PMPM: Prediction by Combining Multiple Partial Matches

Hongliang Gao Huiyang Zhou

School of Electrical Engineering and Computer Science
University of Central Florida
{hgao, zhou}@cs.ucf.edu

***Abstract*—The PPM prediction 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],[9], the prediction by partial matching (PPM) algorithm [2], originally proposed for text compression, can achieve very high prediction accuracy for conditional branches. Recent studies of PPM-like predictors [13], [8] confirm the effectiveness of PPM and show that PPM-like predictors outperform many state-of-art branch predictors.

In this paper, we propose to improve PPM-like branch predictors by combining multiple partial matches rather than the longest partial match as used in the PPM algorithm. We first examine why longest partial match may lead to suboptimal prediction accuracy. We then propose a prediction algorithm to selectively combine multiple partial matches (PMPM) with an adder tree.

Based on the PMPM algorithm, we develop an idealistic predictor which explores extremely long history to push the limit on branch prediction accuracy and a realistic predictor for practical implementation. The realistic design is based on recently developed PPM-like TAGE branch predictors [13] 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 realistic PMPM predictors. Finally, Section 5 concludes the paper.

2. PREDICTION BY COMBINING MULTIPLE PARTIAL MATCHES

2.1. PPM with the (Confident) Longest Match

In the PPM algorithm, the Markov model, which determines the prediction, is updated based on input information. When used for branch prediction, the Markov model keeps track of branch history and uses the longest match to make a prediction. The assumption behind the longest match is that the longer history provides a more accurate context to determine the 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, in which the global branch history is used as the context for each branch. We assign a signed saturating prediction counter with the range [-4, 4] for each (branch address, history) pair. 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 used.

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 make a 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 confident longest match. We simulate both schemes, i.e., PPM with the longest match and PPM with the confident longest match, and report the misprediction rate reductions, measured in mispredictions per 1000 instructions (MPKI), achieved by the confident longest match. The results are shown in Figure 1. In this experiment, the maximum history length 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 benchmarks except the benchmark *vortex*, which implies that the longest match used in PPM may lead to suboptimal

prediction accuracy.

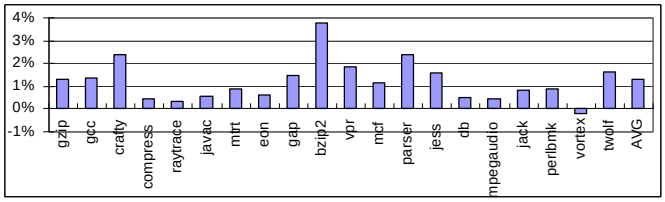

Figure 1. Misprediction rate reductions by the confident longest match in PPM-based branch predictors (Max History Length = 40).

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 [7] and explored in recent works, such as [10]. 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 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.

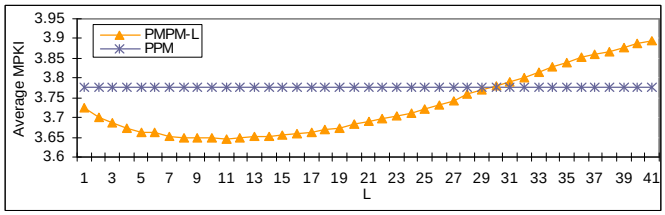

Figure 2. Misprediction rates of PMPM-L predictors and the original PPM predictor

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. 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 this competition, 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 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, 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 branch predictor to filter them out. The simple 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 remaining branches, PBTT sends the branch tag, LHR, meta counter, and bias counter to the PMPM predictor.

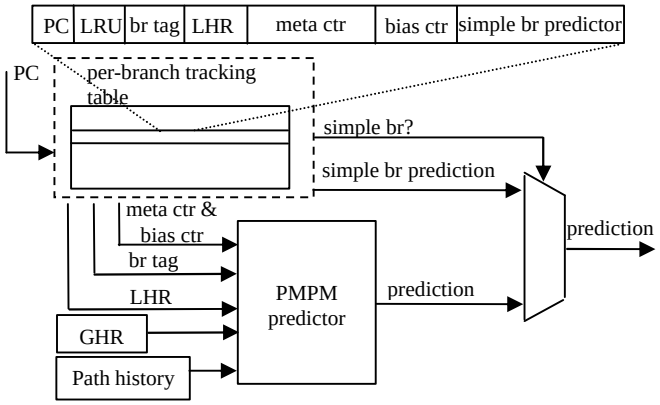

Figure 3. The overall idealistic PMPM branch prediction scheme.

The PMPM predictor has three prediction tables, as shown in Figure 4. 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, a usefulness counter, and a benefit counter. In order to capture very long global histories, we use geometric history lengths [10] in the long GHR table.

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 [13] and [6] with empirically selected prime numbers. The indexes and tags use different primary numbers in their hash functions.

We use the short GHR table to store global histories with lengths 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 history. *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., the branch history has not been retained in the tables; new entries are allocated in the following way:

- For the short-GHR table, we assign new entries when the meta counter indicates that LHR prediction table is not sufficient to make correct predictions.
- For the long-GHR table, we assign new entries when the overall prediction is wrong.
- For the LHR table, we always assign new entries.

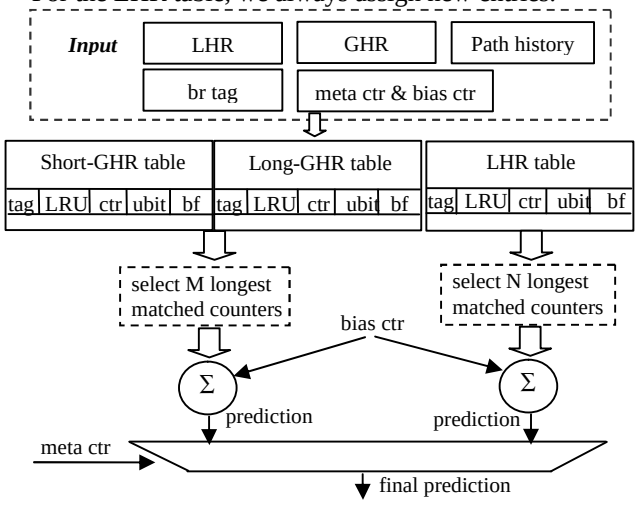

Figure 4. An optimized idealistic PMPM predictor.

3.4. Optimizations

In order to further improve the accuracy of the proposed 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 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 (*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 is not included, the *ubit_pred* would change from correct to wrong. Then, we increase the corresponding benefit counter.

If the prediction counter is not included, the *ubit_pred* would change from wrong to correct. Then, we decrease the corresponding benefit counter.

3.5. Results

We simulate the proposed idealistic PMPM predictor with the configuration shown in Table 1.

Table 1. Idealistic Predictor Configuration.

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]

With the configuration shown in Table 1, the simulator consumes around 226M bytes of memory and takes 1.65 hours to finish all the benchmarks on a Xeron 2.8Ghz computer. The final misprediction rates (measured in MPKI) for the distributed traces of CBP2 are shown in Table 2. For comparison, Table 2 also includes the misprediction rates of the PPM predictor, i.e., in the idealistic predictor presented in Figure 4, 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 *vortex*. The misprediction rates reduction is up to 20.9% (*gzip*) and 15.2% on average.

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

Trace	PPM	PMPM	Trace	PPM	PMPM
Gzip	11.977	9.470	vpr	10.096	8.273
Gcc	2.748	2.594	mcf	9.258	7.688
crafty	2.083	1.794	parser	4.404	3.928
compress	6.526	5.167	jess	0.302	0.290
raytrace	0.274	0.272	db	2.721	2.199
javac	1.054	0.930	mpegaudio	1.051	0.922
mtrt	0.326	0.318	jack	0.484	0.472
eon	0.246	0.239	perlbmk	0.177	0.192
gap	1.164	1.081	vortex	0.090	0.087
bzip2	0.036	0.032	twolf	11.591	10.529
Average	PPM: 3.330		PMPM: 2.824		

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 [8], [13], the TAGE branch predictor [13] 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.

As shown in Figure 5, the PMPM predictor contains a

bimodal prediction table [15], 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 [10] 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 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.

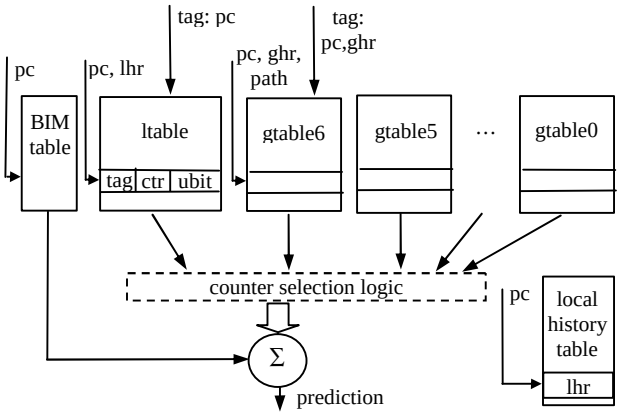

Figure 5. The practical PMPM predictor.

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 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 [4], [5], and the O-GEHL predictor [10], 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 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 also 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 doesn't 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. 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. 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 if its prediction counter is wrong. In this way, we limit the chance of useful entries to be victimized.

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 [11] 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 proposed PMPM predictors will be examined in Section 4.7

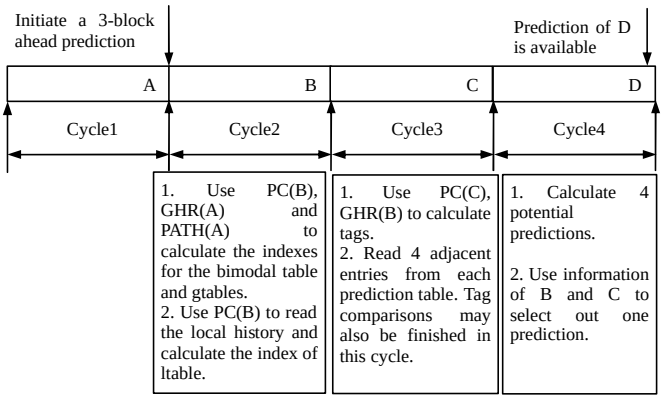

Figure 6. A 3-block ahead scheme for the PMPM predictor. Four basic blocks ending with branches A, B, C and D are fetched in cycle1 to cycle4.

It has been shown in [14] 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 [14]. However, for the update of local history, 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 the 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 using local histories in Sections 4.6 and 4.7.

4.6. Prediction Accuracy

As the PMPM predictor is built upon the TAGE predictor, we compare their prediction accuracies. We simulated two 32kB PMPM predictors, a PMPM predictor with both global history and local history (labeled as "PMPM-GL") and a PMPM predictor with only global history (labeled as "PMPM-G"), and one 32kB TAGE predictor using the base configuration presented in [13]. The configurations are shown in Table 3. Those three predictors have the same global history input and similar table structures. Compared to the TAGE predictor, the PMPM-G predictor has 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. To simplify the configurations, all bimodal tables in Table 3 have the same number of prediction bits and hysteresis bits. Table 4 shows the misprediction rates of these three predictors. Compared to the TAGE predictor, the average misprediction reductions are 6% and 2% respectively achieved by the PMPM-GL and the PMPM-G predictors.

Table 3. Configurations of the PMPM-GL, PMPM-G and TAGE predictors with 32kB storage budget.

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 btis

Table 4. Misprediction rates (MPKI) of the PMPM-GL, PMPM-G and TAGE predictors with 32kB storage budget.

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		

4.7. Realistic PMPM predictors for CBP2

In order to further improve the prediction accuracy of the PMPM predictors, we empirically tuned the configurations and used several optimizations for our realistic PMPM predictors 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 [12].
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 limit the update of *ubits*.

Considering the extra hardware to support local history management, we submitted two versions of the PMPM predictor for 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 configurations and hardware costs of the predictors are shown in Table 5. The prediction accuracies of these PMPM predictors are shown in Table 6. From the table, we can see that the local history is still important for some benchmarks (e.g., *raytrace, mtrt and vortex*) although we already use a very long global history.

Table 5. Predictor Configurations and Hardware Costs

	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 6. Misprediction rates (MPKI) of the PMPM-CBP2-GL and PMPM-CBP2-G predictors

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		

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 7. As shown in the figure, with the 3-block ahead pipelining, the average accuracy loss is less than 0.16 MPKI compared to 1-block ahead pipelining.

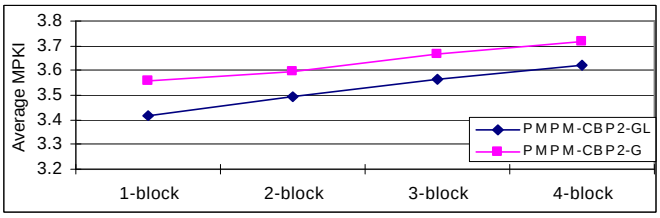

Figure 7. Misprediction rates of the ahead pipelined PMPM predictors for CBP2.

5. 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 and it is used 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*, Oct. 1996.

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

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

[4] 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.

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

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

[7] 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 1998.

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

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

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

[11] 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.

[12] 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*, 2002.

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

[14] 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*, January 2000.

[15] J. E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, 1981.