Analysis of the O-GEometric History Length branch predictor *

André Seznec
IRISA/INRIA/HIPEAC
Campus de Beaulieu, 35042 Rennes Cedex, France
seznec@irisa.fr

Abstract

In this paper, we introduce and analyze the Optimized GEometric History Length (O-GEHL) branch Predictor that efficiently exploits very long global histories in the 100-200 bits range.

The GEHL predictor features several predictor tables $T(i)$ (e.g. 8) indexed through independent functions of the global branch history and branch address. The set of used global history lengths forms a geometric series, i.e., $L(j) = \alpha^{j-1}L(1)$. This allows the GEHL predictor to efficiently capture correlation on recent branch outcomes as well as on very old branches. As on perceptron predictors, the prediction is computed through the addition of the predictions read on the predictor tables.

The O-GEHL predictor further improves the ability of the GEHL predictor to exploit very long histories through the addition of dynamic history fitting and dynamic threshold fitting.

The O-GEHL predictor can be ahead pipelined to provide in time predictions on every cycle.

1. Introduction

Modern processors feature moderate issue width, but deep pipelines. Therefore any improvement in branch prediction accuracy directly translates in performance gain.

The O-GEHL predictor [22] was recently proposed at the first Championship Branch Prediction (Dec. 2004). While receiving a best practice award, the 64 Kbits O-GEHL predictor achieves higher or equivalent accuracy as the exotic designs that were presented at the Championship Branch Prediction workshop [4, 15, 10, 1]. It also outperforms Michaud's tagged PPM-like predictor [17] which was the only other implementable predictor presented at the ChampionShip.

* This work was partially supported by an Intel research grant and an Intel research equipment donation

In this paper, we analyze the O-GEHL branch predictor in great details and explore the various degrees of freedom in its design space.

The O-GEHL predictor (Figure 1) implements multiple predictor tables (typically between 4 to 12) . Each table provides a prediction as a signed counter. As on a perceptron predictor [8], the overall prediction is computed through an adder tree. The index functions of the tables are hash functions combining branch (and path) history and instruction address. GEHL stands for GEometric History Length since we are using an approximation of a **geometric series** for the history lengths L(i), i.e., $L(i) = \alpha^{i-1}L(1)$.

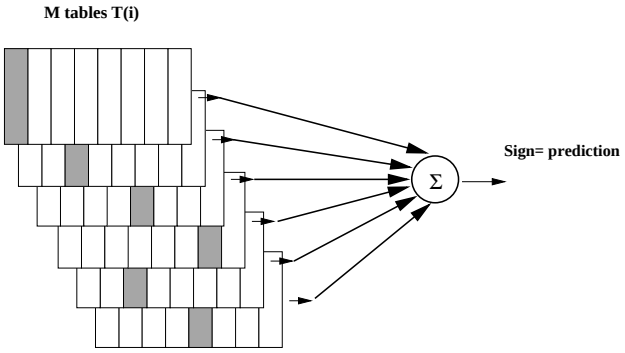

Figure 1. The GEHL predictor

The O-GEHL implements a simple form of dynamic history length fitting [12] to adapt the behavior of the predictor to each application. In practice, for demanding applications, most of the storage space in the O-GEHL predictor is devoted to tables indexed with short history lengths. But, the combination of geometric history length and of dynamic history lengths fitting allows the O-GEHL predictor to exploit very long global histories (typically in the 200-bits range) on less demanding applications. The O-GEHL predictor also implements a threshold based partial update policy augmented with a simple dynamic threshold fitting.

We explore in details the design space for the O-GEHL predictor: number of predictor tables, counter width, min-

imum and maximum history lengths, storage size impact, update threshold. Experiments show that the O-GEHL predictor provides very accurate predictions. We also show that the O-GEHL predictor is very robust in the sense that overdimensioning or underdimensioning parameters such as the maximum global history length, the update threshold, or the counter width does not significantly impair the predictor accuracy.

As for any state-of-the-art branch predictor, the response time of the O-GEHL predictor is longer than a CPU cycle since it involves the read of several large tables followed a small adder tree. However, the O-GEHL predictor can be adapted to provide predictions in time for use through ahead pipelining [25].

Paper outline

The remainder of the paper is organized as follows. Section 2 presents our experimental framework for simulation and evaluation of the branch predictors. Section 3 introduces the O-GEHL predictor principles. Section 4 explores the design space of the O-GEHL predictor. Section 5 shows that the O-GEHL predictor can be efficiently ahead pipelined [25] in order to provide the prediction in time for use. Finally, Section 6 reviews related work and summarizes this study.

2. Evaluation framework

2.1. Simulation traces and evaluation metric

To allow reproducibility, the simulations illustrating this paper were run using the publicly available traces provided for the 1st Championship Branch Predictor (http://www.jilp.org/cbp/). 20 traces selected from 4 different classes of workloads are used. The 4 workload classes are: server, multi-media, specint, specfp. Each of the branch traces is derived from an instruction trace consisting of 30 million instructions. These traces include system activity.

30 million instruction traces are often considered as short for branch prediction studies. However 30 million instructions represent approximately the workload that is executed by a PC under Linux or Windows in 10 ms, i.e., the OS time slice. Moreover, system activity was shown to have an important impact on predictor accuracy [5]. Finally, some traces, particularly server traces, exhibit very large number of static branches that are not represented in more conventional workloads such as specint workloads. The characteristics of the traces are summarized in Table 1.

The evaluation metric used in this paper is misprediction/KI. Due to space limitations, in most places we will use the average misprediction rate computed as the ratio of the total number of mispredictions on the 20 benchmarks divided by the total number of instructions in the 20 traces.

Since this study was performed on traces, immediate update of the predictor is assumed. On a real hardware processor, the effective update is performed later in the pipeline, at misprediction resolution or at commit time. However, for branch predictors using very long global branch history, the differences of accuracy between a delayed updated predictor and an immediately updated predictor are known to be small [7, 24].

2.2. Predictor initialization

Exact reproducibility supposes exact equal initial state. In the simulation results presented in [22], all counters were therefore initialized to zero. However, branch predictor behaviors might be sensitive to the initial state of the predictor. Resetting counters before simulating each trace leads to underestimate cold start effects. On the same hand, for the O-GEHL predictor as well as for all the predictors using wide counters (perceptron predictors for instance), uniform random initialization of the predictor tables is not a realistic initialization, since after an application execution the contents of the predictor tables are not uniformly distributed.

In order to approach a realistic initialization point, the simulations presented in this paper assume that the simulations of the 20 traces are chained without resetting the predictor counters. Compared with assuming resetting all the counters before simulating each trace, the discrepancy in prediction accuracy is relatively marginal for O-GEHL predictors with moderate storage budgets (0.03 misp/KI for the reference 64Kbits O-GEHL predictor). This discrepancy increases with the storage budget and it also increases with the width of the counters used in the predictor.

2.3. Information used for indexing the branch predictor

For computing the indexes for global history predictors, most studies consider either hashing the conditional branch history with the branch address or hashing the path history with the branch address [19]. Both these solutions lead to consider some paths as equal even before computing the effective index in the predictor tables. This phenomenon can be called *path aliasing*. The impact of path aliasing on predictor accuracy is particularly important when a short global history is used.

The O-GEHL predictor uses several tables indexed using short history lengths. Therefore path aliasing would impair the accuracy of the predictions provided by the tables indexed with short histories.

In order to limit this phenomenon, we include non-conditional branches in the branch history *ghist* (inserting a taken bit) and we also record a path history, *phist* consisting of 1 address bit per branch. Since path aliasing impact decreases when history length increases, the maximum

	FP-1	FP-2	FP-3	FP-4	FP-5	INT-1	INT-2	INT-3	INT-4	INT-5
static branches	444	452	810	556	243	424	1585	989	681	441
dynamic branches (x10000)	221	179	155	90	242	419	287	377	207	376
	MM-1	MM-2	MM-3	MM-4	MM-5	SERV-1	SERV-2	SERV-3	SERV-4	SERV-5
static branches	460	2523	1091	2256	4536	10910	10560	16604	16890	13017
dynamic branches (x10000)	223	381	302	488	256	366	354	381	427	429

Table 1. Characteristics of the CBP traces

path history length considered in this paper is 16, i.e., the path history length is equal to Min (history length, 16).

3. The O-GEHL predictor

In this section, we first introduce the general principles of the GEHL branch predictor. Then we introduce two extra features, dynamic history length fitting and dynamic threshold fitting that enables the GEHL predictor to adapt its behavior to each particular application. We will refer to the GEHL predictor augmented with dynamic history length fitting and dynamic threshold fitting as the O-GEHL predictor.

3.1. The GEHL predictor

The GEometric History Length (GEHL) branch predictor is illustrated on Figure 1. The GEHL predictor features M distinct predictor tables Ti, $0 \leq i < M$ indexed with hash functions of the branch address and the global branch/path history. The predictor tables store predictions as signed saturated counters. To compute a prediction, a single counter C(i) is read on each predictor table Ti. As, on perceptron predictors [8], the prediction is computed as the sign of the sum S of the M counters C(i), $S = \frac{M}{2} + \sum_{0 \leq i < M} C(i)$ [1]. The prediction is taken when S is positive or nul and not-taken when S is negative.

Distinct history lengths are used for computing the index of the distinct tables. Table T0 is indexed using the branch address. The history lengths used for computing the indexing functions for tables Ti, $1 \leq i < M$ are of the form $L(i) = \alpha^{i-1} * L(1)$, i.e., the lengths L(i) form a **geometric series**. More precisely, as history length are integers, we use $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$.

Using a geometric series of history lengths allows to use very long history lengths for indexing some predictor tables, while still dedicating the major of the storage space to predictor tables using short global history lengths. As an example on a 8-component GEHL predictor, using $\alpha = 2$ and L(1) =2 leads to the following series $\{0,2,4,8,16,32,64,128\}$. Remark that 5 tables are indexed using less than 16 history bits while correlation on a 128-bit history might still be captured.

1 For p-bit counters, predictions vary between -2^{p-1} and $2^{p-1} - 1$ and are centered on $-\frac{1}{2}$

3.1.1. Updating the GEHL predictor The GEHL predictor update policy is derived from the perceptron predictor update policy [8]. The GEHL predictor is only updated on mispredictions or when the absolute value of the computed sum S is smaller than a threshold θ. Saturated arithmetic is used. More formally, the GEHL predictor is updated as follows, *Out* being the branch outcome:

if ((p !=*Out*) or ($|S| \leq \theta$))
for each i in parallel
if *Out* then $C(i) = C(i) + 1$ else $C(i) = C(i) - 1$

3.1.2. Degrees of freedom in the design of GEHL predictors In Section 4, we will evaluate the degrees of freedom that exist for the design of a GEHL predictor.

First one can vary the number M of tables in the GEHL predictor. We will show that using 4 to 12 tables provides high level of accuracy. Second, we will show that using 4-bit counters is a cost-effective solution. Using a mix of 4-bit counter tables and 5-bit counter tables is the most storage effective solution. However using 3-bit counters or 5-bit counters are also cost-effective solutions. Third, one can vary the parameters of the geometric series of history lengths, i.e., L(1) and L(M-1) since $\alpha = (\frac{L(M-1)}{L(1)})^{\frac{1}{M-2}}$. For storage budget varying from 32 Kbits to 1 Mbits, the GEHL predictor is able to capture correlation on very long history in the hundred bits range. Furthermore, in Section 3.2, we propose a simple dynamic history length fitting mechanism [12] for the GEHL predictor. With this mechanism, the O-GEHL predictor is able to adapt the used history lengths to each application and even to phases in the application.

Fourth, one can vary the threshold θ for updating the predictor. Using $\theta = M$, the number of tables in the GEHL predictor is a good tradeoff when using 4-bit counters. However, the best threshold varies with the application, it may even vary during an application. In Section 3.3, we propose a simple dynamic threshold fitting policy which adapts the update threshold to the behavior of each application.

3.2. Dynamic history length fitting for the GEHL predictor

Experiments showed that, for a 8-component 64Kbits GEHL predictor, some benchmarks would achieve their best performance with a maximum history length L(7) around 50 (most of the server benchmarks) while others would benefit from using very long histories in the 150 bits range.

Juan et al [12] proposed to continuously adapt the branch history length during execution for global history branch predictors. The GEHL predictor offers an opportunity to implement such an adaptative history length fitting. We describe below such a history length fitting mechanism for a predictor featuring 8 tables.

Eleven history lengths L(j) forming a **geometric** series are used. For 3 out of the 8 predictor tables, Tables T2, T4 and T6, two possible histories lengths are used: Table T2 is indexed using either L(2) or L(8), Table T4 is indexed using either L(4) or L(9), Table T6 is indexed using either L(6) or L(10).

The algorithm we propose to dynamically select the history lengths for indexing the predictor is based on a rough estimation of the aliasing ratio on updates encountered on Table T7. Table T7 is the predictor component using the longest history apart L(8), L(9) and L(10). Intuitively, if Table T7 experiences a high degree of aliasing then short histories should be used on Tables 2, 4 and 6; on the other hand if Table T7 encounters a low degree of aliasing then long histories should be used.

In order to dynamically estimate the aliasing ratio on Table T7, we add a tag bit to some entries of Table T7. We use a single saturating 9-bit counter AC (for aliasing counter). On a predictor update, the tag bit records one bit of the address of the branch and we perform the following computation:

if ((p!=out) & ($|S| \leq \theta$)){
if ((PC & 1) == Tag[indexT[7]]) AC++; else AC= AC - 4;
if (AC == SaturatedPositive) Use Long Histories
if (AC == SaturatedNegative) Use Short Histories
Tag[indexT[7]] = (PC & 1);}

When the last update of the corresponding entry in Table T7 has been been performed using the same (branch, history) pair, AC is incremented. When the last update has been performed by another (branch, history) pair, AC is incremented on false hits and decremented by 4 on misses.

When the ratio of conflicting updates on Table T7 by distinct branches remains below 40 %, AC tends to be positive and therefore long histories are used. Using a 9-bit counter and flipping from short to long histories and vice-versa only on saturated values guarantee that such flippings are rare.

Simple is sufficient The proposed dynamic history length fitting only chooses between two modes *short histories* and *long histories*. More complex forms were tested, i.e., controlling history length flipping component by component. However, these more complex forms did not bring any significant accuracy benefit.

Predictor training after history length flipping After an history length flipping, predictor tables T2, T4 and T6 suffer from cold start effects. However the other predictor tables are still warm and this ensures good accuracy during the warming phase on tables T2, T4 and T6.

Other numbers of predictor tables For the simulations presented in this paper, we assume that history lengths are dynamically fitted on three predictor tables when the number of predictor tables is higher than or equal to 8, on two predictor tables for 5 and 6-component O-GEHL predictors and a single predictor table for the 4 component O-GEHL predictor.

Remark Associating a tag bit per entry in predictor table T7 is not needed. For instance one can associated only a tag bit to one out of N entries and ignore the other entries in the algorithm to update the AC counter.

3.3. Adaptative threshold fitting for the GEHL predictor

Experiments showed that the optimal threshold θ for the GEHL predictor varies for the different applications. For some of the benchmarks and using a 8-table GEHL predictor, the difference between using $\theta = 5$ or $\theta = 14$ as a threshold results in 0.5 misp/KI variations. However, we remarked that for most benchmarks there is a strong correlation between the quality of a threshold θ and the relative ratio of the number of updates on mispredictions NU_{miss} and the number of updates on correct predictions $NU_{correct}$: when NU_{miss} and $NU_{correct}$ are in the same range, θ is among the best possible thresholds for the benchmark.

Therefore, we implement a simple algorithm that adjusts the update threshold while maintaining the ratio $\frac{NU_{miss}}{NU_{correct}}$ close to 1. This algorithm is based on a single saturated counter TC (for threshold counter).

if ((p !=*Out*) {TC= TC + 1; if (TC == SaturatedPositive) {$\theta = \theta + 1$; TC=0;} }
if ((p == *Out*) & ($|S| \leq \theta$)) {TC= TC - 1; if (TC == SaturatedNegative){$\theta = \theta - 1$; TC=0;}}

Using a 7-bit counter for TC was found to be a good tradeoff.

3.4. The reference O-GEHL predictor

The simulations displayed in the paper are obtained through varying parameters on the 64 Kbits 8-table O-GEHL predictor presented in [22]. This predictor will be

FP-1	FP-2	FP-3	FP-4	FP-5
1.408	0.934	0.417	0.097	0.031
L 5	L 6	L 7	L 6	L 7
INT-1	INT-2	INT-3	INT-4	INT-5
0.730	5.595	9.032	1.042	0.343
L 2	S 7	S 8	L 2	L 11
MM-1	MM-2	MM-3	MM-4	MM-5
7.245	9.057	0.286	1.419	4.538
S 11	S 13	L 2	L 5	S 5
SERV-1	SERV-2	SERV-3	SERV-4	SERV-5
2.137	1.685	4.519	3.538	2.846
S 2	S 3	S 7	S 4	S 4

Table 3. Accuracy of the 64Kbits reference O-GEHL predictor (misp/KI)

referred to as the reference O-GEHL predictor. We recall below its characteristics.

A O-GEHL predictor featuring 8 tables would normally lead to 8 2K 4-bit counters tables and a 1K 1-bit tag table associated with Table T7, i.e. a total of 65 Kbits. For fitting in the 64Kbits storage budget, the reference O-GEHL features one table with only 1K counters, Table T1. Moreover, experiments showed that using 5-bit counters on the tables indexed with short history is beneficial. This allows to capture the behavior of very strongly biased branches. Therefore while respecting the 64Kbits storage budget constraint for the ChampionShip Branch Prediction, the reference O-GEHL predictor mixes 5-bit counters and 4-bit counters:

- Tables T0 and T1 implement 5-bit counters.
- Tables T2 to T7 implement 4-bit counters.
- Table T7 features an extra tag bit for half of its entries.

The characteristics of the reference O-GEHL predictor are summarized in Table 2. Using 200 as L(10) the maximum history length and 3 as L(1) is one of the best tradeoffs on the set of the benchmark traces. The simulation results in misp/KI for this reference branch predictor are illustrated on Table 3[2]. Table 3 also illustrates if the reference O-GEHL predictor is essentially using long or short (L or S) history length and the value reached by the update threshold at the end of the simulation of the trace.

3.5. Prediction computation time and hardware logic complexity on the O-GEHL predictor

3.5.1. Predictor response time The prediction response time on most global history predictors involves three components: the index computation, the predictor table read and the prediction computation logic.

The indexing functions used for the simulations illustrating this paper can be implemented using a single stage of 3-entry exclusive-OR gates (cf. Section 3.5.3 below). A 2-entry multiplexor is also used to select between the "short" and the long history length for the predictor tables concerned with dynamic history length fitting.

The predictor table read delay depends on the size of tables. For a 8Kbit table, a single cycle table read delay can be considered.

The prediction computation logic on the O-GEHL predictor consists in an adder tree. The delay to cross this logic depends on the number M of predictor tables and on the width of the counter. For the 8-component O-GEHL predictor we are considering as a reference in this paper, a 9-entry 5-bit adder tree is used. It can be implemented using a 3-stage carry save adder tree and a final 2-entry 8-bit adder. Therefore the overall response time of this computation logic is approximately the same as the response time as a 32-bit adder.

Typically, at equal storage budgets, the response time of the O-GEHL will be slightly longer than the response time of more conventional *gshare* or *2bcgskew* predictors, since the computation is slightly more complex.

3.5.2. Branch history management The O-GEHL predictor relies on using a very long branch history. This global branch history as well as the path history are speculatively updated and must therefore be restored on misprediction.

Checkpointing the complete branch history would be very costly for the O-GEHL branch predictor. However, one can use a circular buffer (for instance 512 bits) to store the branch history [11]. Then only the head pointer of the circular buffer has to be checkpointed. Restoring the branch history consists of restoring the head pointer.

3.5.3. Addressing the indexing functions complexity The O-GEHL predictor has been defined in order to capture correlation on very long histories in the 100-200 bits range. Fully hashing the 200-bit history, the 16-bit path history and 32 branch address bits to compute a 11-bit index for the reference O-GEHL predictor would normally require using 23 bit entry functions for computing each of index bit (for instance a 23-entry exclusive-OR tree). The delay for computing such functions can be prohibitive.

The index functions used for the simulations presented in this paper can be computed using single three-entry exclusive-OR gate for computing each index bit. We choose to ignore some of the address bits and some of the history bits as follows. For computing the hash function to index Table Ti, 2^n being the number of entries on Ti, we regularly pick 3n bits in the vector of bits composed with the

2 The slight discrepancy with the results published in [22] is associated with the absence of resetting the predictor when chaining the simulations of the different traces (see Section 2.2)

Table	T0	T1	T2	T3	T4	T5	T6	T7
short branch history length	L(0)=0	L(1)=3	L(2)=5	L(3)=8	L(4)=12	L(5)=19	L(6)=31	L(7)=49
long branch history length	-	-	L(8)=79	-	L(9)=125	-	L(10)=200	-
counter width (bits)	5	5	4	4	4	4	4	4
tag bit	-	-	-	-	-	-	-	0.5
entries	2K	1K	2K	2K	2K	2K	2K	2K
storage budget (bits)	10K	5K	8K	8K	8K	8K	8K	8K +1K (tags)

Table 2. Characteristics of the reference 64Kbits O-GEHL predictor

least significant bits of the branch address, the L(i) branch history bits and the min(L(i),16) path history bits. Then we simply hash this 3n bit vector in a n-bit index through a single stage of 3-entry exclusive-OR gates.

Experiments showed very limited accuracy degradation when using these simple hash functions instead of full hash of the branch address, the branch history and path history. Using a single stage of 2-entry exclusive-OR gates (i.e. picking only 2n bits) would result in 2-3 % global increase of the number of mispredictions on a 64Kbits predictor while using no exclusive-OR (i.e., picking n bits and directly using it as an index) would result in 10-11 % average increase of the number of mispredictions.

3.5.4. Miscellaneous hardware logic Hardware logic for the O-GEHL predictor also includes counter update logic and the logic for managing dynamic history length fitting and dynamic threshold fitting.

3.6. The O-GEHL predictor and local history

As for the perceptron predictor [9], one could combine both local and global histories on the O-GEHL predictor using the adder tree as the metapredictor. Combining local and global histories on the perceptron predictor was shown to result in a substantial accuracy benefit: some correlations that are not captured by the global history perceptron predictor are captured when local and global histories are used.

On the O-GEHL predictor, this phenomenom is much more limited. Some applications marginally benefits from using a local history. But for the 32kbits-1Mbit storage budget range considered in this paper, we did not find any partition of the storage budget between local and global history components that was leading to a total misprediction reduction compared with using only global history. In practice, using a very long global history in the 100-200 range allows to capture most of the correlations that are captured by local history.

The design and implementation of the hardware for maintaining speculative local history is very complex on a deep pipelined processor [26]: many occurrences of a single static branch can be in flight at the same time. Therefore, we advocate for using only global history on a O-GEHL branch predictor.

4. Performance evaluation and analysis of the O-GEHL predictor

As mentioned in the previous section, there are various degrees of freedom for designing a O-GEHL predictor. In this section, we first compare the O-GEHL predictor accuracy with the ones of previous state-of-the-art predictors, then we analyze these degrees of freedom.

4.1. Prediction accuracy against previous state-of-the-art branch predictors

On Figure 2, we report simulation results for the 8-component O-GEHL predictor, the 4-component O-GEHL predictor, the optimized *2bcgskew* predictor described in [21] and the path based neural predictor [6]. These two last predictors are often considered as the most efficient predictors previously proposed in the literature. For the O-GEHL predictor, for each predictor size, simulations were run for L(1) varying from 2 to 8 and the maximum history length (L(10) for 8-component, L(4) for 4-component), varying from 25 to 300 with step 25. We report here the simulation results for the best (L(1),L(10)) pair for each predictor size. For the optimized 2bcgskew predictor, the 4 history lengths for indexing the predictor components in a 2^{N+11} storage bits predictor are N, N, 4*N and 8*N (i.e. 5,5,20 and 40 for a 64Kbits predictor). For the path based neural predictor, we report simulation results for 256, 512, 1024 and 2048 entries predictors and increasing the history length till the total misprediction number reaches a minimum (i.e respectively for 38, 44, 58 and 58 history bits).

Simulation results are reported for predictor storage budgets ranging from 32 Kbits to 1 Mbit.

Both the 4-component and the 8-component O-GEHL predictor outperform both *2bcgskew* and the path based neural predictor. The shapes of the accuracy curves are very similar. There is a large benefit in enlarging the predictor from 32Kbits to 64 Kbits, increasing the predictor size up to 256 Kbits is still cost-effective, but further enlargening the predictor has very limited return.

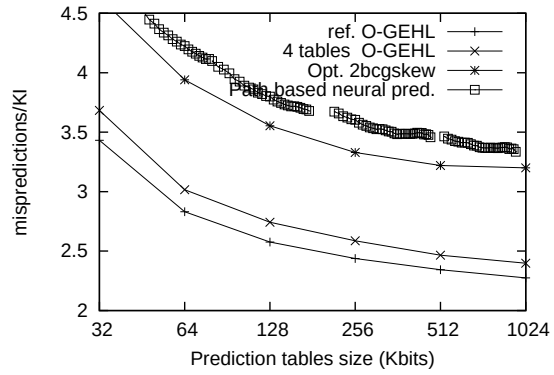

Figure 2. Average misprediction rates

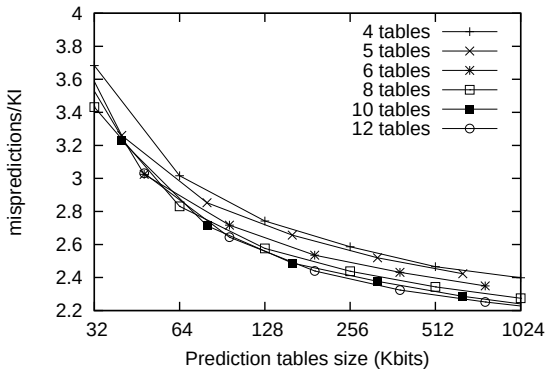

Figure 3. Varying the number of components on the O-GEHL predictor

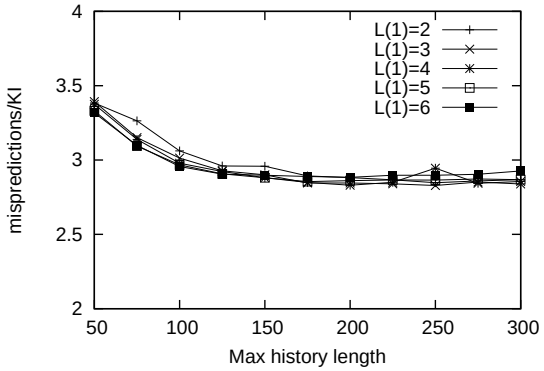

Figure 4. Impact of varying history length parameters

4.2. Impact of the number of components in the GEHL predictor

Figure 3 presents simulation results for O-GEHL predictors featuring 4 to 12 predictor tables. The best (L(1), maximum history length) pair is represented for each configuration.

For large storage budgets, using more components brings more accuracy: the quality of the prediction is augmented by the larger number of inputs for the computation of the prediction. For instance, a 192Kbits 12-component O-GEHL predictor essentially achieves the same accuracy as a 384 Kbits 6-component O-GEHL predictor (in average 2.43 misp/KI). On the other hand and for the set of considered benchmark traces, for small predictor storage budgets (equal or less than 48 Kbits), using 5 or 6 components is more cost-effective. For such a storage budget, doubling the number of predictor components, but halving each individual component storage budget does not bring any accuracy return: while some benchmarks benefit from the larger number of inputs for computing the prediction, the accuracy on other benchmarks is impaired by more conflicts on the predictor tables.

The effective complexity of the design of a predictor does not only depend on the overall storage budget of the predictor. The number of predictor components also influence the overall silicon area, the prediction computation time, the update logic complexity and the power budget. With the O-GEHL predictor, the designer can choose using any number of components between 4 and 12 and get high accuracy.

4.3. O-GEHL predictor and history lengths issues

4.3.1. Sensitivity to variation of history length parameters The O-GEHL predictor is robust to variations of history length parameters. Figure 4 illustrates this robustness for the reference O-GEHL predictor. On this figure, we vary L(10) and L(1), i.e the maximum history length and the geometric factor of the series of history lengths. One can chose any value in the interval [2,6] for L(1) and any value in the interval [125,300] for L(10) without significantly varying the average accuracy of the predictor on the benchmark set (best accuracy 2.83 misp/KI for L(1)= 4 and L(10)= 250, worst accuracy 2.95 misp/KI for L(1)=2 and L(10)= 125).

4.3.2. Impact of dynamic history length fitting Figure 5 illustrates the accuracy of the O-GEHL predictor assuming that dynamic history length fitting is enabled or not. When history length fitting is not used, there is no significant benefit in using an history longer than 100. One can note that dynamic history length fitting allows to reduce the average misprediction rate by approximately 0.2 misp/KI (best average 2.83 misp/KI against 3.03 misp/KI).

Flipping from long to short history is a rare event. Traces using "long" histories with the reference O-GEHL predictor are the 5 FP traces, INT-1, INT4, INT-5, MM-3 and MM-4.

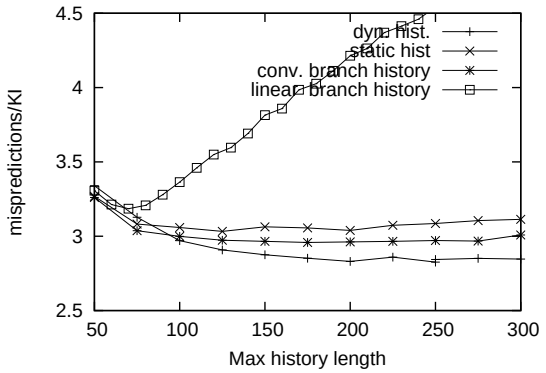

Figure 5. Benefits of using dynamic history length fitting; impact of path information

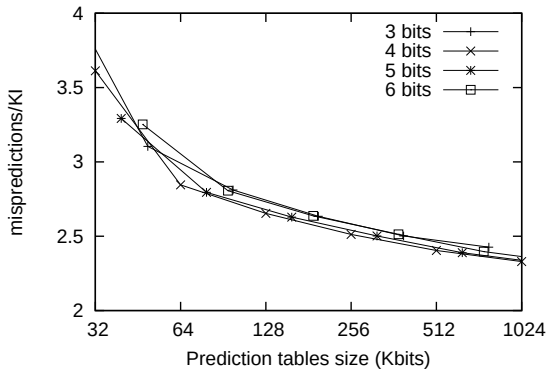

Figure 6. Varying counter width and predictor size

4.3.3. Impact of using path information Curve "conv. branch history" on Figure 5 illustrates the benefit of using the history information combining path and branch history described in Section 2.3. The average misprediction rate is reduced by approximately 0.1 misp/KI (best average 2.83 misp/KI against 2.96 misp/KI).

However the benefit is not uniform over the whole benchmark set, but is concentrated on a few benchmarks (INT-2, MM-1, MM-5, SERV-4).

4.3.4. Using geometric history length instead of linear history length In order to illustrate the benefit of using " geometric history length", Figure 5 also reports the simulation results using " linear history length" predictor , i.e., a predictor where the function indexing component T(i) is using i*L(1) history bits.

For maximum history lengths L(10) smaller than 50, using linear history lengths is competitive with using geometric history lengths. However using linear history lengths does not allow to accommodate very long history lengths in the hundreds of bits range with a reasonable accuracy.

4.3.5. Using geometric history length is a good tradeoff In order to check the possible accuracy limits of a predictor featuring the reference O-GEHL characteristics apart the use of geometric history length, we run an experiment to determine the best series of history lengths as possible for the benchmark set. The overall objective of this experience was to find a better family of series of history lengths.

The experiment was based on simulated annealing and several initial points were used and run for 4 days on a cluster of 20 biprocessor machines. Note that the configuration found through this experiment is biased towards the set of benchmark traces. The best history length series configuration that was found was $\{0, 2, 4, 9, 12, 18, 31, 54, 114, 145, 266\}$ and the average misprediction rate for this configuration was 2.79 misp/KI, i.e., a mere 0.04 misp/KI reduction compared with the reference O-GEHL predictor.

4.4. Counter width

On the perceptron predictor family, using 8-bit counters to represent the weights was shown to be a good tradeoff. On the O-GEHL predictor, using 4-bit counters is sufficient.

Figure 6 represents the accuracy of the O-GEHL predictor varying the predictor storage budgets and different counter widths. In this experiment, all the simulated predictors use the history lengths presented in Table 2.

This experiment shows that using 4-bit counters is a good tradeoff, even for moderate storage budgets. Figure 6 also clearly indicates that the main benefit of using wider counters is to smoothen the impact of aliasing on the predictor. For a storage budgets around 32 Kbits, 3-bit and 4-bit counters fail to deliver the same accuracy 5-bit counters for predictor. For a storage budget around 64 Kbits, using 4-bit counters is sufficient and for large predictors even using 3-bit counters brings competitive accuracy.

4.5. Update threshold

As the perceptron branch predictors, the O-GEHL predictor uses an adder tree and a partial update policy associated with a threshold on the computed prediction. The threshold proposed for the perceptron predictors [8], θ=1.93M+14, does not fit the O-GEHL predictor.

Figures 7 and 9 report simulation results assuming that dynamic threshold fitting is disabled and varying the counter widths and the update thresholds for respectively small (around 64 Kbits) and large (around 512Kbits) O-GEHL predictors. These figures show that the best threshold depends on both the counter width and the predictor size. Moreover, the best threshold also depends on the benchmark as illustrated on Figure 8.

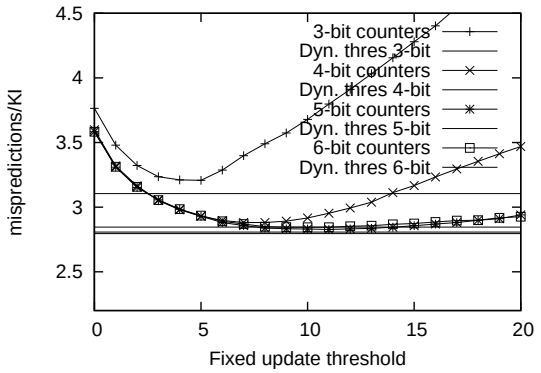

Figure 7. Varying the update threshold θ:8 2Kentries component O-GEHL predictor

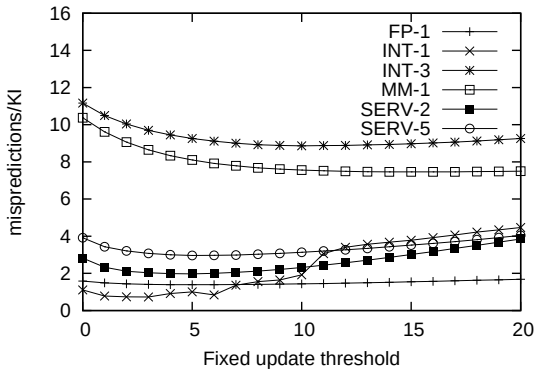

Figure 8. Varying the update threshold θ on the reference 64Kbits O-GEHL predictor: per benchmark results

Curves *3-bit* on Figures 7 and 9 illustrates an interesting phenomenon associated with aliasing. Using a small update threshold decreases the number of updates associated with correct predictions and therefore decreases the destructive aliasing impact. For relatively small predictors, destructive aliasing has a large impact on the predictor behavior, therefore using a small update threshold is better (Figure 7). On larger predictors, aliasing is more limited, stabilizing the prediction through updates on correct predictions has a higher accuracy impact than destructive aliasing (Figure 9).

Figures 7 and 9 also reports simulation results using our dynamic threshold fitting policy. Dynamic threshold fitting adapts the behavior of the predictor to the predictor size and the application. While it does not systematically outperform the best fixed threshold on a per benchmark basis, it outperforms the best fixed update threshold in average for the whole set of traces. In particular, dynamic threshold fitting appears to adapt the update threshold to the aliasing ratio through the following scenario. When the aliasing ratio increases on the predictor tables, the fraction of predictions with small absolute values tends to increase and therefore the number of updates on correct predictions also increases. Hence the dynamic threshold fitting algorithm tends to decrease the update threshold.

Dynamic threshold fitting is a general technique for global history predictors Experiments showed that the dynamic threshold fitting algorithm we have proposed for the O-GEHL predictor also works for the path based neural branch predictor as well as the global history branch predictor. For instance, on the path based neural predictor, our experiments showed that one can use 7-bit counters instead of 8-bit counters and still get a 2-3 % lower misprediction rate than when using the fixed update threshold predictor.

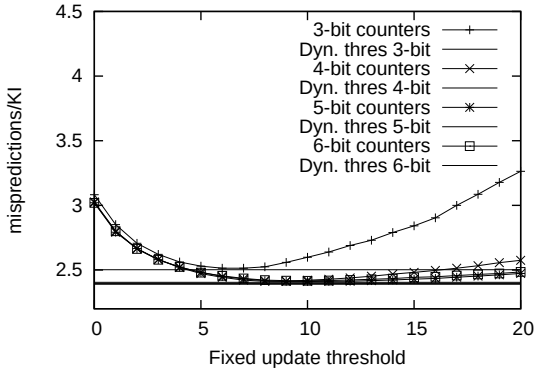

Figure 9. Varying the update threshold θ:8 16Kentries component O-GEHL predictor

5. In time prediction with the O-GEHL predictor

The prediction computation time on any complex branch predictor is longer than a single cycle on a high-performance processor. Therefore, many processors (e.g. Alpha EV6 [13] or EV8 [24]) have been designed using a fast, but relatively inaccurate predictor, backed with a more accurate but slower branch predictor. Such an approach results in a substantial loss of instruction fetch bandwidth.

However, global history (or path) branch predictor can be adapted to provide in-time prediction through ahead pipelining [25, 6, 23, 28]. Prediction computation can be started X cycles before the actual cycle T when the prediction is needed. Information available at cycle T-X is used for the prediction, but also some intermediate information (path or history) collected during the intermediate cycles

can be used at the end of the computation. In this section, we present a general principle to compute prediction in time on any most global history predictors. Then we present simulations results for an ahead pipelined O-GEHL predictor.

5.1. Ahead pipelining a global history branch predictor

If a global history predictor relies on the simultaneous read of several predictor components then it can be ahead pipelined to provide the prediction in time for use applying the principle describe below. This principle was already used in [25].

5.1.1. Prediction computation Let us assume that the computation prediction by the global history predictor requires X-1 cycles.

The prediction computation is done in two steps:

1. *Cycle T0-X: parallel computation of 2^m speculative predictions* The prediction computation is initiated at cycle T0-X i.e. X cycles ahead the use of the prediction. The prediction computation is initiated with the available information, i.e., the X-block ahead address and the X-block ahead history *h*. However, this X-block ahead information is not sufficient to provide accurate prediction. Therefore some information on the intermediate path from the block fetched at cycle T0-X and the block fetched at cycle T0 must also be used. A vector V of m bits of information on the intermediate path is used for the prediction: the value of this vector is unknown at T0-X, then 2^m *predictions p(i) corresponding to the 2^m possible values of the m-bit vector V are computed in parallel.*
2. *Cycle T0: effective prediction selection*: at the end of cycle T0-1, the 2^m predictions p(i) are available. At cycle T0, the effective information vector V is known and is used to select the correct prediction p(V). We found that the exclusive-OR of a few bits of the current block address and of the most recent history bits is a good tradeoff for vector V.

5.1.2. Resuming on a misprediction or an interruption On a misprediction (or an interruption), the X-block ahead global branch predictor cannot deliver the prediction in-time for the next X-1 cycles.

However, if the 2^m possible predictions associated with the 2^m possible m-bit information vectors on intermediate path have been checkpointed, then the checkpoint logic can deliver the predictions for the X-1 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 (see [25]).

Remark that for m smaller than 5 or 6, the storage cost of these 2^m possible predictions remains small compared with the volume of information to be checkpointed.

5.1.3. Hardware tradeoffs Global history branch predictors feature storages tables and hardware computation logic. In order to compute 2^m predictions in parallel, 2^m adjacent entries are read on each table instead of a single counter on a conventional one-block ahead predictor. 2^m copies of the prediction computation logic must also be implemented,

For predictors, with very limited prediction computation logic (e.g. a 4-bit entry gate for 2bcgskew [25]), the cost of these 2^m copies of the prediction computation logic remains small, even for m=6. For the O-GEHL predictor, the cost of duplicating the prediction computation logic is higher, but is still affordable for $m \leq 4$.

5.2. Ahead pipelining the O-GEHL predictor

In this section, we will assume that the O-GEHL predictor is part of an instruction address generator in charge of delivering a basic block address and its size on each cycle as the decoupled instruction fetch front-end proposed in [20].

For medium storage budget such as 64Kbits, the response time of the reference O-GEHL predictor should be around 3 cycles including 1 cycle for computing the index functions, 1 cycle for reading the predictor tables (these tables are 8Kbit tables) and 1 cycle to compute the prediction through the adder tree.

Figure 10 illustrates the accuracy for 3,4 and 5-block ahead pipelined O-GEHL predictors assuming respectively that 2, 3 and 4 bits of intermediate information (i.e., the intermediate branch history information) are used to select the final prediction. The best (L(1),L(10)) pairs are presented for each configuration.

As expected, some loss of accuracy is encountered particularly for small predictors compared with the ideal 1-block ahead predictor. However this accuracy loss is relatively small, and decreases with the size of the predictor. For the 3-block ahead O-GEHL predictor, it decreases from 0.16 misp/KI for a 32 Kbits predictor to 0.06 misp/KI for the 1 Mbit predictor. For small predictors, this loss of accuracy is essentially associated with extra interferences on tables indexed with short histories.

The robustness of the O-GEHL predictor on variations of the (L(1), L(10)) pairs described in Section 4.3.1 still holds for the ahead pipelined O-GEHL predictor as illustrated for a 4-block ahead 64 Kbits O-GEHL predictor on Figure 11.

6. Related works and summary

The design of the O-GEHL predictor capitalizes on the previous research on branch prediction during the 15 last

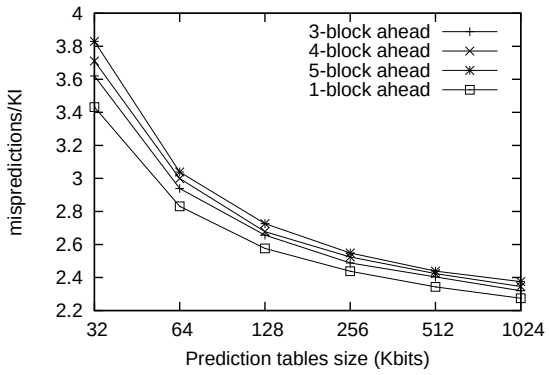

Figure 10. Accuracy of ahead pipelined O-GEHL predictors

years. We borrowed multiple techniques from the previously proposed branch predictors. First, the O-GEHL predictor uses multiple predictor components indexed with different history lengths. The use of multiple history lengths in a single branch predictor was initially introduced in [16], then was refined by Evers et al. [3] and further appeared in many proposals. On a multiple predictor component, the accuracy of each component may be significantly impaired by aliasing [30]. Using multiple components with close history lengths was recognized to reduced the aliasing impact (e.g. the skewed branch predictor[18], the agree predictor [27], the bimode predictor [14], the YAGS predictor [2]). By using several short history components, the O-GEHL predictor suffers very limited accuracy loss due to aliasing on tables indexed with short histories.

The O-GEHL predictor also borrows techniques from the perceptron-like predictors. As perceptron-like predictors [29, 8, 9, 6, 23, 28], the O-GEHL predictor does not use storage based metapredictors, but computes the prediction

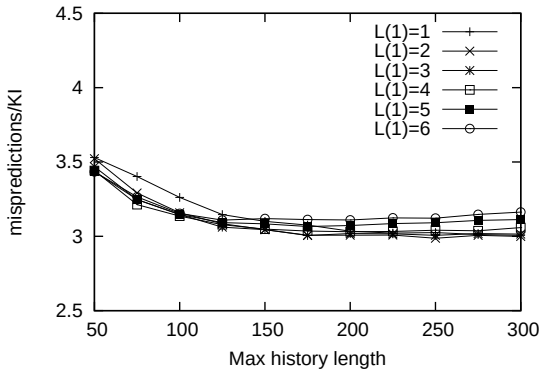

Figure 11. Varying history length parameters on a 4-block ahead O-GEHL branch predictor

through an adder tree. This adder tree does not "waste" storage space for metaprediction. Associating a single weight to a group of history bits was first proposed in [23], then independently refined in [28] and in this work.

As the perceptron predictor, the O-GEHL predictor uses a partial update policy considering a threshold. We introduce dynamic update threshold fitting that adapts the threshold to the behavior of the application. Dynamic update threshold fitting was shown to be beneficial for the O-GEHL predictor as well as for predictors from the perceptron predictor family.

The O-GEHL predictor makes a better usage of the storage budget than the previously proposed global history predictors. Branch outcomes are statistically more correlated with recent branches than with very old branches. By using **geometric** history lengths, the O-GEHL predictor spends most of its storage budget on the shortest history components. On demanding applications, our reference O-GEHL predictor uses $\frac{5}{8}$ths of its storage budget for capturing correlation on history shorter than or equal to 12 bits. This has to be compared with a 48-bit global history perceptron predictor, which uses only $\frac{1}{4}$th of the storage budget for weights associated with the 12 most recent branches or with a *2bcgskew* predictor that devotes most of its storage to long histories. Moreover on less demanding applications, the association of a simple form of dynamic history length fitting [12] with the use of geometric history lengths allows the O-GEHL predictor to capture correlation on very old branch outcomes. Furthermore, this association allows the O-GEHL predictor to be very robust to the choice of history length parameters. Unlike, most previously proposed global history predictors, overdimensioning the maximum history length does not dramatically decrease the predictor accuracy.

The hardware complexity of the computation logic of the O-GEHL predictor is also significantly lower than the ones of the previous perceptron-like predictors since it uses a smaller number of entries in the adder tree and narrower counters.

The design space of cost-effective O-GEHL predictors is large. Depending on implementation tradeoffs one can use from 4 to 12 tables, one can use 4-bit, 5-bit or even 3-bit counters. High level of accuracy are obtained for a broad spectrum of maximum history lengths, for instance any length in the 125-300 range for a 64 Kbits O-GEHL predictor.

As the other global branch history predictors[25, 6, 23, 28], the O-GEHL predictor can be ahead pipelined to provide predictions in time for the instruction fetch engine. The accuracy of a 3-block ahead pipelined O-GEHL predictor remains in the same range as the accuracy as the 1-block ahead O-GEHL predictor, while the complexity of the extra hardware needed to computed the prediction re-

mains limited. It should be noted that the systolic-like ahead pipelining technique originally proposed by Jiménez [6] for the path based neural predictor and later refined by Tarjan and Skadron [28] for the hashed perceptron could also be adapted to the O-GEHL predictor.

References

[1] V. Desmet, H. Vandierendonck, and K. D. Bosschere. A 2bcgskew predictor fused by a redundant history skewed perceptron predictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[2] A. N. Eden and T. Mudge. The YAGS branch predictor. In *Proceedings of the 31st Annual International Symposium on Microarchitecture*, Dec 1998.

[3] M. Evers, P. Chang, and Y. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In 23^{rd} *Annual International Symposium on Computer Architecture*, pages 3–11, 1996.

[4] H. Gao and H. Zhou. Adaptive information processing: An effective way to improve perceptron predictors. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[5] N. Gloy, C. Young, J. B. Chen, and M. D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *ISCA '96: Proceedings of the 23rd annual international symposium on Computer architecture*, pages 12–21. ACM Press, 1996.

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

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

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

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

[10] D. A. Jiménez. Idealized piecewise linear branch prediction. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[11] S. Jourdan, T.-H. Hsing, J. Stark, and Y. N. Patt. The effects of mispredicted-path execution on branch prediction structures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, October 1996.

[12] T. Juan, S. Sanjeevan, and J. J. Navarro. A third level of adaptivity for branch prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 30 1998.

[13] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[14] C.-C. Lee, I.-C. Chen, and T. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, Dec 1997.

[15] G. Loh. The frankenpredictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[16] S. McFarling. Combining branch predictors. Technical report, DEC, 1993.

[17] P. Michaud. A ppm-like, tag-based predictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[18] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, June 1997.

[19] R. Nair. Dynamic path-based branch prediction. In *International Symposium on Microarchitecture (MICRO-28)*, pages 15–23, 1995.

[20] G. Reinman, B. Calder, and T. Austin. A scalable front-end architecture for fast instruction delivery. In *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA'99)*, 1999.

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

[22] A. Seznec. The o-gehl branch predictor. In *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[23] A. Seznec. Revisiting the perceptron predictor. Technical Report PI-1620, IRISA Report, May 2004.

[24] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeidès. Design tradeoffs for the ev8 branch predictor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, 2002.

[25] 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.

[26] K. Skadron, M. Martonosi, and D. Clark. Speculative updates of local and global branch history: A quantitative analysis. *Journal of Instruction-Level Parallelism, vol. 2, Jan. 2000*, January 2000.

[27] E. Sprangle, R. Chappell, M. Alsup, and Y. Patt. The agree predictor: A mechanism for reducing negative branch history interference. In 24^{th} *Annual International Symposium on Computer Architecture*, 1995.

[28] D. Tarjan and K. Skadron. Revisiting the perceptron predictor again. Technical Report CS-2004-28, University of Virginia, September 2004.

[29] L. N. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings.*, 1999.

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