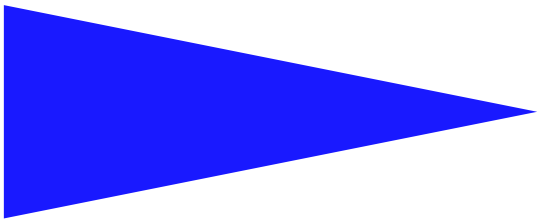
I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
1620



REVISITING THE PERCEPTRON PREDICTOR

ANDRÉ SEZNEC

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Revisiting the perceptron predictor*

André Seznec

Systèmes communicants
Projet CAPS

Publication interne n ° 1620 — Mai 2004 — 21 pages

Abstract: The perceptron branch predictor has been recently proposed by Jiménez and Lin as an alternative to conventional branch predictors. In this paper, we build upon this original proposal in three directions.

First, we show that the potential accuracy that can be achieved by perceptron-like predictors was largely underestimated. The accuracy of the redundant history skewed perceptron predictor (RHSP) we are proposing is significantly higher than the one of the original perceptron predictor. The RHSP predictor combines pseudo-tagging, i.e., using address bits as part of the input vectors, use of a redundant representation of the history (4 counters per history bit) and skewing, i.e., use of split tables indexed with different hashing functions.

Second, we drastically simplify the hardware logic needed for prediction computation and predictor update. We introduce the MAC representation (for Multiply-Add Contribution) for the perceptron counters in the RHSP predictor. This new representation allows to reduce the width of the multiply-accumulation tree in the RHSP predictor by a factor 16. The width of the represented values can also be reduced from 8 bits to 6 bits without significant accuracy loss on the MAC-RHSP predictor.

Response time is a major obstacle for the use of perceptron predictors in real processors. The third contribution of this paper is to propose an ahead pipelined MAC-RHSP predictor. Even assuming that the prediction is initiated 10-block ahead the instruction flow, the accuracy of the ahead pipelined MAC-RHSP predictor remains in the same range as the one of the hypothetical 1-cycle MAC-RHSP predictor. Moreover the volume of information to be checkpointed with each fetch block to allow smooth restart with no extra misprediction penalty remains limited.

(Résumé : tsvp)

* This work was partially supported by Intel

Centre National de la Recherche Scientifique (UMR 6074) Université de Rennes 1 – Insa de Rennes
Institut National de Recherche en Informatique et en Automatique – unité de recherche de Rennes

Une revisite du prédicteur à perceptrons

Résumé : Dans ce rapport, nous proposons un nouveau prédicteur de branchement: le le prédicteur MAC-RHSP (Multiply-Accumulate Contribution Redundant History Skewed Perceptron Predictor).

1 Introduction

Recently, a completely new approach to branch prediction was proposed by Jiménez and Lin [3, 4]. They proposed to use perceptrons, a technique coming from artificial neurons for predicting branches. The perceptron predictor allows to capture correlation on very long histories. However, simulations illustrating this paper shows that the global history perceptron predictor [3] fails to match the accuracy of optimized version of the *2bcgskew* predictor [8] on many applications.

In this paper, we build upon the seminal work by Jiménez and Lin and improve the perceptron predictor in three directions.

As a first contribution, we propose several orthogonal enhancements for the perceptron predictor to increase its accuracy. Pseudo-tagging is introduced to decrease aliasing impact on the perceptron table. Using a redundant representation of the history also allows to improve the perceptron predictor accuracy: up to 4 bits are used to represent a single branch. Skewing the perceptron predictor, i.e. splitting storage tables and indexing the different tables with different hashing functions, is also beneficial. These three techniques can be applied considering global history as well as global/local history.

Combining these three techniques, the redundant history skewed perceptron predictor (RHSP) using only global history is more accurate than the optimized version of *2bcgskew* predictor [8], and also the local/global history perceptron predictor. Combining local and global history on the RHSP predictor is shown to bring limited accuracy return compared with using only global history.

The hardware complexity of the multiply-add tree for prediction computation and of the counter update logic is a major obstacle for the use of perceptron predictors in real processors. The use of a 4 times redundant history on the RHSP predictor leads to use 4 times more counters per perceptron and therefore the use of a 4 times wider multiply-accumulation tree. As a second contribution of the paper, we introduce a new representation of the perceptron counters for redundant history perceptron predictors, the MAC representation (for Multiply-Add Contributions). With the MAC representation of the perceptron weights, one accesses one counter per block of 4 history bits instead of 16 counters. Using the MAC representation instead of the conventional counter representation allows to reduce the number of entries of the adder tree in the RHSP predictor by a factor 16. Moreover the width of represented values can be reduced from 8 bits to 6 bits without sacrificing prediction accuracy. On a 64-input MAC-RHSP predictor, a 16-entry 6-bit adder tree is used instead of a 256-entry 8-bit multiply-accumulate tree if conventional weight representation was used.

The third contribution of the paper is the proposition of the ahead pipelined global history MAC-RHSP predictor. The response time of a 4N-input MAC-RHSP predictor is a major issue since it involves the read of perceptron tables followed by a N-entry adder tree. On the ahead pipelined MAC-RHSP predictor, prediction computation is initiated X-block ahead using X-block ahead address and X-block ahead history. 16 distinct predictions are computed in parallel for 16 distinct intermediate path vectors. The effective intermediate path is used to select the correct prediction in-time. The extra logic for computing the 16 distinct predictions is limited to 15 extra 10-bit adders. Even assuming that the prediction is initiated 10-block ahead the instruction flow, the accuracy of the ahead pipelined RHSP remains in the same range as the one of the hypothetical 1-cycle MAC-RHSP predictor. On each cycle, the 16 distinct predictions for the 16 possible intermediate paths are

checkpointed. Thus, after a misprediction, instruction fetch can resume immediately after branch execution: for the X-1 next cycles, the checkpoint can provide the branch prediction instead of the branch predictor. This has to be contrasted with the recently proposed path based neural predictor [6] that also provides the prediction in-time, but that assumes that instruction fetch is resumed at commit time on a branch misprediction.

The remainder of the paper is organized as follows. In Section 2, we first briefly present an optimized version of the *2bcgskew* predictor [11, 10] handling very long global history that significantly outperforms the original global history perceptron predictor on most benchmarks. We then recall the principles of the perceptron predictor introduced by Jiménez and Lin [3, 4]. We also recall the recent proposed solutions to provide branch prediction in-time for instruction fetch while using complex branch predictors. In Section 3, we present our experimental framework for simulation and evaluation of the branch predictors Section 4 presents the enhancements we propose for the perceptron predictor scheme and evaluates their respective as well their combined benefits. Section 5 introduces the MAC representation and the MAC-RHSP predictor. Section 6 shows that the MAC-RHSP predictor can be efficiently ahead pipelined [9] in order to provide the prediction in time for use. Finally, Section 7 summarizes this study.

2 Background and related work

2.1 An optimized version of the hybrid skewed predictor *2bcgskew*

As a reference for global history conventional branch predictor, an optimized *2bcgskew* predictor will be used throughout this paper [8].

The Alpha EV8 branch predictor [11] was derived from the initial *2bcgskew* proposal. *2bcgskew* combines an *e-gskew* predictor and a bimodal predictor. *2bcgskew* consists of four 2-bit counters banks. Bank **BIM** is the bimodal predictor, but is also part of the *e-gskew* predictor. Banks **G0** and **G1** are two gshare-like predictors included in the *e-gskew* predictor. Bank **Meta** is the meta-predictor. Depending on **Meta**, the prediction is either the prediction coming out from **BIM** or the majority vote on the predictions coming out from **G0**, **G1** and **BIM**.

Optimizations were proposed for the design of the EV8 branch predictor [11]: (a) different sizes for the distinct tables in the predictor, (b) different prediction and hysteresis table sizes: prediction tables and hysteresis tables are accessed at different pipeline stages, and hence can be implemented as physically distinct tables and with distinct sizes, (c) variable history lengths: the four logical tables in the *2bcgskew* predictor are accessed using four different history lengths, i.e., the BIM table can be indexed using a short history, G0 and G1 the two gshare-like tables are indexed with respectively a “long” and a “very long” history.

The optimized *2bcgskew* predictor [8] features two extra optimizations:(d) sharing physical tables between logical tables in the predictor. Normally a physical table is associated with a logical table in the predictor, but one can share the physical tables as follows: depending on the parity of branch address (or on a bit in history), table G0 (resp. G1) will be address with “long history” (resp. “very long history”) or “very long history” (resp. “long history”). 4 physical tables may even be shared among the 4 logical tables.

(e) In order to avoid ping-pong phenomenon in the predictor. a degree of randomness in the update of the predictor is used on mispredictions, i.e., randomly, in one out of 32 mispredictions, if

the majority vote and the bimodal component are agreeing then the three predictions from G0, G1 and BIM are randomly forced to weakly good prediction, if they are disagreeing the meta predictor is forced to the correct component. With this introduction of this kind of randomness, the pathological Behaviours associated with different initializations disappear.

The configurations proposed in [8] have the following characteristics for a total of 2^N bits of storage:

- 2^{N-1} bits are shared by GO and G1 prediction tables.
- 2^{N-2} bits are shared by BIM and Meta prediction tables.
- 2^{N-2} bits are shared by the four hysteresis tables.
- History lengths are respectively set to (N-11) for BIM and Meta, 4*(N-11) for the "long history" length and 8*(N-11) for the "very long history length".

One will note that the "very long history" is much longer than in any of the previously considered global history branch predictors apart the perceptron predictor [3], 40 bits for a medium 64 Kbits predictor, 72 bits for a large 1 Mbits predictor.

2.2 Brief presentation of the perceptron predictor

Branch predictors use the correlation between the branch address and the branch or path history to predict the branch direction. Neural nets, and particularly perceptrons, are able to exploit such a correlation. Jiménez and Lin proposed the perceptron predictor illustrated on Figure 1

A table of S perceptrons is implemented. Each perceptron consists of W signed saturating counters (or weights in the neural net terminology) w_i. Using 8-bit counters was shown to be a reasonable trade-off. Values of the counters are therefore in the interval $[-128, 127]$

$H = (h_{W-1}, .., h_1)$ the branch history being the input vector, the prediction p is computed as the sign of $y = w_0 + \sum_{i=1}^{W-1}(2h_i - 1)w_i$.

The perceptron is trained as follows, *Out* being the output of conditional branch:

if ((p !=*Out*) or ($|y| < \theta$)){
for each bit in parallel
{ if *Out*= h_i then $w_i = w_i + 1$ else $w_i = w_i - 1$ }}

Threshold θ is a parameter to the perceptron training algorithm that is used to decide whether the predictor needs more training. The best threshold θ for W weights was shown to be $\theta = 1.93W + 14$.

For a complete description, the reader is referred to the original study[4].

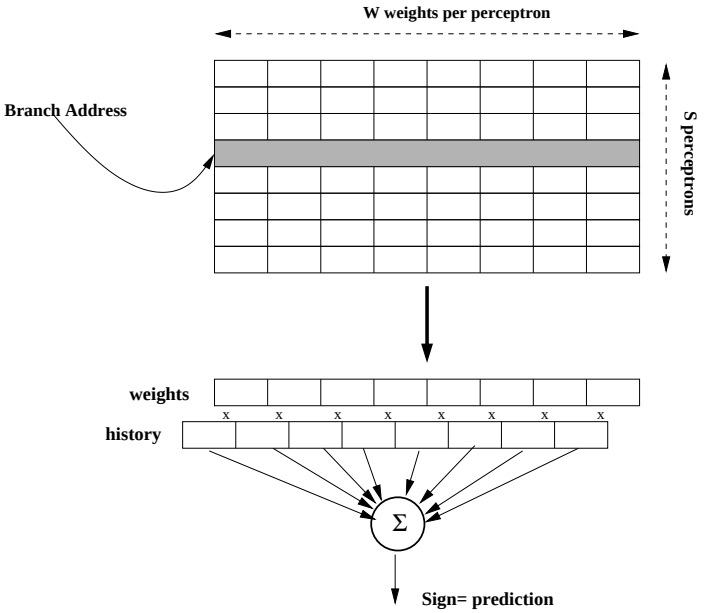


Figure 1: The perceptron predictor

2.3 Advantages and drawbacks of the perceptron predictor

Scalability The main advantage of the perceptron predictor is its scalability. The size of an individual perceptron in the perceptron predictor scales linearly with the length of the inputs (i.e, the branch history length): for a given branch B, a single perceptron is used independently of the number of different paths that converge to the branch. This has to be contrasted with the conventional two-level adaptive schemes where the effective number of entries used by a branch B is potentially equal to the number of different branch histories leading to branch B. Therefore, on these conventional predictors, increasing the history length generally leads to an increase of destructive aliasing in the predictor tables [12].

Simple global/local history perceptron predictor Combining local and global history in a conventional branch predictor generally requires the use of a metapredictor. On the global/local history perceptron predictor, the local history and the global history are concatenated to form the input vector. The multiply-accumulate tree acts as the metapredictor.

Linear separability A limitation of perceptrons is that they are only capable of learning linearly separable functions [1] (see [4]). That is, even if the behaviour of branches in a program was completely determined by the pair (history,branch), the perceptron would not able to completely learn it, no matter of the training time. By contrast a hypothetical conflict free *gshare* predictor [7] would deliver perfect prediction after training.

Prediction computation time and hardware logic complexity The prediction computation time on the perceptron predictor involves two components, the perceptron table read and the multiply-accumulate tree. The perceptron table read delay depends on the technology and the size of the

table. For large predictors, it will span over several cycles. The multiply-accumulate tree delay depends on the number of entries: for a 64 8-bit weights perceptron, the delay represents a multiply by 1 or -1, a 9-stage carry save adder tree and a final 2-entry 13-bit addition.

Hardware logic for the perceptron predictor includes the multiply-add tree and the logic for weights update, i.e., a counter update logic per input bit.

2.4 Addressing the prediction computation time

The response time of state-of-art accurate branch predictors are largely longer than a CPU cycle, since they involve two components, the response time of the read on large tables and some hardware logic. Therefore these slow accurate branch predictors are often used for overriding the prediction of a fast (but relatively inaccurate) predictor as suggested in [5].

However solutions to provide in-time branch predictions were proposed in [9] and [6]. Both solutions rely on initiating the prediction ahead the instruction flow. Initiating a branch prediction ahead the instruction flow faces two issues described in [9]: an accuracy issue and a misprediction penalty issue.

When only the X-block or X-branch ahead information is used for the prediction, the accuracy of the predictor significantly decreases when X grows, therefore some intermediate path information must be used to enable high accuracy.

But incorporating intermediate information in the prediction computation leads to the following misprediction penalty issue. If the latency of the branch predictor is X cycle then, after a misprediction or on an interruption on block B, the predictions for (X-1) subsequent fetch blocks can not be computed in time by the branch predictor. Therefore one must either wait for the branch predictor to resume the prediction with the correct path information (and suffer an extra X-1 cycles penalty) or rely on some back-up mechanism to provide the branch predictions for the (X-1) next blocks.

Path Based Neural Predictor In [6], Jiménez proposed the Path-Based Neural predictor to reduce branch prediction computation time on perceptron-like predictor. On a N-branch Path-Based Neural predictor, the prediction for a branch is initiated N-branch ahead. The predictions for the N next branches are computed in parallel. A row of N counters is read using the current instruction block address. On blocks featuring a branch, one of the read counters is added to each of the N partial sums. Therefore, the delay between the instruction block address availability and the availability of the prediction is the perceptron table read delay followed by a single multiply-add delay.

The Path-based neural predictor does not address the table read delay. On a misprediction, the predictor logic must be reinitialized with the correct partial sums. However the volume of these partial sums is too large to be checkpointed to allow to resume instruction fetch immediately after branch direction computation (e.g. on a 64-input Path-Based Neural Predictor, one would need to checkpoint 64 13-bit counters). Then Jiménez [6] proposed to resolve the mispredictions at commit time through maintaining a non-speculative set of partial sums. Unfortunately, most modern out-of-order execution superscalar processors resume instruction fetch after branch execution, since commit time often occurs several cycles after branch execution.

Ahead pipelined *2bcgskew* Even on predictors involving only limited computation logic, the delay for reading the predictor tables is longer than a CPU cycle (e.g. Alpha EV8 [11]). Ahead pipelining

the conditional branch predictor was studied in details for a *2bcgskew* predictor by Seznec and Fraboulet in [9]. They showed that when initiating the read of predictor tables X-cycle ahead, one can incorporate intermediate path information in the table indices and therefore obtain in time prediction and accuracy close to an hypothetical one-cycle ahead *2bcgskew* predictor.

In order to resume to handle mispredictions without paying extra mispenalty, all the information needed to recompute the branch predictions during the X intermediate cycles is checkpointed. On a misprediction, one can resume instruction fetch just after branch execution. The volume of checkpoint information needed to allow smooth restart on a misprediction remains limited.

3 Evaluation framework

In order to compare the accuracy of branch predictors considered in this paper, simulations were run. Since this study is primarily concerned with the potential accuracy of different branch predictors, trace driven conditional branch simulations with immediate updates were used. On a real hardware processor, the effective update is performed late in the pipeline, at misprediction resolution or at commit time. However, for branch predictors using very long global branch history and particularly for the perceptron predictor, the differences of accuracy between a delayed updated predictor and an immediately updated predictor are known to be relatively small [2, 11].

Benchmark selection and description Traces from in conventional predictors. SPEC2000 and SPEC95 integer benchmarks were used to check the behaviours of the predictors tested in the paper.

Traces were gathered on a Solaris Sun workstation. Programs were compiled with the Sun compiler with optimization -xO5 in order to avoid the artefacts associated with non-optimized codes. Reference inputs were used. 100 millions instructions were collected after skipping the initialization phases of the programs.

Some of the traced applications exhibited very low misprediction ratios. In this paper, we only report results for 15 applications (out of a total of 17 recorded traces) from SPEC2000. The remaining two traces *wupwise, perlbmk* exhibited insignificant misprediction ratios, at least on the traced segment of the applications and on the spectrum of branch predictors considered in this study. Traces from 4 SPEC95 integer benchmarks are also considered since they exhibit interesting behaviours which are not encountered on the SPEC2000 traces. In particular *gcc95* is much more demanding on the branch predictor than its counterpart in SPEC2000.

Note that the misprediction ratios presented in this paper should not be considered as representative of the behaviour of the total applications, but only of an execution slice of the application.

The characteristics of each individual trace are given in Table 1

For sake of simplicity, in the remainder of the paper, we will denote the original perceptron predictor from Jiménez and Lin by JL perceptron predictor. Simulation results are reported as percentages of mispredictions.

Instruction fetch blocks In Section 6, we consider ahead pipelining the branch predictor. For the sake of simplicity, all the simulation results illustrating this paper consider that a fetch block ends either on a control flow instruction (even on not-taken branches) or at the end of an 16-instruction aligned block. The address used for indexing the branch predictor is the address of the first word of the instruction fetch block.

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
dyn. branches (x 10000)	981	1215	1687	1777	237	1008	1402	649	1102	1126
static branches	182	146	3582	75	358	1077	441	1398	33	212
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
dyn. branches (x 10000)	380	1076	479	1724	277	723	1382	834	926	18913
static branches	38	74	185	432	47	80	12317	695	407	

Table 1: Benchmark characteristics

Predictor initialization Branch predictor behaviour are sensitive to the initial state of the predictor. In order to model a realistic initial state in the predictor, the predictor is initialized by running first the simulations on the most demanding benchmarks then running the 19 simulations in the order displayed in the simulation result tables.

History length Perceptron predictors are effective at capturing correlation on long history. For sake of simplicity, in this paper we will consider mainly 64-bit input length for the perceptron-like predictors.

3.1 Initial simulation results comparison

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
JL global perc.	10.05	6.79	4.19	8.20	0.39	6.98	3.48	0.85	0.44	7.60
JL local/global perc.	9.41	6.34	3.45	7.93	0.28	5.25	2.37	0.39	0.42	6.12
2bcgskew (0,13,13,13)	10.42	8.19	4.21	8.30	2.31	6.28	3.08	1.07	0.45	9.63
2bcgskew (5,5,20,40)	10.56	7.81	2.58	8.76	0.83	4.88	1.83	0.54	0.48	8.03
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
JL global perc.	1.91	0.25	1.11	2.88	0.22	9.90	11.78	2.88	1.12	**4.97**
JL local/global perc.	0.92	0.03	1.06	2.16	0.22	9.18	7.86	2.32	0.56	**4.08**
2bcgskew (0,13,13,13)	1.92	1.78	1.25	4.41	1.85	10.91	6.57	3.26	3.26	**5.21**
2bcgskew (5,5,20,40)	2.05	0.23	1.26	1.78	0.51	11.63	6.43	2.80	1.58	**4.38**

Table 2: Reference misprediction rate: 64Kbits range predictors

For sake of simplicity, in the remainder of the paper, we will denote the original perceptron predictor from Jiménez and Lin by JL perceptron predictor. Simulation results are reported as percentages of misprediction.

We report in Tables 2 and 3 simulation results (measured in misprediction percentages) for the JL global history perceptron predictor using 64 weights for storage budgets, the JL global/local history perceptron predictor assuming 64 global history bits and 16 local history bits, the original

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
JL global perc.	9.98	6.53	3.62	8.14	0.38	5.87	3.11	0.64	0.44	6.78
JL local/global perc.	9.35	6.20	3.20	7.87	0.28	4.79	2.22	0.28	0.42	5.80
2bcgskew (0,16,16,16)	10.33	7.95	3.79	7.93	1.27	5.67	2.60	0.94	0.46	9.15
2bcgskew (8,8,32,64)	10.37	7.42	1.56	7.05	0.52	3.94	1.40	0.45	0.47	7.26
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
JL global perc.	1.90	0.25	1.11	2.60	0.22	9.90	7.18	2.81	1.10	**4.38**
JL local/global perc.	0.92	0.03	1.04	2.14	0.22	9.19	4.97	2.26	0.54	**3.76**
2bcgskew (0,16,16,16)	1.89	1.78	1.25	3.67	1.85	10.55	4.98	3.06	2.51	**4.77**
2bcgskew (8,8,32,64)	2.25	0.26	1.21	0.97	0.31	11.31	4.87	1.94	1.23	**3.71**

Table 3: Reference misprediction rates: 512Kbits range predictors

2^n *2bcgskew* predictor (i.e., 2^{n-3} entries per table, no hysteresis sharing, history length n for G0, G1 and Meta), the optimized *2bcgskew* described in the previous section. We report this accuracy for predictors with storage size equal to 64Kbits and 512Kbits for *2bcgskew* and JL global history perceptron, and 96 Kbits and 512 Kbits for the JL global/local history perceptron predictors. Average accuracy on a larger spectrum of predictor size is reported later in the paper (see Table 6 in Section 5.4 [1].

Our simulations show that the optimized *2bcgskew* predictor significantly outperforms the initial *2bcgskew* predictor. It also generally outperforms the JL global history perceptron predictor, but also that it fails seriously to capture correlation on a few benchmarks *gzip, vpr, twolf* and *comp*.

4 Enhancing the perceptron prediction scheme

In this section, we propose three orthogonal optimizations of perceptron prediction scheme, pseudo-tagging, redundant history and skewing. These techniques can be applied on the global history perceptron as well as the global/local history perceptron and improve their accuracy.

4.1 Pseudo-tagging

On finite storage predictors, aliasing may induce a very large number of mispredictions. On perceptron predictors, this is also true. For instance, let us consider that two branches with strong but opposite biases share a single perceptron on in the predictor and are used with approximately the same frequency. The bias counter in the perceptron is not able to predict the branches. The accuracy of the prediction then only relies on the ability of the perceptron to linearly separate the history paths from the two branches.

[1] FOR REVIEWERS: We do not report simulation results on the Path-Based neural predictor, since our simulation results for this predictor were not completely coherent with the results presented in [6]. On our benchmark set, we found that the path-based neural predictor is approximately as accurate as the initial *2bcgskew* proposal, i.e slightly less accurate than the JL global history predictor, but signifi cantly less accurate than the JL local/global history perceptron predictor.

In order to limit this phenomenon, **pseudo-tagging** can be used on the perceptron predictor. Pseudo-tagging consists in using a few bits of the address of the branch in the vector of weights. When several strongly biased branches with different bias share a perceptron, the perceptron will be able to predict them correctly if it is able to linearly separate (i.e., recognize) their respective addresses.

In practice, using 8 address bits was found to be effective for 64-input predictors with sizes ranging from 64Kbits to 512 Kbits.

4.2 Using a redundant history

As pointed out in Section 2, the perceptron is not able to learn functions that are not linearly separable on its set of inputs.

However a simple mean to increase the number of functions that can be recognized by a perceptron on a set of n binary inputs is to use m weights ($m > n$) and a redundant representation of the input vector (i.e, the n inputs plus $m - n$ functions of the n inputs). As an example, a perceptron using only the 2 weights and an input vector ($x1$,$x2$) is not able to recognize the function $x1 \oplus x2$, but a perceptron using the 3 weights, and ($x1$, $x2$, $x1 \oplus x2$) as input vector can obviously recognize this function.

Experiments on perceptron predictors showed that introducing up to four bits to represent a branch significantly improves the potential accuracy of the predictor when considering same length of branch history, (i.e 256 bits to represent 64 branches). Further increasing the number of bits for representing a branch does not lead to significant extra accuracy gain.

The four versions of the history H0, H1, H2, H3 that are used in the results illustrating the paper. were obtained by updating the history on each branch as follows. Let *Out*, *Out1*, *Out2*, *Out3* be the respective outcomes of the current branch, the previous branch and the second previous branch, *Add* the address of the branch, $>>$ be the right shift, $\oplus$ be the exclusive OR:

- $H0 := (H0 << 1) + Out$
- $H1 := (H1 << 1) + (Out \oplus Out1)$
- $H2 := ((H2 << 1) + (Out \oplus Out2)$
- $H3 := ((H3 << 1) + (Out \oplus Out3)$

4.3 Skewing the perceptron predictor

With the conventional model of a perceptron predictor, all the counters for a single branch prediction are associated with a single index, for instance the lowest significant bits of the address. However, for most branches, most of the correlation with previous branches is captured by the counters associated with recent history. On the associated perceptron table entries, the counters associated with the older history are underutilized.

On the other hand, in some cases the behaviour of a branch is correlated with the values of old history bits. As an example, let us consider a branch B which can be reached by two different paths corresponding respectively to history H and history H'. Let us consider that the behaviour of branch B is strongly biased with the 26th bit in both cases, but in opposite directions. A "self-aliasing" phenomenon occurs on the perceptron counter associated with this history bit.

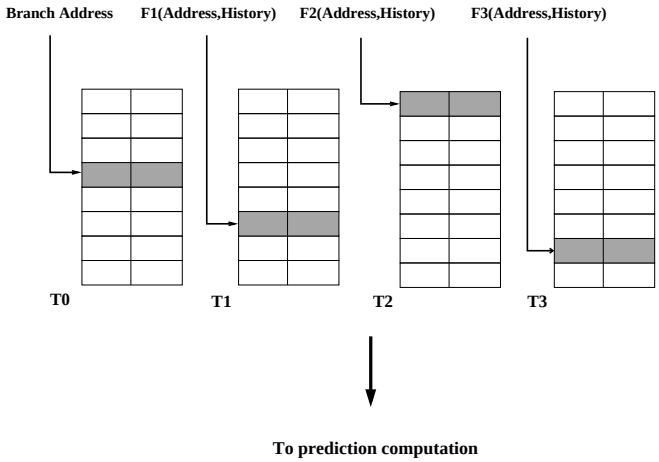


Figure 2: Skewing the perceptron predictor

To reduce the impact of this self-aliasing phenomena, the perceptron tables can be split in distinct physical tables, the counters being spread over the distinct tables. The distinct tables are indexed through hash functions incorporating different history lengths. This is illustrated using four tables on Figure 2.

Table T0 corresponds to the most recent bits in histories, Table T3 corresponds to the least recent bits of the histories. Tables T0 to T3 are indexed using hash functions combining the address of the branch with the least significant bits of the history H0. In the presented simulation results, 0, 4, 8 and 16 bits of the history are used in the hash function for respectively indexing tables T0, T1, T2 and T3 for the 64 Kbits predictor and 0, 7, 14 and 28 for the 512 Kbits predictor.

We will refer to the predictor using pseudo-tagging redundant history and skewed indexing of tables as the redundant history skewed predictor (RHSP).

4.4 Evaluation

In Tables 4 and 5, we illustrate the benefits of adding the three proposed enhancements on a global history perceptron predictor. As expected each of the three propsitions brings some extra accuracy. Pseudo-tagging being more efficient for the 64Kbits predictor than on the 512 Kbits predictor (the smaller the predictor, the higher the aliasing impact). Using redundant representation of the history further brings some extra accuracy despites each perceptron uses 4 times more counters. Finally skewing also enhances accuracy.

Benefits in accuracy are very significant on many benchmarks (*gcc*, *mcf*, *parser*, *gap*, *twolf*, *fma3d*, *comp gcc95* and *li*). On other benchmarks, the original JL perceptron was already capturing most of the correlation (*gzip*, *crafty*, *bzip2*, *equake*, *ammp*, *apsi*, *m88*).

Skewing, pseudo-tagging and redundant history is also beneficial on RHSP predictors combining local and global history. However the accuracy difference between using only global history and combining local and global history is much more limited on the RHSP predictor than on the original JL perceptron predictor.

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
JL global perc.	10.05	6.79	4.19	8.20	0.39	6.98	3.48	0.85	0.44	7.60
+ 8 Pseudo-tag bits	9.79	6.51	3.73	7.98	0.45	5.76	2.90	0.52	0.42	6.08
+Redundant hist.	9.65	6.40	3.25	7.67	0.41	5.15	2.03	0.43	0.42	6.26
RHSP global	9.67	6.30	2.46	7.47	0.41	4.53	1.50	0.42	0.42	5.57
RHSP global/local	9.38	6.18	2.06	7.39	0.27	4.11	1.13	0.31	0.42	5.33
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
JL global perc.	1.91	0.25	1.11	2.88	0.22	9.90	11.78	2.88	1.12	**4.97**
+ 8 Pseudo-tag bits	1.87	0.25	1.11	2.68	0.22	9.85	8.71	2.87	1.10	**4.42**
+Redundant hist	1.79	0.25	1.08	1.74	0.22	9.39	8.58	2.59	1.08	**4.14**
RHSP global	1.78	0.25	1.02	1.38	0.22	9.05	8.53	2.15	1.06	**3.87**
RHSP global/local	0.86	0.05	0.99	1.05	0.22	8.62	7.11	1.73	0.43	**3.51**

Table 4: Enhancing the prediction scheme: 64Kbits predictors

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
JL global perc.	9.98	6.53	3.62	8.14	0.38	5.87	3.11	0.64	0.44	6.78
Pseudo-tag	9.78	6.44	3.51	7.93	0.43	5.40	2.80	0.47	0.42	5.90
+Redundant hist	9.60	6.11	2.86	7.39	0.41	4.33	1.82	0.37	0.42	5.48
RHSP global	9.68	5.96	1.65	6.08	0.41	3.51	1.31	0.38	0.42	4.59
RHSP global/local	9.40	5.82	1.34	6.07	0.27	3.25	0.91	0.25	0.42	4.48
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
JL global perc.	1.90	0.25	1.11	2.60	0.22	9.90	7.18	2.81	1.10	**4.38**
Pseudo-tag	1.87	0.25	1.11	2.62	0.22	9.84	5.28	2.84	1.08	**4.09**
+Redundant hist	2.10	0.25	1.08	1.73	0.22	9.39	5.11	2.49	1.07	**3.69**
RHSP global	1.80	0.25	1.02	1.16	0.22	8.88	4.98	1.84	1.05	**3.21**
RHSP global/local	0.87	0.05	0.99	0.91	0.22	8.37	4.42	1.43	0.36	**2.95**

Table 5: Enhancing the prediction scheme: 512Kbits predictors

5 Reducing the complexity of the prediction computation

The large number of additions and counter updates required by a prediction is a major obstacle to the effective use of a perceptron predictor. This difficulty is even higher when using the RHSP predictor: if a 64-bit history is used then a multiply-accumulation tree on 256 values has to be implemented. Moreover on a prediction update, 256 values must be updated. The logic for computing the predictions and for updating the predictor would require a large silicon area and would be power hungry, moreover the latency for computing the prediction includes a 256-entry multiply-accumulation tree.

In this section, we introduce the MAC representation (for Multiply-Accumulation Contribution) for redundant input perceptron predictor. When using a 4-way redundant 64-bit history, the complexity of prediction computation is reduced from a 256 8-bit entry multiply-add tree to a 16 6-bit entry adder tree. Moreover only 16 values have to be updated on a predictor update.

5.1 Principle of MAC representation on a simple example

In order to illustrate the principle of MAC representation, let us first consider the example of a four weights perceptron predictor using only 2 bits of global history, i.e., using some input redundancy.

Let input bits R0= 0, R1= h0, R2= h1, R3 = h0 $\oplus$ h1 be associated with h=(h0,h1)
let W0, W1, W2, W3 be the weights respectively associated with R0, R1, R2, R3, then the possible final multiply-accumulate results are:

- h= (0,0): C0= -W0 -W1 - W2 - W3
- h= (1,0): C1= -W0 + W1 - W2 + W3
- h= (0,1): C2= -W0 - W1 + W2 + W3
- h=(1,1): C3= -W0 + W1 + W2 - W3

Moreover the update of the predictor results in the update of a single final multiply-accumulate result. For instance if h= (0,1) and Out =1 then C2 is increased by four, but C0, C1 and C3 remain unchanged.

Therefore, for such a simple predictor, storing the multiply-add results Ci instead of the weights Wi saves the multiply-adder tree in the prediction computation as well as 3 out of the 4 counter updates on predictor update.

5.2 Using MAC representation for 4-way redundant input predictor

The principle illustrated in the previous example can be used for 4 times redundant input perceptron predictor as follows.

Let us consider that the redundant input vector must represent 4N binary information (for instance branch history) through 16N bits. For each block $I = (I0, I1, I2, I3)$ of four consecutive binary information, one can associate the 16 input bits R_i defined by the 16 possible exclusive-OR combinations of the bits in I, i.e., $\{0, I0, I1, I2, I3, I0 \oplus I1, I0 \oplus I2, I0 \oplus I3, I1 \oplus I2, I2 \oplus I3, I2 \oplus I3, I0 \oplus I1 \oplus I2, I1 \oplus I2 \oplus I3, I2 \oplus I3 \oplus I0, I3 \oplus I0 \oplus I1, I0 \oplus I1 \oplus I2 \oplus I3\}$.

The total contribution to a prediction of the 16 counters associated with this block I is one out of only 16 possible contributions, since I and therefore the associated vector R have only 16 possible distinct values.

Therefore one can choose to store the 16 possible multiply-accumulate contributions to the prediction instead of storing the 16 effective counters. Instead of computing the contribution by a 16-entry adder tree, the contribution is selected out of the 16 possible contributions. Moreover, as on the example previously illustrated, only the selected contribution is modified on a predictor update: 16 is added or subtracted to this contribution while the 15 other contributions remain unchanged. Therefore the stored values can be divided by 16 without loss of accuracy.

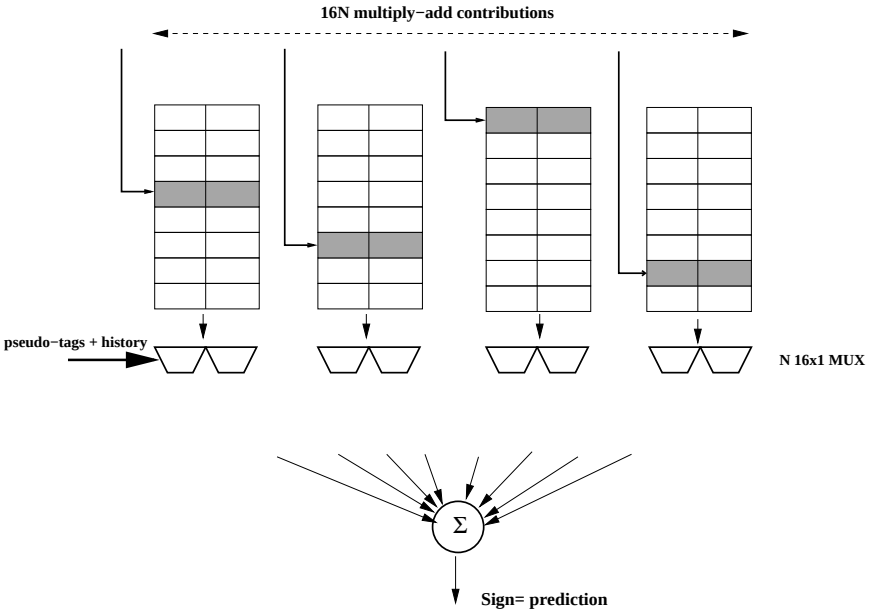


Figure 3: The MAC-RHSP predictor

Note that this reduces the prediction computation time (a multiply by 1 or -1 followed 16-to-1 sum are replaced by a 16-to-1 multiplexor) as well as the complexity of hardware logic for prediction computation and for predictor update.

We will refer to this representation of the multiply-accumulate contributions as the MAC representation instead of the full weights representation.

Formal definition of MAC representation $4*N$ being the width of the input vector (i.e. pseudo-tags + history), the MAC representation of a (4 times redundant input) perceptron consists in $16*N$ values $C(j)$, $0 \leq j < 16*N$.

$H = (h_N, .., h_1)$ being the hexadecimal representation of the branch history, the prediction p is computed as the sign of $y = 16 * \sum_{i=0}^{N-1} C(h_i + 16*i)$.

The (redundant history) perceptron is trained as follows, *Out* being the output of conditional branch:

if ((p !=*Out*) or ($|y| < \theta$)){
for each i in parallel
{ if *Out* then $C(h_i+16*i) = C(h_i+16*i)+1$ else $C(h_i+16*i) = C(h_i+16*i)-1$ }}

Since the basic scheme is a perceptron with 16*N weights the threshold is chosen as $\theta = 1.93 * 16N + 14$.

The MAC RHSP predictor A RHSP predictor can also use MAC representation as illustrated in Figure 3. We will be referred to such a RHSP predictor using MAC representation as a MAC-RHSP predictor.

5.3 MAC representation is accuracy effective

Simulations showed that the MAC-RHSP predictor exhibit behaviour in the same range as the RHSP predictor for global history predictors as well as for combined local/global history predictors. Moreover, due to the use of saturated arithmetic, using MAC contribution is slightly more precise than using conventional weights. Using 6-bit counters instead of 8-bit counters for MAC representation was found to result in equivalent misprediction rates, and saves one fourth of the storage budget in the predictor.

5.4 Accuracy summary

Table 6 summarizes the average misprediction percentage for the predictors considered in this paper for storage sizes ranging from 48Kbits to 1.5 Mbit.

	64K	128K	256K	512K	1024K
Opt. 2bcgsk	4.38	4.12	3.94	3.71	3.51
JL global	4.97	4.75	4.53	4.38	4.26
RHSP global	3.87	3.55	3.38	3.21	3.08
	48K	96K	192K	384K	768K
MAC-RHSP global	3.92	3.59	3.42	3.25	3.12
	96K	192K	384K	768K	1.5Mbit
JL global/local	4.08	3.95	3.84	3.76	3.70
RHSP global/local	3.51	3.24	3.10	2.95	2.84
	76K	152K	304K	608K	1216K
MAC-RHSP glob/loc	3.53	3.25	3.12	2.97	2.87

Table 6: Summary of predictor accuracies

5.5 Prediction computation time and hardware logic complexity on the MAC-RHSP predictor

The prediction computation time on the RHSP predictor involves three components: the predictor tables read delay, the 16-to-1 multiplexor delay and the adder tree. The adder tree delay depends on the number of entries: for the 64-input predictors we are considering in this paper, a 16 6-bit entry adder tree is used. This can be implemented using a 6-stage carry save adder tree and a final 2-entry 9-bit adder.

Hardware logic for the perceptron predictor includes the multiply-add tree and the logic for weights update, i.e., a counter update logic per block of 4 input bits.

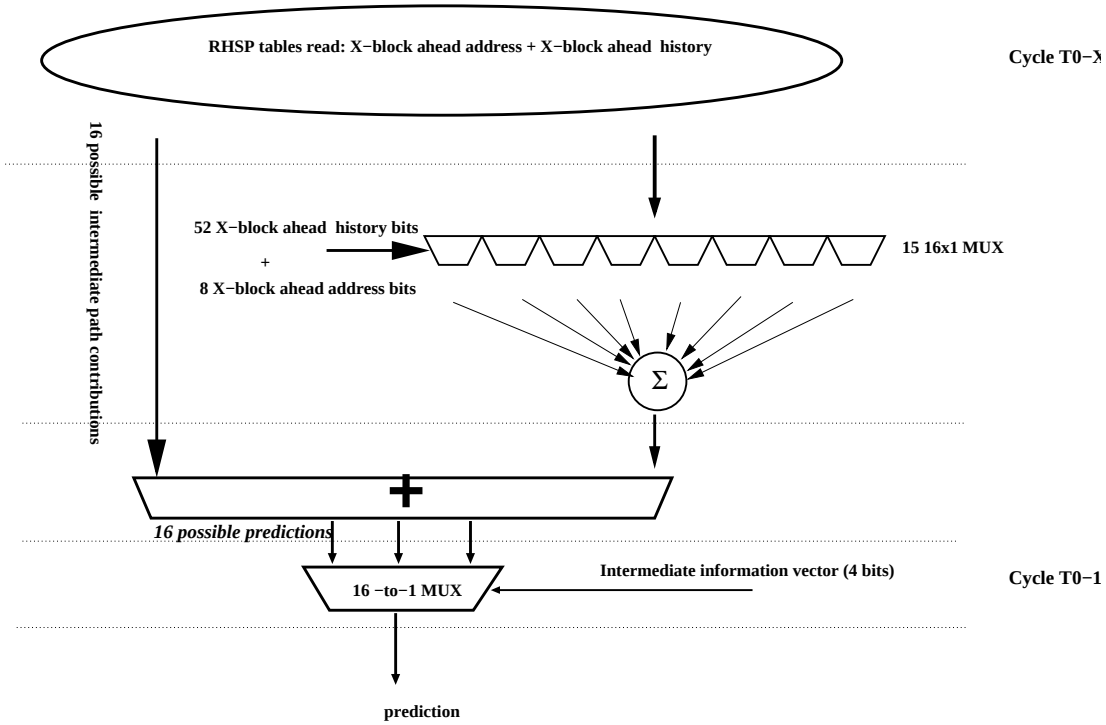


Figure 4: Ahead pipelining the RHSP

6 Ahead pipelining the MAC-RHSP predictor

As just mentioned above, prediction computation on the 64-bit input MAC-RHSP predictor will not be achieved in single cycle on a high-performance processor. However, the X-block ahead MAC-RHSP predictor presented in this section will allow to provide predictions in-time. By in-time prediction, we mean that, if the address of a fetch block is known on cycle T the prediction of a conditional branch in this fetch block is known before cycle T+1.

6.1 Principles

In Figure 4, we illustrate a X-block ahead MAC-RHSP branch predictor. We describe below the sequencing of the computation of the prediction of a branch that will be fetched at cycle T0:

1. At cycle T0-X, the MAC-RHSP predictor tables are read using the X-block ahead address and X-block ahead history *h* for indices computation.
2. 15 values are selected using *h* and pseudo-tag bits. Note that 16 values associated with the 16 possible 4-bit information vectors on intermediate path *I* have also been read.
3. The result R of the accumulation on the 15 read values associated with the X-block ahead history and pseudo-tag bits is computed.

4. R is added to each of the 16 possible contributions associated with the 16 possible information vectors *I* 32 possible predictions are then available.
5. On cycle T0-1, the effective 4-bit one-block ahead history *H* exclusive-OR with bits of the 1 block-ahead address is used to select the correct prediction through a 16-to-1 multiplexor.

   Using this exclusive-OR as an intermediate vector information is more effective than using only the 1-block ahead history.

One can notice that, in step 1, the prediction is initiated using X-block ahead information (block address and global history vector).

As already mentioned in Section 2, use of some information on intermediate blocks is needed to obtain correct accuracy. Therefore we compute in parallel 16 predictions corresponding to the 16 possible 1-block ahead global history.

Computing a prediction on the MAC-RHSP predictor normally consists in a tree of adders. However, the 16 possible predictions share the same intermediate result of a 15-entry adder tree. The overhead logic for the parallel computation of the 16 possible intermediate predictions is therefore limited to 15 extra 10-bit adders.

6.2 Resuming on a misprediction or an interruption

On a misprediction (or an interruption), the X-block ahead MAC-RHSP predictor cannot deliver the prediction in-time for the next X-1 cycles.

However, if the 16 possible predictions associated with the 16 possible 4-bit information vectors on intermediate path have been checkpointed, the checkpoint logic can deliver the predictions for the X blocks following the misprediction. This allows to resume instruction fetching just after branch execution without extra mispenalty and/or without waiting for committing the branch instruction.

6.3 Performance evaluation

Storage size in bits	48K	96K	192K	384K	768K
1-block ahead	3.92	3.59	3.42	3.25	3.12
6-block ahead	4.30	3.90	3.67	3.48	3.34
10-block ahead	4.65	4.18	3.88	3.66	3.48

Table 7: Average misprediction rates for ahead pipelined MAC-RHSP predictors

Simulation results assuming 6-block ahead and 10-block ahead pipelined MAC-RHSP are displayed on Table 7 and Table 8. Table 8 presents the results for the whole set of benchmarks for a 384Kbits MAC-RHSP predictor. Table 7 presents the average misprediction percentage for a spectrum of predictors ranging from 48Kbits to 768 Kbits.

As expected, using the X-block ahead RHSP predictor instead of a 1-block ahead predictor results in some accuracy loss. Two factors explain this loss. First the direction of the branch is less correlated with the X-block ahead address than with the 1-block ahead branch address. Therefore the

	gzip	vpr	gcc	mcf	crafty	parser	gap	vortex	bzip2	twolf
1-block ahead	9.69	6.00	1.77	6.09	0.42	3.54	1.32	0.39	0.42	4.64
6-block ahead	10.00	6.13	1.61	6.07	0.43	3.81	1.43	0.43	0.43	5.13
10-block ahead	10.09	6.26	1.71	6.20	0.44	4.08	1.48	0.45	0.43	5.75
	equake	facerec	ammp	fma3d	apsi	comp	gcc95	li	m88	**Average**
1-block ahead	2.11	0.25	1.03	1.18	0.22	8.94	4.96	1.92	1.04	**3.25**
6-block ahead	1.93	0.25	1.04	1.42	0.43	9.08	6.89	1.73	1.32	**3.48**
10-block ahead	1.84	0.25	1.07	1.32	0.43	9.21	8.13	1.77	1.32	**3.66**

Table 8: Ahead pipelined MAC-RHSP: 384Kbits predictor

counters associated with the low order inputs of the information vector (address bits, intermediate path vector and low order history bits) are less likely to force the prediction on a X-block ahead RHSP predictor than on a 1-block ahead predictor. More aliasing effects are likely to be encountered on the X-block ahead predictor than on the 1-block ahead predictor. Second, only 32 different possible intermediate path vectors are considered, there may some path aliasing., particularly if X is high.

However, on our set of benchmarks the accuracy loss is very limited. In average, the following rule of thumb is respected: the 4S storage size 10-block ahead pipelined MAC-RHSP, the 2S storage size 6-block ahead pipelined MAC-RHSP and the 1S storage size 1-block ahead MAC-RHSP have the same average prediction accuracy.

7 Conclusion

Jiménez and Lin [3, 4] proposed to use perceptrons for branch predictions. Their original JL perceptron predictor was achieving accuracy in the same range or even higher than the one of previous academic proposals. The JL perceptron predictor is handicapped by the complex hardware logic (a multiply-accumulate tree) required for prediction computation and predictor update. Its response time would be significantly longer than the response time of more conventional branch predictors and longer than a CPU cycle.

In this paper, we have improved the initial work by Jiménez and Lin in three directions.

We have first shown that the accuracy that can be achieved by a perceptron-based predictor, the RHSP predictor, is significantly better than the one achieved by the original JL perceptron predictor To the best of our knowledge, it also outperforms current state-of-the-art conventional global history branch predictors. This accuracy increase is allowed by the combination of three techniques: use of a redundant history, skewing technique, pseudo-tagging. These techniques can be applied on perceptron predictors using only global history as well as combining local and global histories.

At second, we have shown that the use of a redundant history creates an opportunity for dramatically reducing the hardware complexity of the logic needed for prediction computation and predictor update. The MAC representation we have proposed allows to reduce the number of adders and the numbers of counter updates by a factor 16. It also allows to reduce the width of the stored values

from 8 bits to 6 bits without sacrificing prediction accuracy. This translates in a reduction of the depth of the adder tree in prediction computation, and therefore in a reduction of the prediction computation time.

Third, a 64-input MAC-RHSP predictor still requires a 16-entry tree of adders. Therefore its response time will still be longer than a CPU cycle. To address this issue, we have shown that the MAC-RHSP predictor can be efficiently ahead pipelined. On most benchmarks and for X smaller or equal 10, a X-block ahead pipelined RHSP predictor suffers only very limited accuracy degradation. Moreover, the volume of information to be checkpointed to allow immediate restart after the execution of a branch on a misprediction remains very limited (16 bits).

The contributions of this paper show that perceptron-based branch predictors can now be considered for real implementations in high-end microprocessors. In this paper, we have essentially focussed on a 16-adder tree MAC-RHSP predictor. Depending on implementation constraints (storage budgets, predictor latency, power consumption), other implementations featuring wider or narrower adder tree can be considered. Tradeoffs in the number, in the size and in the width of the tables in RHSP predictor may even lead to more cost effective predictors.

References

[1] L. Faucett. *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*. Prentice-Hall, 1994.

[2] D. Jiménez. Reconsidering complex branch predictors. In *Proceedings of the 9th International Symposium on High Perform ance Computer Architecture*, 2003.

[3] D. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the Seventh International Symposium on High Perform ance Computer Architecture*, 2001.

[4] D. Jiménez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, November 2002.

[5] Daniel Jiménez, Stephen W. Keckler, and Calvin Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, Monterey, California, December 2000.

[6] Daniel A. Jiménez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, dec 2003.

[7] Scott McFarling. Combining branch predictors. Technical report, DEC, 1993.

[8] A. Seznec. An optimized *2bcgskew* branch predictor. Technical report, http://www.irisa.fr/caps/people/seznec/optim2bcgskew.pdf, Irisa, Sep 2003.

[9] A. Seznec and A. Fraboulet. Effective ahead pipelining of the instruction addres generator. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[10] A. Seznec and P. Michaud. De-aliased hybrid branch predictors. Technical Report RR-3618, Inria, Feb 1999.

[11] André Seznec, Stephen Felix, Venkata Krishnan, and Yanos Sazeidès. Design tradeoffs for the ev8 branch predictor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, 2002.

[12] C. Young, N. Gloy, and M.D. Smith. A comparative analysis of schemes for correlated branch prediction. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.