (#0)  # after compiling process this
                  branch has the address 58 (PC=58)
**************************************************
```

In the following simulations the settled parameters are: *Path* = not selected, *Unbiased polarization degree* = 0.95, HRl and HRg being the local and global history.

We define polarization index (***bias***) of a certain branch context as: $\text{bias} = \max(\frac{T}{T+NT}, \frac{NT}{T+NT})$, where T and NT represent number of "*taken*" respective "*not taken*" branch instances corresponding to that certain context.

1. Parameters: **HRl = not selected, HRg on 3 bits**, => Unbiased contexts: **25.0**[%]
From the unbiased branches list we selected just two branch instructions in two global contexts:
PC: 35 HRg: 101 T: 2520 NT: 1100 Bias: 0.696
PC: 58 HRg: 111 T: 1419 NT: 3620 Bias: 0.718

2. Parameters: **HRl = not selected, HRg on 4 bits**, => Unbiased contexts: **17.813**[%]
PC: 35 HRg: **0**101 T: 840 NT: 260 Bias: 0.763
PC: 35 HRg: **1**101 T: 1680 NT: 840 Bias: 0.667
PC: 58 HRg: **0**111 T: 1419 NT: 1100 Bias: 0.563
PC: 58 HRg: **1**111 T: 0 NT: 2520 Bias: 1.000 **=> *The branch with the address PC: 58 in context HRg: 1111 became fully biased.*** Practically it doesn't appear in the unbiased branch list.

3. Parameters: **HRl on 1 bit, HRg on 4 bits,** => Unbiased contexts: **17.813**[%]
PC: 35 HRg: **0**101 HRl: **0** T: 840 NT: 260 Bias: 0.763
~~PC: 35 HRg: 0101 HRl: 1 – this context doesn't occur~~
PC: 35 HRg: **1**101 HRl: **0** T: 1680 NT: 840 Bias: 0.667
~~PC: 35 HRg: 1101 HRl: 1 – this context doesn't occur~~
PC: 58 HRg: **0**111 HRl: **0** T: 1419 NT: 1100 Bias: 0.563
~~PC: 58 HRg: 0111 HRl: 1 – this context doesn't occur~~

4. Parameters: **HRl on 2 bits, HRg on 4 bits**, => Unbiased contexts: **9.673**[%]
PC: 35 HRg: 0101 HRl: **0**0 T: 840 NT: 260 Bias: 0.763
~~PC: 35 HRg: 0101 HRl: 10 – this context doesn't occur~~
~~PC: 35 HRg: 1101 HRl: 00 – this context doesn't occur~~
PC: 35 HRg: 1101 HRl: **1**0 T: 1680 NT: 840 Bias: 0.667
PC: 58 HRg: **0**111 HRl: **0**0 T: 1419 NT: 260 Bias: 0.845
PC: 58 HRg: **0**111 HRl: **1**0 T: 0 NT: 840 Bias: 1.000**=>** ***The branch with the address PC: 58 in context HRg: 0111 and*** HRl: 10 ***became fully biased.*** Practically it doesn't appear in the unbiased branch list.

…

5. Parameters: **HRl on 2 bits, HRg on 7 bits,** => Unbiased contexts: **9.668**[%]
PC: 58 HRg: 1110111 HRl: **0**0 T: 1419 NT: 260 Bias: 0.845

6. Parameters: **HRl on 2 bits, HRg on 8 bits,** => Unbiased contexts: **8.134**[%]
PC: 58 HRg: **0**1110111 HRl: **0**0 T: 579 NT: 260 Bias: 0.690
PC: 58 HRg: **1**1110111 HRl: **0**0 T: 840 NT: 0 Bias: 1.000=> ***The branch with the address PC: 58 in context HRg: 11110111 and*** HRl: ***00 became fully biased.*** Practically it doesn't appear in the unbiased branch list.

Conclusion: As it can be observed, *increasing the context length, some branches in certain contexts became fully biased, but a great percentage still remains unbiased.*

Comparing the previous results it can be observed that as longer (increase the history length) or richer (local history it is added beside global history) became the context as smaller became the unbiased branches percentage. From the 1st case to 2nd one, the unbiased branches percentages decrease with 7.187% and it can be observed how the two unbiased branches in small contexts are still unsolved. However, the branch with the address **PC: 58** in context **HRg: 1111** became fully biased and decrease the number of unbiased branches with 2520. Practically it doesn't appear in the unbiased branch list. In the 3rd case (adding one bit of local history) the unbiased branches percentage remains unchanged**.** In the 4th local history is set on 2 bits and much more contexts became biased (the unbiased branches percentage decreases with 8.14%). Although, there are some contexts that remain unbiased (see above: **PC: 35 HRg: *x*101 HRl: *x*0** – where ***x*** could be 0 or 1).

Analyzing the code sequence it can be said that the results regarding to unbiased branches are correct. It can be observed that to reach conditional branch 58, the previously 3 branches are every time Taken (return from *permute* function, call of *swap* function and return from it – not necessarily correlated with the branch 58). One reason for the larger percentage of unbiased branches refers to the fact that the branches within the global history length may not have correlation with the current branch, or the relevant history might be too far away. If the context would permit it could be seen a correlation between branches situated at a large distance in the dynamic instruction stream. Recurrence and function calls hide some branches that are really correlated with the analyzed one. Also, the local correlation reduces the noise included in global history. Similar examples we found in *tower* benchmark that solves the *Hanoi towers* problem.

The insufficiency of global correlation information is remarked also in the case of programs or data structures, which produce a variable number of history bits as the data changes (data correlation). This occurs in the link lists or trees cases where it is tested the address of an element (usually comparison with 0) and then follow a recurrent call of the same function to test the next element in the tree (left or right sub-tree). The same situation it happen in the *hash table* cases having link lists to solve the collisions. A possible solution could be to use data values or structural information to keep the predictor more synchronized with data. We tried such an approach in [7].

B. Predicting Unbiased Branches using State-of-the-art Predictors

The prediction process supposes accessing the tables for every instruction from traces and establishing the prediction function of associated prediction automaton or perceptron computed output. After branch's resolution, it starts the updating algorithm (every good prediction increase the automatons state or perceptron weights, otherwise decreasing the same parameters). The automatons are implemented as saturating counters and, in the neural predictors' case, the threshold keeps from overtraining, permitting the perceptron to adapt quickly at every changing behavior.

ABPS includes the following predictors implemented: GAg, PAg, PAp, GShare and Perceptron. The two-level predictors implemented (first 4) request as inputs parameters: number of entries in prediction table, the history length (global / local). Besides input parameters used by two-level predictors, the neural predictors (*Simple Perceptron* and *Fast Path-based Perceptron*) need some additional: threshold value used for learning algorithm, number of bits for storing the weights. Each predictor can predict all branches or only unbiased branches.

If the user selected the *Perceptron* predictor (*Simple* or *Fast Path-based*), the simulation results consist in four important metrics. The **prediction accuracy** is the number of correct predictions divided to total number of dynamic branches. We compute also a **confidence** metric that represents the total cases when the prediction was correct and the perceptron didn't need to be trained (the magnitude of perceptron output was greater than threshold) divided to the total number of correct predictions. While the first two have impact on processor's performance, the next two metrics have direct influence on transistors' budget and integration area (the **number of perceptrons used** in prediction process and respectively the **saturation degree** of perceptrons). The saturation degree represents the percentage of cases when the weights of perceptrons can't be increased / decreased because they are saturated. If the last two metrics are quite low means that the perceptrons are underused. The **prediction accuracy** and the usage degree of prediction table are computed also in the case of classical two-level predictors.

IV. SOFTWARE DESIGN OF ABPS SIMULATOR

The **user diagram** (Fig.1a) illustrates the general user interaction process with ABPS. A generic user can mainly interact with ABPS in two ways (not fully distinct):

- *Default start* -> the user starts a simulation using the default input parameters.
- *Custom start (Choose simulation type)* -> the user chooses:
  1. The simulation type – detection or prediction;
  2. The benchmarks (Stanford and/or SPEC 2000);
  3. The values for the simulation parameters.

After the three steps presented above, the user can start the simulation process. Both in the *Default start* and in the *Custom start* cases, after the simulation process is ready, simulation results are shown.

NOTES: Steps 1, 2, 3 can be executed in any order. Either of steps 1 and 3 is not mandatory. If one of them is not executed, default values are used. Step 2 (choosing the benchmarks) is necessary the first time (initially no traces are selected for simulation) for both user interaction types.

The **activity diagram** (Fig.1b) shows a general view for the simulation process flow in ABPS:

- *Initialization* – all simulation parameters are set (traces, simulation type: detection / prediction, detector / predictor values);
- *Starts simulation* – the simulation begins after all the inputs had been set. The simulation process consists basically in processing each trace included (in a multithreaded manner);
- *Read trace* – each trace is processed, branch after branch. Each branch instruction is fed to the selected detector / predictor. This is done until all branch instructions (from the selected trace) are processed. During this, results are accumulated.
- *Processing results* – after a trace had been processed, the obtained results are processed in order to compute certain metrics;
- *Display results* – the results are displayed and the simulation process stops.

NOTE: At any time the simulation process can be aborted from the GUI (Graphic User Interface).

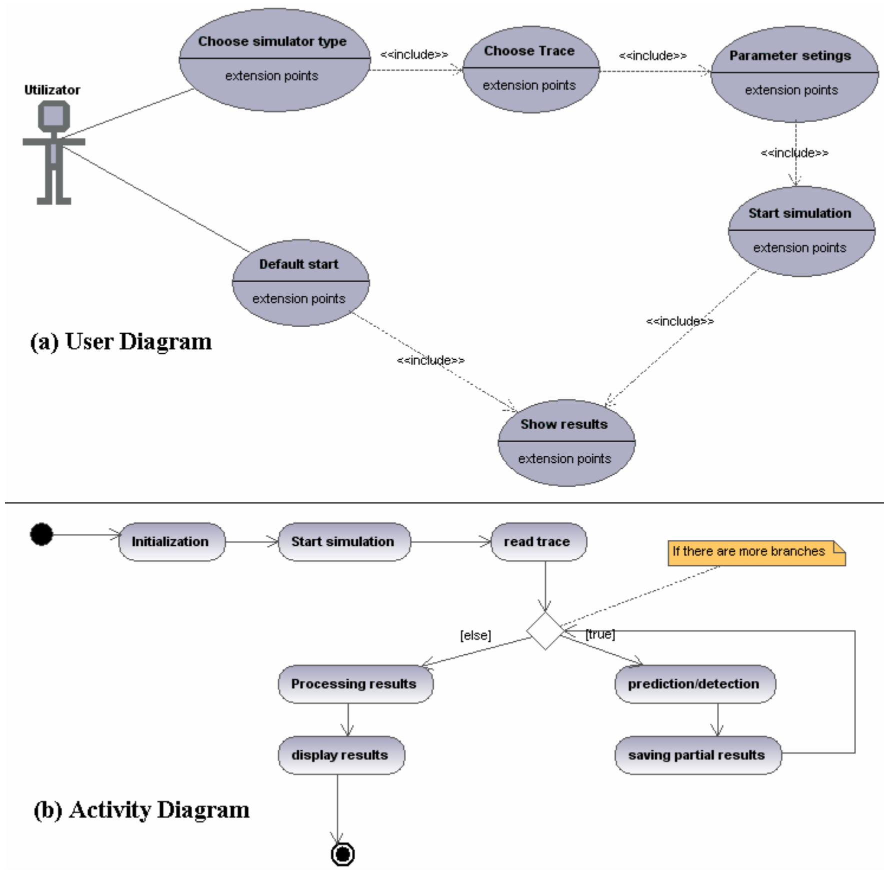

Figure 1. UML Diagrams – *User* and *Activity* perspectives

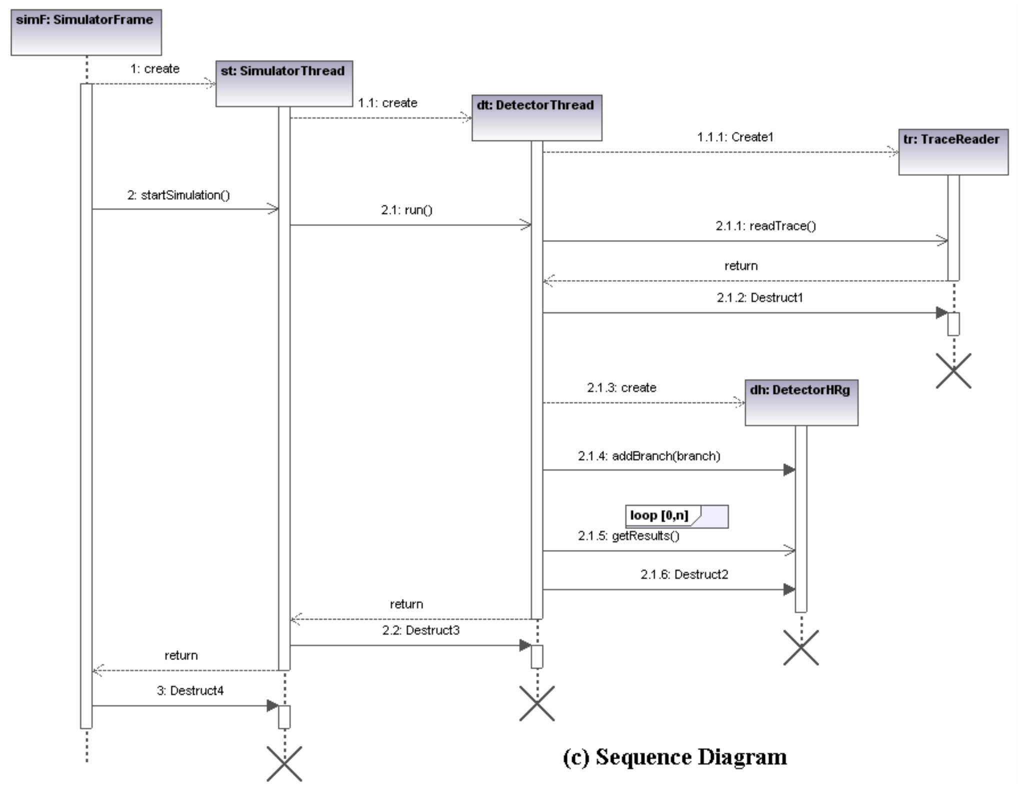

Figure 2. Sequence Diagram

The **sequence diagram** (Fig.2) presents in detail how ABPS performs the process of detecting unbiased branches. The process starts in the GUI, where the detection parameters are set. After this initialization, the user can trigger the detection process, which will be managed by another thread (***1: create, st:SimulatorThread***). In this way, the GUI will not block itself, leaving the user with the ability to perform other tasks from ABPS. The simulation thread will create and start a detection thread (**1.1: create, dt:DetectorThread**). The detection thread will manage all the detection process (**1.1.1: Create1, tr:TraceReader**). When all the above initializations were performed, the detection process actually starts (**2: startSimulation(), 2.1: run()**): the trace used for simulation is processed using the appropriate detector (see: **2.1.1 – 2.1.6**). Finally, the detection thread signals (by returning the results) the simulation thread that the detection is done (2.2: Destruct3). In the same manner, the simulation thread signals the GUI thread (**3:Destruct4**), which will display the results.

NOTE: Although the above diagram doesn't show, at any time the detection process can be aborted from the GUI.

V. SIMULATION METHODOLOGY AND EXPERIMENTAL RESULTS

We developed ABPS (Advanced Branch Prediction Simulator) an original interactive graphical trace-driven simulator [8]. We simulate eight C Stanford integer benchmarks, designed by Professor John Hennessy (Stanford University), to be computationally intensive and representative of non - numeric code while at the same time being compact. Also, we simulate all of the SPEC CPU2000 integer benchmarks, and all of the SPEC CPU95 integer benchmarks that are not duplicated in SPEC CPU2000, each benchmark having 1 million dynamic branch instructions. All these benchmarks cover a lot of applications ranging from compression (text/image) to word processing, from compilers and architectures to games enhanced with artificial intelligence, etc. We choose to simulate different version of benchmarks (Stanford and SPEC) in order to discover how these different testing programs influence the neural branch predictors' micro-architectural features.

The ABPS simulator provides a wider variety of configuration options. Thus, it can be determined how vary prediction accuracy with input parameters (number of entries in prediction tables, history length, number of bits for weights representation, threshold value used for perceptron training, etc). ABPS is written in Java and assures three of the features specific to almost high-performance standard simulators: free availability for use, extensibility and portability. Full inheritance and polymorphism is used, allowing for ease of extension in the future adding new functionalities.

Repeating the detection methodology for a length-ordered set of contexts it could be observed how decreases the number of unbiased branches from tested benchmarks. Figure 3 shows the reduction in the number of unbiased branches varying the length of prediction contexts from 8 to 32 bits. The percentage reduction in the number of unbiased branches decreases from **25.12%** to **9.26%.** We consider that the last value is too high and further investigations are required.

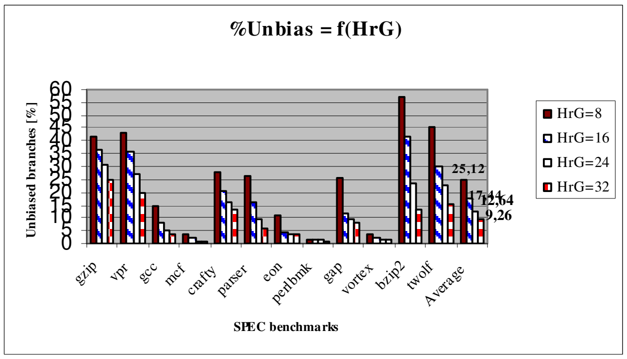

Figure 3. Reducing the number of unbiased branches with increasing global history register length

Figure 4 graphical illustrates the influence of global history on prediction accuracy using a fast path-based perceptron predictor. It is very clear that as longer became the global history length as greater became the prediction accuracy on all branches. Also, the prediction accuracy ascendant trend still remains when the number of perceptrons increases. The best prediction accuracy obtained with a fast path-based perceptron predictor – 95.21%, is superior to that provided by Alpha 21264, but having a hardware budget of 8th times smaller (≈32Kbytes vs. ≈257Kbytes).

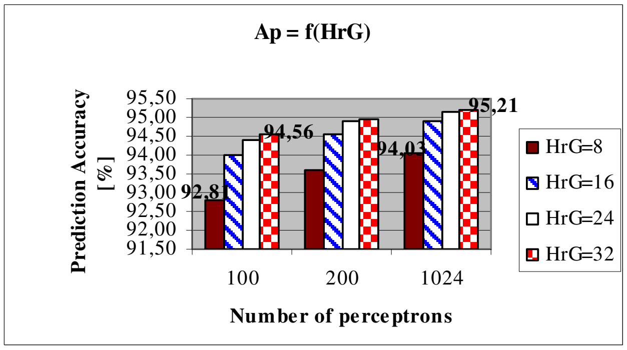

Figure 4. The influence of global history on prediction accuracy using a fast path-based perceptron

Despite of significant reduction of unbiased branches percentages (in average **26.79%**) on five of SPEC benchmarks (gzip, vpr, parser, bzip2 and twolf) the prediction accuracy varies asymptotically (under 1.30% in average) whether global history length raises from 8 to 32 bits (see figure 5). We named these SPEC testing programs as critical benchmarks. The average prediction accuracy on these benchmarks is very low (**91.06%** – see figure 5). When global history length is 32 bits the unbiased branches percentage on the 5 critical benchmarks is still high (in average **15.90%**) and may be responsible for the lower prediction accuracy. This is because the current amount of prediction information is limited (global-correlations). The use of such limited information means that unbiased branches cannot be predicted to a high degree of accuracy. Consequently, other information is required to predict branches which have been classified as unbiased (local, path or sign condition).

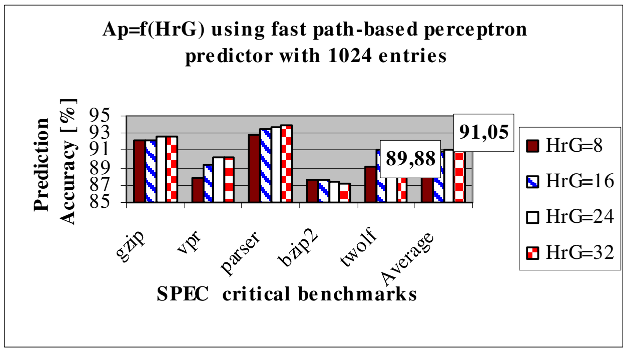

Figure 5. Prediction accuracy on SPEC critical benchmarks

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have shown that the design of branch predictors should consider the identification of unbiased branches due to their negative impact on prediction accuracy. Repeating the detection methodology for a length-ordered set of contexts (varying the global history length from 8 to 32 bits) it could be observed that the percentage of unbiased branches decreases from 25.12% to 9.26% but still remains a quite significant percentage of unbiased branches. Further, we have demonstrated the insufficiency of global correlation information. We have therefore shown that even state of the art branch predictors are unable to accurately predict these unbiased branches (the best prediction accuracy measured on all branches using a fast path-based perceptron predictor is 95.21%). We therefore consider that the use of more prediction contexts (some HLL code information) is required to further improve prediction accuracies. In order to efficiently use such information we consider it will be necessary to have a significant amount of compiler support.

REFERENCES

[1] Yeh T., Patt Y., *Alternative Implementations of Two-Level Adaptive Branch Prediction*, 19th Annual International Symposium on Computer Architecture, 1992.

[2] Jiménez D., Lin C., *Neural Methods for Dynamic Branch Prediction*, ACM Transactions on Computer Systems, Vol. 20, New York, USA, November 2002.

[3] Jiménez D., *Fast Path-Based Neural Branch Prediction*, Proceedings of the 36th Annual International Symposium on Microarchitecture, December 2003.

[4] Jiménez D., *Idealized Piecewise Linear Branch Prediction*, Journal of Instruction-Level Parallelism, Vol. 7, pp. 1-11, (2005).

[5] Srinivasan R., Frachtenberg E., Lubeck O., Pakin S., Cook J., *Neuro-PPM Branch Prediction,* The 2nd Journal of Instruction-Level Parallelism Championship Branch Prediction Competition (CBP-2), Orlando, Florida, USA, pp. 30-35, (2006).

[6] Vintan L., Gellert A., Florea A., Oancea M., Egan C., Understanding Prediction Limits through Unbiased Branches, Lecture Notes in Computer Science, vol. 4186-0480, Springer-Verlag, ISSN 0302-9743, Berlin Heidelberg, (2006), pp. 480-487.

[7] Gellert A., Florea A., Vintan M., Egan C. Vintan L., *Unbiased Branches: An Open Problem*, The 12th Asia-Pacific Computer Systems Architecture Conference (ACSAC 2007), Seoul, Korea, August 2007.

[8] Radu C., Calborean H., Crapciu A., Gellert A., Florea A. – *An Interactive Graphical Trace-Driven Simulator for Teaching Branch Prediction in Computer Architecture*, The 6th EUROSIM Congress on Modeling and Simulation, 2007, Ljubljana, Slovenia.