Journal of Instruction-Level Parallelism 9 (2007) 1-18 Submitted 04/09/07; published 04/30/07

PMPM: Prediction by Combining Multiple Partial Matches

Hongliang Gao HGAO@CS.UCF.EDU
Huiyang Zhou ZHOU@CS.UCF.EDU
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida 32816-2362
Voice (407) 823-5210

Abstract

The prediction by partial matching (PPM) algorithm has been well known for its high prediction accuracy. Recent proposals of PPM-like predictors confirm its effectiveness on branch prediction. In this paper, we introduce a new branch prediction algorithm, named Prediction by combining Multiple Partial Matches (PMPM). The PMPM algorithm selectively combines multiple matches instead of using the longest match as in PPM. We analyze the PPM and PMPM algorithms and show why PMPM is capable of making more accurate predictions than PPM.

Based on PMPM, we propose both an idealistic predictor to push the limit of branch prediction accuracy and a realistic predictor for practical implementation. The simulation results show that the proposed PMPM predictors achieve higher prediction accuracy than the existing PPM-like branch predictors such as the TAGE predictor. In addition, we model the effect of ahead pipelining on our implementation and the results show that the accuracy loss is relatively small.

1. Introduction

Given its importance on high performance microprocessor design, branch prediction has been extensively studied. As analyzed in [1],[11], the prediction by partial matching (PPM) algorithm [2], originally proposed for text compression, can achieve very high prediction accuracy for conditional branches. Recent proposals of PPM-like predictors by Michaud and Seznec [18], [10] confirm the effectiveness of PPM and show that PPM-like predictors outperform many state-of-art branch predictors.

In this paper, we detail our predictor design, both idealistic and realistic, that participated in the 2nd JILP Championship Branch Prediction Competition (CBP-2) [5]. The key idea of our proposed design is to improve PPM-like branch predictors by combining multiple partial matches rather than using the longest partial match as in the PPM algorithm. We first examine why the longest partial match may lead to suboptimal prediction accuracy. We then develop a prediction algorithm to selectively combine multiple partial matches (PMPM) with an adder tree.

Based on the PMPM algorithm, we develop an idealistic predictor to push the limit on branch prediction accuracy and a realistic predictor for practical implementation. The idealistic design explores extremely long branch history and shows the potential of the PMPM algorithm. The realistic design is based on recently developed PPM-like TAGE branch predictors [18] and it has similar hardware complexity compared to the TAGE predictor. Besides exploiting correlation from various global branch histories, our design enables efficient integration of local history information. The experimental results show that the proposed designs can achieve higher accuracy than TAGE predictors. Finally, we implement ahead pipelining to evaluate the impact of the access latency of the proposed PMPM predictor and the results show that with ahead pipelining, our design can provide a prediction every cycle with relatively small accuracy loss.

The rest of this paper is organized as follows. In Section 2 we study the PPM branch prediction algorithm and introduce the PMPM algorithm. An idealistic PMPM predictor is presented in Section 3. Section 4 describes our design of the realistic PMPM predictor. The design space of the realistic predictor is explored in Section 5. Finally, Section 6 concludes the paper.

2. Prediction by Combining Multiple Partial Matches

In this section, we study the performance of using the PPM algorithm for branch prediction and introduce a prediction algorithm to selectively combine multiple partial matches (PMPM) with an adder tree instead of using the longest match as in PPM.

2.1. PPM with the (Confident) Longest Match

In the PPM algorithm, a Markov model is used to capture the statistical behavior of inputs and to make a prediction accordingly. When used for branch prediction, the Markov model keeps track of branch histories and uses the longest partial match to make a prediction. The assumption behind using the longest match is that longer history provides a more accurate context to determine the incoming branch behavior.

However, since branches exhibit non-stationary behavior, partial context matches may not be sufficient to accurately predict branch outcomes. To examine this effect, we implemented a PPM-based branch predictor as described in [11], in which global branch histories are served as contexts for branches. We assign a signed saturating prediction counter within the range of [-4, 4] for each (branch address, history) pair. The range of [-4, 4] is selected because it produces the highest prediction accuracy in our setup for the PPM algorithm with the maximum history length as 40. The prediction counter is incremented if the branch outcome is taken and decremented otherwise. When both of the branch address and history are matched, the corresponding prediction counter is used to make a prediction. When there are multiple history matches for the same branch with different history lengths, the prediction counter associated with the longest history is chosen.

In order to show that the longest match may not be the best choice for branch prediction, we implemented another scheme, in which the prediction counter with the longest-history match is not always selected to provide the prediction. Instead, we use the prediction counter as a confidence measure of the potential prediction. Only when the prediction counter is a non-zero value, it can be selected to make a prediction. We call such a scheme as PPM with the longest confident match. We simulate both schemes, i.e., PPM with the longest match and PPM with the longest confident match, and report the misprediction rate reductions achieved by the longest confident match over the longest match in Figure 1. The misprediction rates are measured in the number of mispredictions per 1000 instructions (MPKI). In this experiment, the maximum length of the global history register (GHR) is set as 40.

From Figure 1, we can see that the confidence-based PPM has lower misprediction rates than the PPM scheme for all the traces except *vortex*, which implies that the longest match used in PPM may lead to suboptimal prediction accuracy.

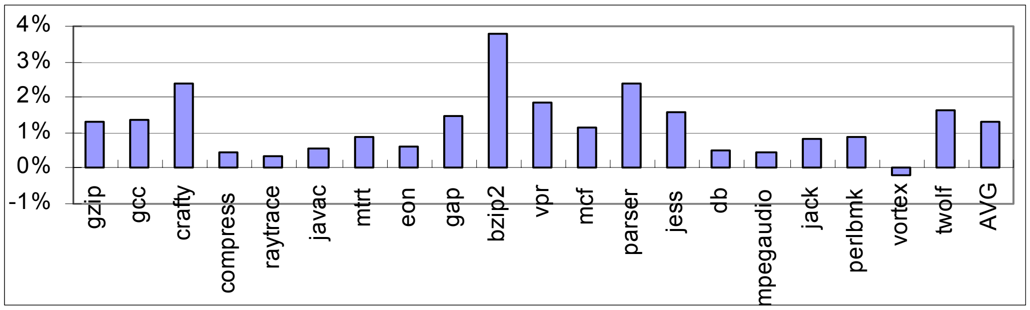

Figure 1: Misprediction rate reductions achieved by using the confident longest match over the longest match in a PPM-based branch predictor (Max History Length = 40).

2.2. Prediction by Combining Multiple Partial Matches

From the experiments with the PPM-based branch predictors, we observed that when there are multiple matches, i.e., matches with various history lengths, in the Markov model, the counters may not agree with each other and different branches may favor different history lengths. Such adaptivity is also reported in [9] and explored in some recent works, such as [12]. Based on this observation, we propose to use an adder tree to combine multiple partial matches in a PPM-based predictor (other combination schemes including linear combination have been explored in [3] and the adder tree is selected for CBP-2 due to its effectiveness and simplicity in implementation) and we call this novel prediction algorithm as Prediction by combining Multiple Partial Matches (PMPM). In a PMPM predictor, we select up to L confident longest matches and sum the counters to make a prediction. Figure 2 shows the average misprediction rates of the proposed PMPM predictors with L varying from 1 to 41 and the original PPM predictor with the longest match. The maximum history length is set as 40 in this experiment.

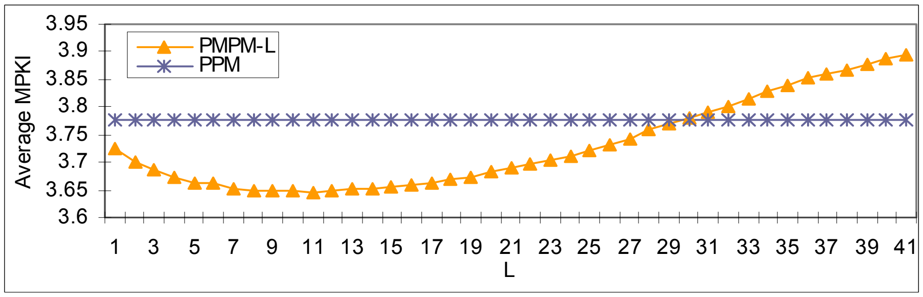

Figure 2: Misprediction rates of PMPM-L predictors and the original PPM predictor

From Figure 2, it can be seen that on average, combining multiple partial matches can provide higher prediction accuracy than utilizing a single partial match. Combining too many partial matches, on the other hand, can be harmful since many low-order Markov chains are included, which are susceptible to noises.

3. The Idealistic PMPM Predictor

Similar to PPM predictors, PMPM predictors require extensive history information and the number of different (branch address, history) pairs increase exponentially with the history length. Therefore, modeling an idealistic PMPM predictor to push the limit of branch prediction accuracy is still a challenge. In CBP2, we developed an idealistic PMPM predictor which explores extremely long global history (651) information with acceptable requirements on memory storage and simulation time.

3.1. Predictor Structure

The overall predictor structure is shown in Figure 3. We use a per-branch tracking table (PBTT) to record some basic information of each static branch. PBTT is a 32k-set 4-way cache structure. Each PBTT entry includes the branch address, LRU bits for replacement, a branch tag, a 32-bit local history register (LHR), a meta counter used to select either GHR-based or LHR-based predictions, a bias counter, and a simple (bias) branch predictor. A unique tag is assigned to each static branch and it is used to replace the branch address in index and tag hashing functions. Since there are a large number of highly biased branches, we use a simple bias branch predictor to filter them out. The simple bias branch predictor detects fully biased (always taken or always not taken) branches. A branch only accesses the main PMPM predictor when it is not fully biased. For those branches, PBTT sends the branch tag, LHR, the meta counter, and the bias counter to the PMPM predictor.

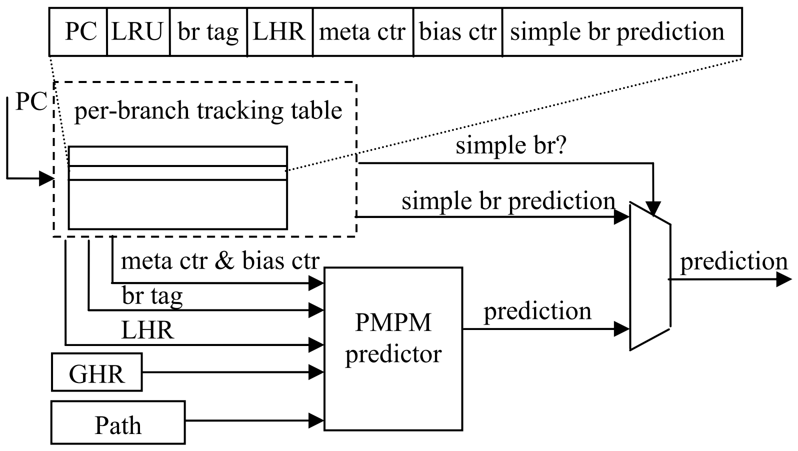

Figure 3: The overall idealistic PMPM branch prediction scheme

The PMPM predictor is used to implement the PMPM algorithm with extremely long history information. The structure of it is shown in Figure 4. There are three prediction tables in the PMPM predictor. A short-GHR table is used to store information corresponding to the most recent 32-bit global history. A long-GHR table is used to store information corresponding to longer global histories. We also use a LHR table to make local-history-based predictions and the LHR table uses 32-bit local histories. Those three prediction tables are 4-way set-associative structures and each entry has a tag, an LRU field, a prediction counter (*ctr*), a usefulness counter (*ubit*), and a benefit counter (*bf*). In order to capture very long global histories, we use geometric history lengths [12] in the long GHR table.

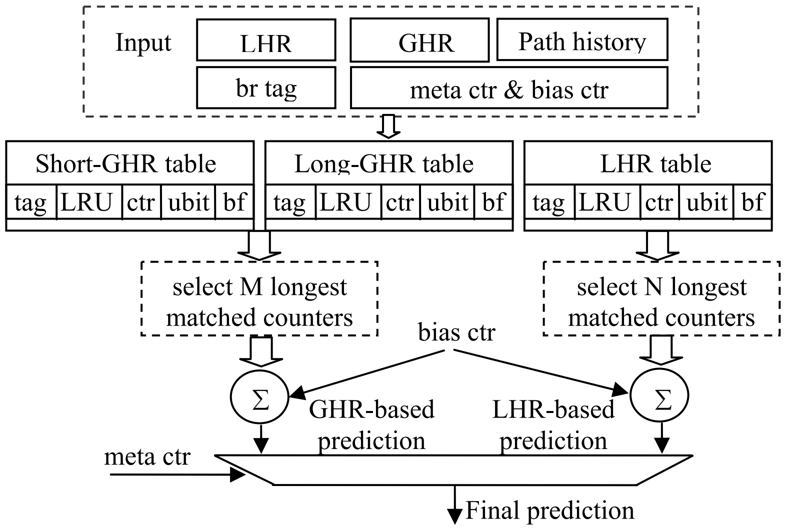

Figure 4: An optimized idealistic PMPM predictor.

3.2. Prediction Policy

We hash the branch tag, global history, path history, history length to get the index and tag to access each table in the PMPM predictor. We use similar hash functions to those in [18] and [8] with empirically selected prime numbers. The index and tag hash functions use different primary numbers.

We use the short GHR table to store global histories with lengths from 1 to 32. For longer histories, we use 21 different lengths in order to reduce the storage requirement. Both short and long GHR tables use 32-bit path histories. *M* of the hit counters (i.e., the prediction counters with a tag match) are summed up with the bias counter to generate the GHR-based prediction. The LHR prediction table works the same way as the short GHR table with the selection of *N* hit counters. The final prediction is selected by the meta counter.

3.3. Update Policy

The prediction counters in the prediction tables are updated if they have a tag match with the current branch. The tag contains both branch address and context information as described in Section 3.2.

In the case of a miss, i.e., when the branch history has not been retained in the tables, new entries are allocated in the following manner:

- For the short-GHR table, we assign new entries when the meta counter indicates that the LHR prediction table is not sufficient to make correct predictions. When the meta counter is saturated to the direction of using the LHR-based prediction and the prediction is correct, we assume that using the LHR-based prediction is sufficient.
- For the long-GHR table, we assign new entries when the overall prediction is wrong.
- For the LHR table, we always assign new entries.

3.4. Optimizations

In order to further improve the accuracy of the proposed idealistic PMPM predictor, we track the usefulness of each prediction counter using the usefulness (*ubit*) and benefit (*bf*) fields in the same entry. The *ubit* shows whether the prediction counter agrees with the branch outcome and the benefit counter shows whether the inclusion of the prediction counter is beneficial. The reason is that for some prediction counters, the inclusion has no or harmful impact on the final prediction since other prediction counters may be sufficient to make a correct prediction.

With the usefulness and benefit counters, we make two predictions: a prediction (*ubit_pred*) obtained by selecting the useful (*ubit* >= 0) counters and a prediction (*final_pred*) by selecting useful and beneficial (*ubit* >= 0 && *bf* >= *0*) counters. The *final_pred* is used as the final prediction. The *ubit_pred* is used to update the benefit counter as follows:

If the prediction counter was not included, the *ubit_pred* would change from correct to wrong. In this case, we increase the corresponding benefit counter.

If the prediction counter was not included, the *ubit_pred* would change from wrong to correct. In this case, we decrease the corresponding benefit counter.

Otherwise, the inclusion of the prediction counter has no impact on the prediction. We will not update the benefit counter in this case.

3.5. Prediction Accuracy

We simulate the proposed idealistic PMPM predictor with the configurations shown in Table 1. The configurations are based on our empirical tuning under the memory consumption and simulation time constrains of CBP2.

Short-GHR Prediction Table	1M sets, 4-way
Long-GHR Prediction Table	2M sets, 4-way
LHR Prediction Table	512K sets, 4-way
Max # of selected counters for GHR matches	7
Max # of selected counters for LHR matches	16
Minimum GHR length of the geometric history lengths used by the Long-GHR Table	38
Maxmum GHR length of the geometric history lengths used by the Long-GHR Table	651
Range of a prediction counter	[-6, +6]
Range of a ubit	[-1, +1]
Range of a benefit counter (bf)	[-31, +31]

Table 1: Idealistic Predictor Configuration.

With the configuration shown in Table 1, the simulator consumes around 226M bytes of memory and takes 1.65 hours to finish all CBP2 distributed traces on a Xeron 2.8Ghz computer. The final misprediction rates (measured in MPKI) of the idealistic PMPM predictor (labeled as "*IPMPM-GL*") are shown in Table 2. In order to show the impact of using local branch histories, we also provide the performance of using only global histories (labeled as "*IPMPM-G*"). For comparison, Table 2 also includes the misprediction rates of the PPM predictor (labeled as "*PPM*"). The PPM predictor is implemented using the idealistic predictor presented in Figure 4 except that the longest match is used instead of combining multiple matches. From Table 2, we can see that with the same predictor structure, the PMPM algorithm achieves significantly higher prediction accuracy than PPM for all traces except *perlbmk*. The misprediction rates reduction is up to 20.9% (*gzip*) and 15.2% on average. Although the idealistic PMPM predictor can explore very long global histories, we notice that local histories are still helpful for all traces. Without local histories, the average misprediction rate is increased from 2.824 MPKI to 2.969 MPKI.

Trace	PPM	IPMPM-GL	IPMPM-G	Trace	PPM	IPMPM-GL	IPMPM-G
gzip	11.977	9.470	10.334	*vpr*	10.096	8.273	8.318
gcc	2.748	2.594	2.615	*mcf*	9.258	7.688	7.714
crafty	2.083	1.794	1.848	*parser*	4.404	3.928	4.201
compress	6.526	5.167	5.522	*jess*	0.302	0.290	0.337
raytrace	0.274	0.272	0.368	*db*	2.721	2.199	2.263
javac	1.054	0.930	1.040	*mpegaudio*	1.051	0.922	1.041
mtrt	0.326	0.318	0.396	*jack*	0.484	0.472	0.524
eon	0.246	0.239	0.292	*perlbmk*	0.177	0.192	0.335
gap	1.164	1.081	1.446	*vortex*	0.090	0.087	0.152
bzip2	0.036	0.032	0.040	*twolf*	11.591	10.529	10.590
Average	PPM: 3.330		IPMPM-GL: 2.824		IPMPM-G: 2.969		

Table 2: Misprediction rates of the idealistic predictors using the PPM and PMPM algorithms.

3.6. Impact of Loop Branches in the Idealistic PMPM Predictor

It has been shown that a loop predictor is helpful for perceptron predictors [3] and the TAGE predictor [12], [13]. In this experiment, we study the impact of augmenting a loop predictor [19] to the idealistic PMPM predictor. In order to predict regular loop branches, we add a loop predictor in the PBTT table to track the branch outcome pattern for each static branch. If the pattern shows the behavior of a loop branch (e.g., taken, taken, …, not-taken) and has the same number of iterations successively for at least 8 time, we will assume it is a loop branch. The average misprediction rates of the idealistic PMPM predictors with the loop predictor are shown in Table 3. For the IPMPM-G predictor that does not use local histories, the loop predictor helps to reduce the misprediction rate by 0.068 MPKI (from 2.969 to 2.901 MPKI). However, for the IPMPM-GL predictor with 32-bit local histories the benefit of loop predictor is very marginal (0.006 MPKI). Since the IPMPM-GL predictor is able to capture small loops (number of iterations is less than or equal to 31) with local histories, the benefit of extra loop prediction support is only useful for loops with a higher number of iterations. As shown in Table 3, if the loop predictor is only used to capture small loops by limiting number of iterations less than 32, there is no improvement over the IPMPM-GL predictor. When a loop predictor is included to only target at the loops with numbers of iterations larger than 32, the average misprediction rate is reduced from 2.824 to 2.815 MPKI.

Predictor	AVG MPKI
IPMPM-G	2.969
IPMPM-GL	2.824
IPMPM-G + Loop	2.901
IPMPM-GL + Loop	2.818
IPMPM-GL + Loop (# of iterations <= 31)	2.827
IPMPM-GL + Loop (# of iterations > 31)	2.815

Table 3: Misprediction rates of the idealistic PMPM predictors without / with loop branch prediction.

4. Realistic PMPM Predictor

In this section, we present our PMPM branch predictor design for practical implementation. As discussed in Section 2, the PMPM algorithm is built upon PPM. Therefore, we choose to develop our design based on the recently proposed PPM-like branch predictors [10], [18], the TAGE branch predictor [18] in particular.

4.1. Predictor Structure

The overall structure of the proposed PMPM predictor is shown in Figure 5. The predictor structure is very similar to a TAGE predictor, except that a local history prediction table is incorporated. The prediction and update policies, however, are completely redesigned to implement the PMPM algorithm.

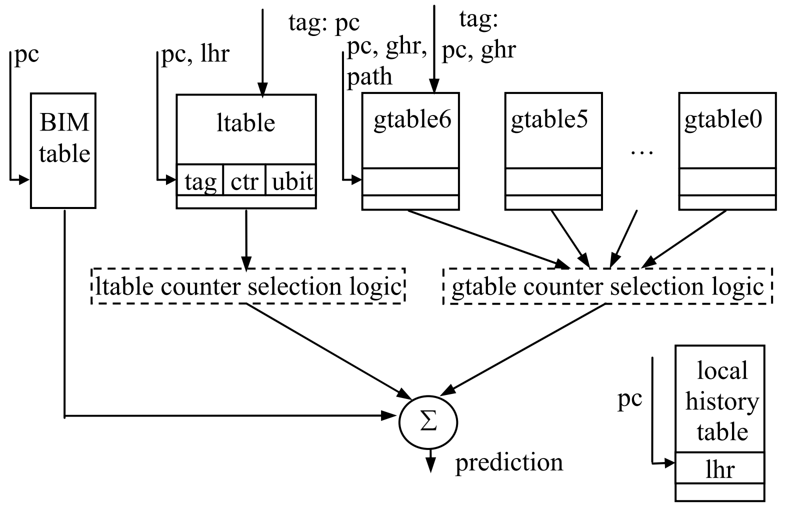

Figure 5: The practical PMPM predictor.

As shown in Figure 5, the PMPM predictor contains a bimodal prediction table [21], seven global prediction tables (labeled as "*gtable0*" to "*gtable6*") indexed by the branch address, global history and path history, and a local prediction table (labeled as "*ltable*") indexed by the branch address and local history. Geometrical history lengths [12] are used for the global prediction tables: *gtable6* is associated with the shortest global history and *gtable0* is associated with the longest global history. Each entry of the global and prediction tables has three fields: a *tag*, a signed saturated prediction counter (labeled as "*ctr*") and an unsigned saturated counter (labeled as "*ubit*") to track the usefulness of the prediction counter. Index and tag hashing functions for the global prediction tables are the same as those used in TAGE predictors. The local prediction table uses hashed branch address and local history as the index and the XOR-folded (i.e., PC ^ (PC >> M)) branch address as the tag.

4.2. Prediction Policy

The first phase of prediction is to calculate indexes and tags for each table. Among the entries that have tag matches (i.e, the hit entries), we select out up to 4 prediction counters from global prediction tables and sum those counters together with the prediction counter from the local prediction table, if there is a hit, and the counter from the bimodal table. If the sum is zero, we use the prediction from the bimodal table. Otherwise we use the sign of the sum as the prediction. In order to reduce the latency of counter selection, we devise a simple policy to select up to 4 counters from the global prediction tables rather than selecting several prediction counters with longest matches as used in the idealistic PMPM predictor. We divide the global prediction tables into 4 groups, (*gtable6*, *gtable5*), (*gtable4*, *gtable3*), (*gtable2*, *gtable1*) and (*gtable0*), and select out the longer match from each group.

The prediction counter from the local prediction table (*ltable*) is used only if its usefulness (*ubit*) is larger than 1.

The (tag-less) bimodal counter is always used in the summation process.

4.3. Update Policy

The prediction counter in the bimodal table is always updated. The update policies of the global prediction tables and the local prediction table are described as follows.

Similar to the perceptron predictor [5], [7], and the O-GEHL predictor [12], the prediction counters of the global prediction tables are updated only when the overall prediction is wrong or the absolute value of the summation is less than a threshold. We also adopt the threshold adaptation scheme proposed in the O-GEHL predictor to dynamically fit different applications. We only update those counters that have been included in the summation. At the same time, for each of these prediction counters, the associated *ubit* counter is incremented when the prediction counter makes a correct prediction. In the case of a misprediction, we try to allocate a new entry. The new entry will be selected from tables where the branch misses. We select one entry that has a zero *ubit* from those tables as a new entry allocated for the current branch. If there are multiple entries with zero *ubits*, we select the one with the shortest history length. If there is no entry with a zero *ubit*, we don't allocate a new entry. At last, for each entry corresponding to a longer history than the longest match, its *ubit* counter is decremented.

If current branch hits in the local prediction table, we always update the prediction counter. The *ubit* counter is decremented if the corresponding prediction counter makes a wrong prediction. If the prediction counter makes a correct prediction, we increment *ubit* only when the overall prediction is wrong. If the current branch does not hit in the local prediction table and the overall prediction is wrong, we will try to allocate a new entry in the local prediction table. If the indexed entry has a zero *ubit*, a new entry is allocated. Otherwise, its *ubit* counter is decremented.

The base update policy described above is also improved by two optimizations. First, we modify the update policy so that in each group of two global prediction tables, a new entry will not be allocated in the table with shorter history length if the branch hits in the table with longer history length. Second, for applications with a large number of hard-to-predict branches, some otherwise useful entries could be evicted due to frequent mispredictions using the base update policy. To address this issue, we use a misprediction counter to periodically detect those applications/program phases with a large number of hard-to-predict branches. For those application/phases, we slightly vary the update policy: on a misprediction, we don't decrement the *ubit* counters in those prediction tables that have tag misses if we already allocate a new entry; and we will decrement *ubit* of the longest match only if its prediction counter is wrong. In this way, we limit the chance of useful entries to be victimized by those hard-to-predict branches.

4.4. Hardware Complexity and Response Time

Similar to TAGE and O-GEHL predictors, the response time of the proposed PMPM predictor has three components: index generation, prediction table access, and prediction computation logic. In the proposed PMPM predictor, we use similar index functions to those in TAGE. The index and tag generation for the local prediction table has two sequential parts, the local branch history (LHR) table access and simple bitwise XOR of the LHR and the branch address for index. Since we use a small 1K-entry tagless LHR table with short LHRs, we assume the overall latency for generating index for the local prediction table is comparable to the complex index functions for global prediction tables, i.e., one full cycle. The prediction table access is also similar to the TAGE predictor. Rather than selecting the longest two matches (for the optimization related to the newly allocated entries) as in the TAGE predictor, we use an adder tree with up to 6 5-bit values to calculate the prediction, which should have similar latency to the prediction computation of the O-GEHL predictor. Therefore, we expect the response time of the proposed PMPM predictor should be very close to the TAGE or O-GEHL predictors.

4.5. Ahead Pipelining

As the prediction latency is more than one cycle, we use ahead pipelining [16] to generate one prediction per cycle. Assuming 3-cycle prediction latency, our 3-block ahead pipelining scheme for the PMPM predictor is shown in Figure 6. The impact of ahead pipelining on the prediction accuracy of the proposed PMPM predictors will be examined in Section 5.5.

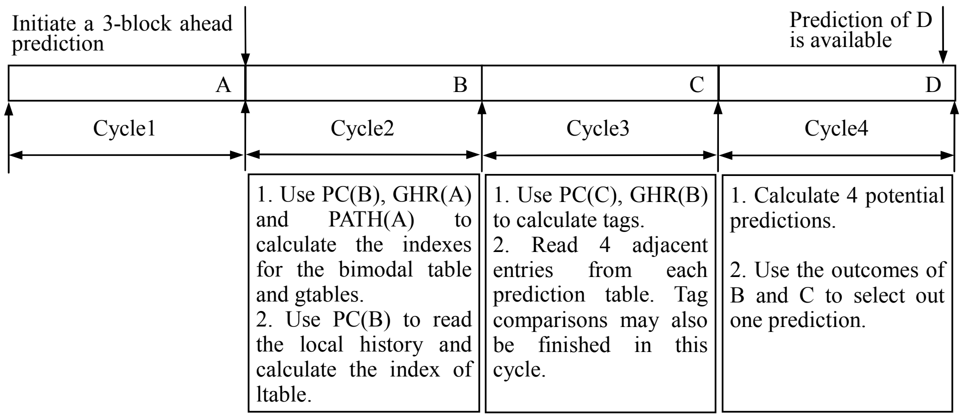
Initiate a 3-block ahead prediction
Prediction of D is available
A
B
C
D
Cycle1
Cycle2
Cycle3
Cycle4
1. Use PC(B), GHR(A) and PATH(A) to calculate the indexes for the bimodal table and gtables.
2. Use PC(B) to read the local history and calculate the index of ltable.
1. Use PC(C), GHR(B) to calculate tags.
2. Read 4 adjacent entries from each prediction table. Tag comparisons may also be finished in this cycle.
1. Calculate 4 potential predictions.
2. Use the outcomes of B and C to select out one prediction.

Figure 6: A 3-block ahead scheme for PMPM branch predictors. Four basic blocks ending with branches A, B, C and D are fetched in cycle1 to cycle4. Associated operations in each cycle are listed in the figure.

It has been shown in [20] that speculative updates of branch history are important for prediction accuracy and an outstanding branch queue (OBQ) is proposed to manage speculative updates of both global and local branch histories. With ahead pipelining, the global history can be managed in the same way as proposed in [20]. However, for updating local histories, the indexes of the local history table and the OBQ need to use the n-block ahead branch address instead of the current branch address. In this paper, we assume that an OBQ is available to manage speculative updates for local branch history. Considering the extra hardware complexity of the OBQ, we also report the results of PMPM predictors without local histories in Sections 5.

5. Performance Evaluation of the Realistic PMPM predictor

In this section, we study the performance and explore design trade-offs of realistic PMPM predictors.

5.1. Prediction Accuracy

As the PMPM predictor is built upon the TAGE predictor, we compare their prediction accuracies. In this experiment, we simulated two 32kB PMPM predictors, a PMPM predictor with both global histories and local histories (labeled as "*PMPM-GL*") and a PMPM predictor with only global histories (labeled as "*PMPM-G*"), and one 32kB TAGE predictor.

The detailed configurations of the predictors are shown in Table 4. The configuration for the TAGE predictor is scaled from what presented in [18]. In stead of tuning the PMPM predictors, we use the same global history input and similar table structures as the TAGE predictor for a fair comparison. Compared to the TAGE predictor, the PMPM predictors have larger prediction counters but smaller tags. A larger prediction counter is necessary for the PMPM predictors to suppress noises in the summation process. In order to save some space for local history related tables, the PMPM-GL predictor has a smaller bimodal table and smaller tags for three gtables compared to the PMPM-G predictor. All bimodal tables in Table 4 have the same number of prediction bits and hysteresis bits.

PMPM-GL	History	10 bits local history; Geometrical global histories from 5 to 131; 16 bits path history.
	Table sizes	bimodal table: 8k entries; gtables: 2k entries; ltable: 1k entries; LHR table: 1k entries.
	Counter widths	Prediction ctr: 5 bits; ubit counter: 2 bits.
	Tag widths	gtable4 to gtable6: 8 bits; gtable0 to gtable3: 9 bits; ltable: 5 bits.
PMPM-G	History	Geometrical global histories from 5 to 131; 16 bits path history.
	Table sizes	bimodal table: 16k entries; gtables: 2k entries.
	Counter widths	Prediction ctr: 5 bits; ubit counter: 2 bits.
	Tag widths	9 bits.
TAGE	History	Geometrical global histories from 5 to 131; 16 bits path history.
	Table sizes	bimodal table: 16k entries; gtables: 2k entries.
	Counter widths	Prediction ctr: 3 bits; ubit counter: 2 bits.
	Tag widths	11 bits

Table 4: Configurations of the PMPM-GL, PMPM-G and TAGE predictors with a 32kB storage budget.

Table 5 shows the misprediction rates of those three predictors. Compared to the TAGE predictor, the average misprediction reductions of 6% and 2% are achieved by the PMPM-GL and the PMPM-G predictors, respectively. The PMPM-GL predictor achieves higher prediction accuracy than the TAGE predictor for all traces except *gcc*. Since the PMPM predictor has smaller tag widths, it is more sensitive to aliasing than the TAGE predictor. The *gcc* trace has a large number of static branches, which result in many aliases and degrade the performance of the PMPM predictor.

Trace	PMPM-GL	PMPM-G	TAGE	Trace	PMPM-GL	PMPM-G	TAGE
gzip	9.685	10.300	10.899	*vpr*	8.926	9.003	9.361
gcc	3.826	3.794	3.536	*mcf*	10.182	10.128	10.254
crafty	2.555	2.558	2.682	*parser*	5.332	5.437	5.422
compress	5.571	5.827	5.934	*jess*	0.413	0.464	0.456
raytrace	0.652	1.112	1.099	*db*	2.343	2.408	2.527
Javac	1.105	1.144	1.160	*mpegaudio*	1.093	1.137	1.163
mtrt	0.734	1.188	1.139	*jack*	0.724	0.845	0.831
eon	0.305	0.470	0.487	*perlbmk*	0.319	0.497	0.480
gap	1.436	1.776	1.783	*vortex*	0.141	0.336	0.312
bzip2	0.037	0.043	0.041	*twolf*	13.447	13.466	13.758
Average		PMPM-GL: 3.441		PMPM-G: 3.597		TAGE: 3.666	

Table 5: Misprediction rates (MPKI) of the PMPM-GL, PMPM-G and TAGE predictors with a 32kB storage budget.

5.2. Design Space Exploration

Similar to recently proposed multi-table branch predictors [10], [12], [18], the PMPM predictor has a large design space. In this section, we examine the impacts of different design parameters of PMPM predictors, including the maximum global history length, the prediction counter size, and the number of global prediction tables. For each of those parameters, we also vary the table sizes to show the trend under different storage constraints. Due to their limited impact on prediction accuracy, we keep intact the local history length and the path history length. For those parameters, the same configurations as presented in Table 4 for the PMPM-GL predictor are used. In addition, we only use one local prediction table and fix the *ubit* counter width as 2. The minimum global history length (i.e., the number of global history bits used in the gtable corresponding to the shortest global history) is set to 4, which is optimal in our experiments.

For the ease of the discussion, we parameterize the configurations of PMPM-GL predictors in the following manner. We assume that the local prediction table has 2^N entries. The sizes of other tables are then scaled based on that of the local prediction table, as shown in Table 6. The total number of bits of the prediction counter and the tag in each entry of a prediction table is also specified in Table 6 based on our empirical tuning results. If we use seven gtables, 5-bit prediction counters, and N set as 10, the resulting configuration would be exactly the same as the PMPM-GL predictor in Table 4. When we change the number of gtables to K, up to $\lceil K/2 \rceil$ prediction counters will be summed since the prediction policy is designed to select one counter out of each group of two gtables.

Table	Ltable	gtable (0 to 3)	Other gtables	Bimodal table	LHR table
Entries	2^N	2^{N+1}	2^{N+1}	2^N	2^N
pred ctr + tag	10 bits	16 bits	15 bits		

Table 6: Parameterized configuration of PMPM predictors.

We first study the impact of the maximum global history length on PMPM predictors. With seven gtables and 5-bit prediction counters, we vary the maximum global history lengths from 50 to 400 for PMPM-GL predictors. For each history length, we also change the sizes of prediction tables by varying N from 9 to 12, corresponding to the storage budget ranging from 128 kbits to 1024 k btis. The average misprediction rates of those predictors are reported in Figure 7. As shown in the figure, the misprediction rates typically decrease when we increase the maximum history length. The reduction, however, is very limited when the maximum history length is beyond 150. For example, among all the different history lengths that we examined from 150 to 400 for the 256 kbits (N = 10) PMPM predictor, the minimum misprediction rate is 3.43 MPKI and the maximum misprediction rate is 3.45 MPKI. Such robustness makes the predictor insensitive to the selection of the maximum global history length and this observation is consistent with the findings from the O-GEHL and TAGE predictors [15], [18]. For remaining experiments presented in this subsection, we use 160 as the maximum history length.

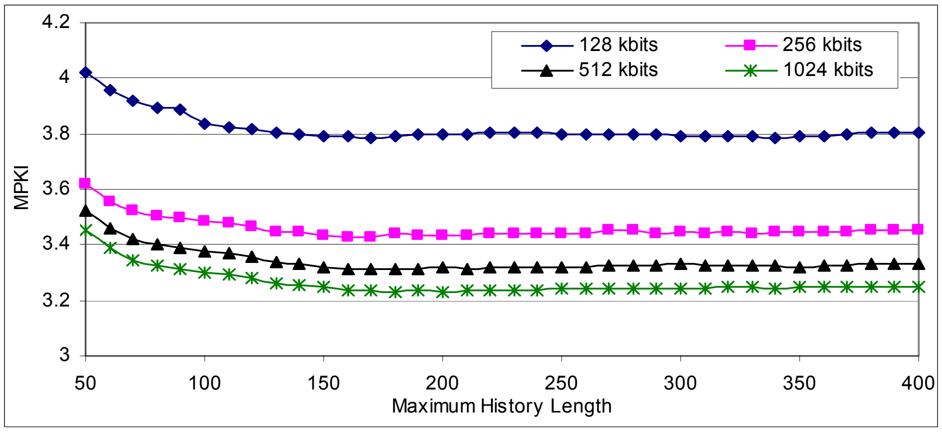

Figure 7: Misprediction rates of the PMPM predictors with different maximum global history lengths.

Second, we determine the prediction counter size of PMPM-GL predictors. With a limited storage budget, the prediction counters compete with the tag bits for resource. As a result, if we use large prediction counters to account for hysteresis, the reduced tag bits may result in increased many aliases due to partial tag matching. In order to find the optimal tradeoff between the prediction counter sizes and tag widths, we simulate PMPM-GL predictors by varying the size of the prediction counters from 2 to 8 bits. The maximum history length is set as 160 as determined from the previous experiment. The misprediction rates of those PMPM-GL predictors are shown in Figure 8. It can be seen that small prediction counters are very sensitive to noise even for large tables, resulting in high misprediction rates. Limited tag widths as a result of large prediction counters expose the problem of aliasing, especially for small prediction tables. Among different prediction counter sizes, the best tradeoff is achieved when we use 5-bit prediction counters. Therefore, in remaining experiments, we keep the prediction counter size as 5 bits.

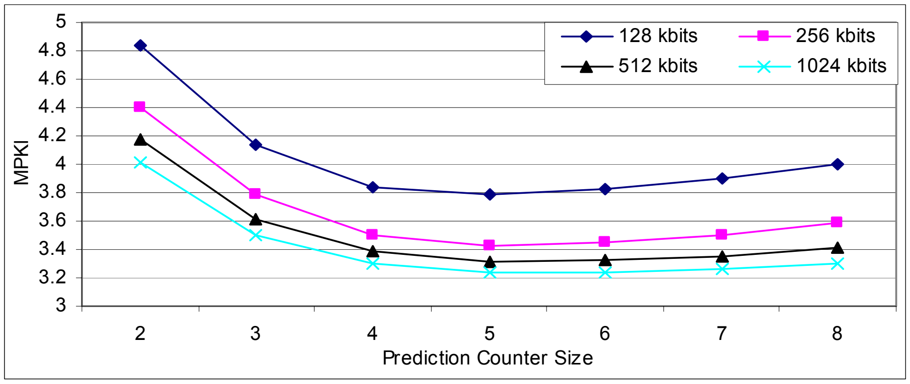

Figure 8: Misprediction rates of the PMPM predictors with different prediction counter sizes.

Next, we study the impact of different number of global prediction tables in PMPM-GL predictors. With more prediction tables, more partial matches can be available to be combined, which potentially may improve prediction accuracy. On the other hand, extra prediction tables will increase the complexity of the predictor since more prediction counters need to be summed to produce the final prediction. Table 7 presents the misprediction rates of PMPM-GL predictors with 4 to 9 gtables under different storage budgets. It can be seen from the table that more gtables show higher prediction accuracy by integrating more prediction counters. For example, the 256 kbits (N = 10) PMPM predictor with 7 gtables has higher accuracy than the 664 kbits (N = 12) PMPM predictor with 4 gtables (3.427 MPKI vs. 3.451 MPKI). Among different number of gtables, seven gtables provides a good tradeoff between cost and complexity since further increasing the number of gtables only results in marginal returns.

N	Number of gtables	4	5	6	7	8	9
9	Predictor size (kbits)	83	98	113	128	143	158
	AVG MPKI	4.201	3.970	3.869	3.789	3.760	3.714
10	Predictor size (kbits)	166	196	226	256	286	316
	AVG MPKI	3.741	3.562	3.497	3.427	3.413	3.379
11	Predictor size (kbits)	332	392	452	512	572	632
	AVG MPKI	3.590	3.416	3.374	3.311	3.321	3.289
12	Predictor size (kbits)	664	784	904	1024	1144	1264
	AVG MPKI	3.451	3.311	3.282	3.238	3.239	3.213

Table 7: Misprediction rates of the PMPM predictors with different numbers of gtables and predictor sizes.

5.3. Impact of Fully Biased Branches in the Realistic PMPM Predictor

In many applications, a large number of branches have highly biased outcomes (i.e., always taken or always not-taken). A simple bias predictor was proposed in [3] to filter out those branches in order to reduce the aliasing impact on main prediction tables. To study the impact of highly biased branches on PMPM-GL predictors, we augment them with a bias predictor as proposed in [3]. The experimental results show that a small bias predictor with 1k entries provides no benefit to the 256 kbits PMPM-GL predictor. A large bias predictor with 64k entries only reduces the average misprediction rate by 0.006 MPKI. The reason is that the bimodal prediction table in PMPM-GL predictors already filters out most of those highly biased branches.

5.4. Impact of Loop Branches in the Realistic PMPM Predictor

We also tried to combine PMPM predictor with a low cost loop predictor [19], [3], [12] to examine how accurate loop branches are predicted. The loop predictor has 256 entries and consumes 13 kbits storage budget. The experimental results show that the extra loop predictor reduces the misprediction rate of the 256 kbits PMPM-G predictor without using local histories by 0.058 MPKI. The reduction of misprediction rate of the PMPM-GL predictor with both global and local histories is 0.01 MPKI. Those results are consistent with the discussion of using the loop predictor for idealistic PMPM predictors in Section 3.6.

5.5. The Realistic PMPM Predictor for CBP2

Based on our design space exploration, we chose to use the following configuration for a 256 kbits PMPM predictor: minimum global history length as 4, seven gtables, and 5-bit prediction counters. In order to further improve the prediction accuracy of PMPM predictors, we empirically fine tuned the configurations and used several optimizations for our realistic PMPM predictor for CBP2. The optimizations are listed as follows:

1) In the bimodal prediction table, four prediction bits will share one hysteresis bit as proposed in [17].
2) We use the number of branches that miss in all global prediction tables as a metric to detect traces with large branch footprints. For those traces, we periodically reset the *ubits* as used in the TAGE predictor.
3) If the predictions from global prediction tables are same, we will not update the *ubits*.

Considering the extra hardware to support local history management, we submitted two versions of the PMPM predictor to CBP2. One predictor (labeled as "PMPM-CBP2-GL") uses both global and local histories. The other (labeled as "PMPM-CBP2-L") only uses global history. The detailed configurations and hardware costs of those two predictors are shown in Table 8. The prediction accuracies of these PMPM predictors are shown in Table 9. From the table, we can see that the local history is still important for some traces (e.g., *raytrace, mtrt and vortex*) although we already use a very long global history.

With the ahead pipelining scheme described in Section 4.5, we simulated the proposed PMPM predictions with ideal 1-block ahead and 2 to 4-block ahead pipelining schemes. The prediction accuracies of them are shown in Figure 9. From the figure, it can be seen that with 3-block ahead pipelining, the average accuracy loss is about 0.11 MPKI for the PMPM-CBP2-G predictor compared to ideal 1-block ahead pipelining. The accuracy loss for the PMPM-CBP2-GL predictor is about 0.15 MPKI. As discussed in Section 4.5, with ahead pipelining, the local history table is indexed by the n-block ahead branch address, which will introduce extra aliasing to the local history table and increase the accuracy loss of the PMPM-CBP2-GL predictor.

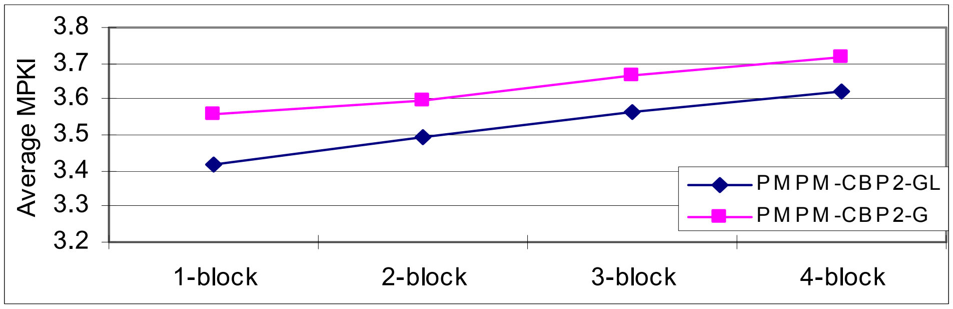

Figure 9: Misprediction rates of the ahead pipelined PMPM predictors for CBP2.

	PMPM-CBP2-GL	PMPM-CBP2-G
Each prediction counter	5 (bits)	5 (bits)
Each ubit	2 (bits)	2 (bits)
History lengths for gtable 6 to gtable 0	4, 7, 15, 25, 48, 78, 203	4, 7, 15, 25, 48, 78, 203
Number of entries for each gtable	2k	2k
Tag widths for gtable 6 to gtable 0	6, 7, 7, 8, 9, 10, 10 (bits)	7, 8, 8, 10, 11, 12, 12 (bits)
Total cost of gtables	217088 (bits)	239616 (bits)
Local prediction table	1k entries; 5 bits tags	
Cost of the ltable	12288 (bits)	
Local history table	1k entries; His Len: 11	
Cost of the local history table	11264 (bits)	
Cost of the bimodal table	20480 (bits)	20480 (bits)
Global history register	203 (bits)	203 (bits)
Path history register	16 (bits)	16 (bits)
Cyclic shift registers for gtable tags and indexes	184 (bits)	206 (bits)
Cost of adaptation counters and flags	71 (bits)	71 (bits)
Total Hardware Cost	261594 (bits)	260592 (bits)

Table 8: Predictor Configurations and Hardware Costs

Trace	CBP2-GL	CBP2-G	Trace	CBP2-GL	CBP2-G
gzip	9.712	10.346	*vpr*	8.945	9.063
gcc	3.690	3.637	*mcf*	10.092	10.033
crafty	2.581	2.565	*parser*	5.215	5.244
compress	5.537	5.819	*jess*	0.393	0.433
raytrace	0.542	0.963	*db*	2.319	2.380
javac	1.107	1.159	*mpegaudio*	1.102	1.159
mtrt	0.657	1.009	*jack*	0.688	0.763
eon	0.276	0.359	*perlbmk*	0.314	0.484
gap	1.431	1.745	*vortex*	0.137	0.331
bzip2	0.037	0.042	*twolf*	13.551	13.616
Average		PMPM-CBP2-GL: 3.416		PMPM-CBP2-G: 3.557	

Table 9: Misprediction rates (MPKI) of the PMPM-CBP2-GL and PMPM-CBP2-G predictors

6. Conclusions

In this paper, we show that the PPM algorithm with the longest match may lead to suboptimal results. By combining multiple partial matches, the proposed PMPM algorithm can better adapt to various history length requirements. An idealistic implementation of the PMPM predictor is presented in the paper to evaluate how high the prediction accuracy can be achieved. We also proposed a realistic design based on the PMPM algorithm. Our results show that the realistic PMPM predictors can achieve higher accuracy than the TAGE predictors with the same storage cost.

References

[1] I.-C. Chen, J. Coffey, and T. Mudge, "Analysis of branch prediction via data compression," in *proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)* , Oct. 1996.

[2] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications (TC), vol. 32, pp. 396-402*, Apr. 1984.

[3] 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.

[4] H. Gao and H. Zhou, "Branch Prediction by Combining Multiple Partial Matches," *Technical report, School of EECS, Univ. of Central Florida*, Oct. 2006.

[5] H. Gao and H. Zhou, "PMPM: Prediction by Combining Multiple Partial Matches," In *the 2nd JILP Championship Branch Prediction Competition (CBP-2)*, 2006.

[6] D. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," In *proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, 2001.

[7] D. Jiménez and C. Lin, "Neural methods for dynamic branch prediction," *ACM Trans. on Computer Systems*, 2002.

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

[9] 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 (ISCA-25)*, June 1998.

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

[11] P. Michaud and A. Seznec, "A comprehensive study of dynamic global-history branch prediction," *Research report PI-1406, IRISA*, June 2001.

[12] A. Seznec, "A 256 Kbits L-TAGE Branch Predictor," In *the 2nd JILP Championship Branch Prediction Competition (CBP-2)*, 2006.

[13] A. Seznec, "Looking for Limits in Branch Prediction with the GTL Predictor," In *the 2nd JILP Championship Branch Prediction Competition (CBP-2)*, 2006.

[14] A. Seznec, "The O-GEHL branch predictor," In *the 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[15] A. Seznec, "Analysis of the o-gehl branch predictor," In *proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-32)*, June 2005.

[16] A. Seznec and A. Fraboulet, "Effective ahead pipelining of the instruction address generator," In *proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30)*, June 2003.

[17] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeidés, "Design tradeoffs for the ev8 branch predictor," In *proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA-29)*, 2002.

[18] A. Seznec, P. Michaud, "A case for (partially) tagged Geometric History Length Branch Prediction," *Journal of Instruction Level Parallelism (JILP), vol. 8,* February 2006.

[19] J. Shen and M. Lipasti, "Modern Processor Design Fundamentals of Superscalar Processors," Mc Graw Hill, 2005.

[20] K. Skadron, M. Martonosi, and D. Clark, "Speculative updates of local and global branch history: A quantitative analysis," *Journal of Instruction Level Parallelism (JILP), vol. 2,* January 2000.

[21] J. E. Smith, "A study of branch prediction strategies," In *proceedings of the 8th Annual International Symposium on Computer Architecture (ISCA-8)*, 1981.