Global-Local Combined Branch History: The Alternative Way to Improve TAGE Branch Predictor

Yasuo Ishii
NEC Corporation
y-ishii@bc.jp.nec.com

ABSTRACT

In this paper, we propose global-local combined branch history. On the combined branch history, global branch histories and local branch histories are mixed with fixed ratio. The branch predictor can capture the behavior of the branch pattern from both global history and local history by using combined branch history. We apply it for the state-of-the-art tagged predictor. The optimized branch predictor using combined branch history achieves 2.401 MPKI with a 32KB budget and 3.629 MPKI with a 4KB budget.

1. INTRODUCTION

Modern branch predictor uses branch history to predict the future branch behaviors. Generally, the branch history can be classified into two types, global histories and local histories. The global histories are widely used for state-of-the-art branch predictors [6,7,8]. The local histories are used to complement the prediction results generated from the global history because the local histories are good to capture some control structure such as loop structure.

Recently, advanced branch predictors employ multiple prediction components to improve the prediction accuracy. These prediction components can be classified into two types, global components and local components. The global components generate their prediction result from global histories and the local components generate their prediction result from local histories. Typically, modern advanced branch predictors that use both global histories and local histories have to fuse prediction results from local components and global components by adder trees [9]. This approach is widely used in previous branch prediction championship and improves performance [1,2,8,9]. However, this reduction approach cannot apply for PPM-like partially tagged branch predictors [6] because its prediction algorithm does not use the adder tree to compute final prediction result from multiple prediction components.

On partially tagged branch predictors, each prediction component uses different branch history length and puts high priority to the prediction components which use long histories. Based on this assumption, the partially tagged branch predictors compute the final prediction result. For such predictors, there is no effective ways to select the final prediction from local components and global components.

In this paper, we propose the global-local combined branch history which is new branch history representation. On this combined branch history, one prediction component uses both global histories and local histories to make prediction results. This history can be applied for the partially tagged branch predictors. We apply the global-local combined branch history for the LSC-TAGE branch predictor [8] which is the state-of-the-art branch predictor and the successor of the winner of the previous branch prediction championship. We also apply some optimizations for LSC-TAGE branch predictor to improve the performance. We call our submitted branch predictor as the global-local combined TAGE (GL-TAGE) in this paper.

This paper is organized as follows. We describe the global-local combined branch history in Section 2. Our main contribution GL-TAGE is proposed in Section 3. The other optimization techniques are explained in Section 4. We describe the submitted predictor in Section 5. We conclude this paper in Section 6.

2. COMBINED BRANCH HISTORY

In this paper, we combine local histories and global histories to enhance the accuracy of existing tagged branch predictors. As far as we know, there is no effective way to combine local branch histories for tagged predictors.

On global history, all branch outcomes are pushed into one branch history storage. On the other hand, the branch outcomes are categorized into multiple groups in local history. Generally, PC is used for the classification. The local history table (LHT) is used for the storage of local history. When the number of entries of LHT is N_{LHT}, LHT holds N_{LHT} dedicated branch history series.

Figure 1 shows the relationship between global history and local history. The character in each box represents the corresponding PC, and the number represents the ordering. On global branch history, the youngest N_G branch outcomes are always used for the prediction as shown in Figure 1(a). On the local branch history, youngest N_L branch histories from one group used for the prediction as shown in Figure 1(b). In Figure 1, $N_G = 6$ and $N_L = 3$ is used.

As shown in the figure, the information of the local history (C0,C1,…) appeared on the global history series when the global history length N_G is enough long. On the other hand, the prediction which uses local histories easily captures the loop structure even if the history length N_L is not so long. On the combined branch history, N_G global branch history and N_L local branch history are mixed by fixed ratio. Figure 1(c) shows the $N_G = 6$ and $N_L = 2$ combined history.

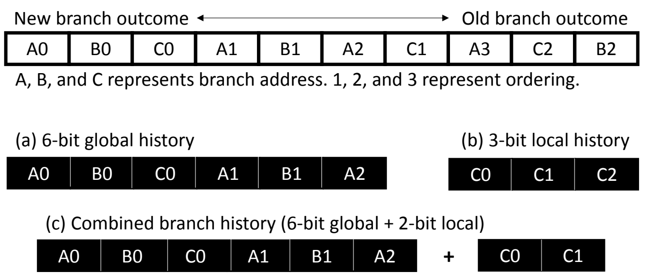
New branch outcome
Old branch outcome
A0 B0 C0 A1 B1 A2 C1 A3 C2 B2
A, B, and C represents branch address. 1, 2, and 3 represent ordering.
(a) 6-bit global history
A0 B0 C0 A1 B1 A2
(b) 3-bit local history
C0 C1 C2
(c) Combined branch history (6-bit global + 2-bit local)
A0 B0 C0 A1 B1 A2 + C0 C1

Figure 1. Combined Branch History

We found that $N_L = N_G / N_{LHT}$ is good balance to mix these two different branch histories because the length of local history become $1/N_{LHT}$ of the global history length when PC of all branches are distributed uniformly. In Figure 1, three branch groups (A, B, and C) are appeared. Therefore, the frequency of branch C (3 times) is about 1/3 of all branch outcomes (10 branches).

3. GL-TAGE BRANCH PREDICTOR

Algorithm of GL-TAGE is derived from that of LSC-TAGE [8] whose prediction algorithm is derived from TAGE [6].

3.1 Predictor Configuration

Figure 2 shows the overview of the GL-TAGE predictor. It employs multiple prediction components. The first table is the base component. It provides prediction from only corresponding program counters (PC). The prediction of the base component is overwritten by the prediction result from the tagged components T_i (i = {0,…,N-1}). Each entry of tagged component has three fields which are a 3-bit signed saturation counter, partially tag, and a 2-bit usefulness counter. The signed saturation counter represents the bias of the corresponding branch history pattern. The negative value indicates not-taken. Otherwise, the counter indicates taken. The usefulness counter is used for the replacement information of each entry.

The difference of each tagged components is corresponding branch history length. We assume that Ti corresponds to longer branch history than that of T_j (j = {0, …, i-1}). In this paper, branch history length L(i) for each T_i forms a geometric series [4,6]. Each tagged component has own hash functions to compute the index and the partial tag. The difference from LSC-TAGE branch predictor is the use of hash function for combine branch history. We combine local histories for the index computation of prediction components T_k (k = {N/2,…,N-1}).

3.2 Prediction Algorithm

On the prediction phase, index and tag for all components are computed simultaneously. The base component always provides prediction result. Each tagged component provides its prediction result only when the tag held in each table matches the computed tag. Generally, the prediction result from the tagged component that uses the longest history length is selected as the final prediction result. When there

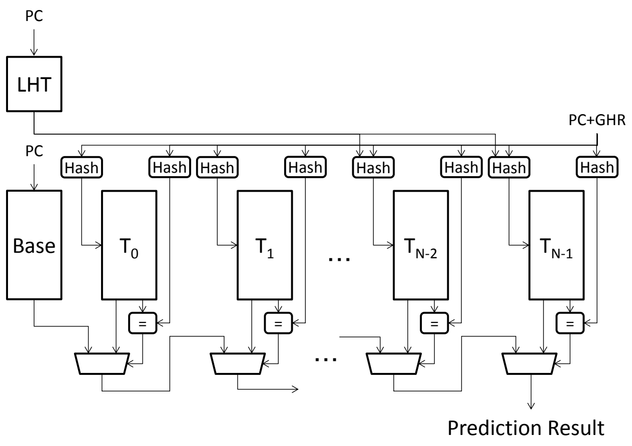

Figure 2. GL-TAGE branch predictor

is no hit tagged component, the prediction result from the base component is used as the final prediction result.

However, TAGE select the prediction from second longest match component for the final prediction result when its prediction result is more accurate than that of the longest match component. The prediction of the second longest match component is used when the following conditions are met: (1) the saturation counter of the longest match component is weakly biased (the counter value is 0 or -1) and (2) dynamic monitoring (UA) counter is negative which indicates that the second longest match tagged component seems to be more accurate than that of the longest match component. The UA counter tracks the usefulness of the second longest match tagged component. Unlike existing TAGE branch predictor, the predictor employs multiple UA counters to realize precise tracking. The detail update algorithm of UA is described in following section.

3.3 Updating Algorithm

On the update phase, the entry of longest match component is updated. The saturation counter is incremented when the branch outcome is taken, otherwise decremented. The useful counter is set to the maximum value when the following two conditions are met: (1) the longest match component provided the correct prediction and (2) the second longest match component provided the wrong prediction. The update strategy of the useful counter is different from existing TAGE predictor. This strategy is derived from Re-Reference Interval Prediction [11]. The saturation counter of the second longest match component is also update when the usefulness counter of the longest match component shows the minimum value.

When the prediction result is wrong, the new entry is allocated. When the longest match entry is Ti, the predictor checks entries of T_j from j=i+1 to j=N-1. When the usefulness counter of T_j is zero, the corresponding entry is allocated. This allocation process is performed five times. After N^{th} allocation, we skip 1+(N/2) tables to check its usefulness counters. On the allocation process, the partially tag is set to the corresponding hash , the usefulness counter is set to zero, and the saturation counter is set to weakly taken (0) when the branch outcome is taken. Otherwise, the saturation counter is set to weakly not-taken (-1).

As shown in this section, the usefulness counters are increased monotonically for usual updating scenario. It can prevent the allocation of new prediction entries. Therefore, all usefulness counters are decremented when the new allocation of the entry seems to be hard. To track the difficulty of the allocation, TICK counter is employed. TICK counter increments when the allocation is succeeded, and decrements when the allocation is not succeeded because the useful counter of corresponding entry is not zero. When the TICK counter is saturated, all usefulness counters are decremented and the TICK counter is reset to the minimum value. We also decrement these counter when the following two conditions are met: (1) no entries are allocated and (2) the accuracy of the TAGE branch predictor is less than threshold. We implement difficulty counter (DC) to track the accuracy of the TAGE predictor.

4. OPTIMIZATIONS

We describe the other prediction techniques that uses in the submitted predictor.

4.1 Statistical Corrector Predictor

The branch prediction result from TAGE branch predictor is generally accurate. However, when the branch outcome is not correlated with the previous branch history pattern, the TAGE branch predictor cannot provide accurate branch prediction. To cover such situation, the statistical corrector predictor was proposed in previous championship [7].

Statistical Corrector Predictor forms the neural branch predictor which composed of multiple components. Each component is indexed by either global history or local history. Each component provides its prediction by the small integer value. The counter value is also used for the input of the adder tree. The summation value of these small integer values represents the bias of the corresponding branch. The sign of the summation represents the direction of the prediction result and the absolute value of the summation represents the confidence level [3]. When the confidence level is higher than a threshold, the statistical corrector predictor overwrites the prediction result of the TAGE predictor. The threshold is adaptively changed as performed in the original statistical corrector predictor [7].

We use both global history and local history as shown in FTL predictors [1,2]. The same configuration is also described in LSC-TAGE predictor [8].

4.2 Loop Predictor

We implement the loop predictor to predict constant loops. The configuration is derived from ISL-TAGE [7]. Each entry of the loop predictor has a direction bit of the loop, a 10-bit tag of the corresponding branch address, a 10-bit current iteration counter, a 10-bit loop length counter, a 3-bit confidence counter, and a 3-bit age counter. Each entry tracks the loop behavior of the corresponding branch. For each loop structure is appeared in the application, the confidence counter is incremented. When the confidence counter is saturated, the prediction result from loop predictor overwrites the final prediction. The predictor employs an additional 7-bit counter (WITHLOOP) which tracks the usefulness of the loop prediction. This counter is incremented when the loop predictor provides correct prediction result otherwise decremented. Only when the counter shows the positive value, the loop predictor worked.

4.3 Sharing Table

On TAGE predictor, each tagged component has own prediction table. However, it often wastes the storage. For effective use of the storage, the interleaving was proposed in the previous championship [7]. With this feature, multiple logical tagged components share same physical tables. We apply this feature on the submitted predictor.

4.4 Dedicated UA Counter

The second longest match prediction is used when UA counter show the second longest match prediction is more accurate than that of the longest match prediction when the longest match entry shows weakly biased. We provide dedicated UA counters for arbitrary pair of base predictor and four shared table groups for dedicated tracking.

4.5 Special History Treatment

The previous study shows the special treatment for function call is good to improve the prediction accuracy [2,7,10]. In our predictor, multiple branch histories are pushed into the global history register when the function call is appeared.

5. DESIGN OF SUBMITTED PREDICTOR

Table 1 shows the configuration of the 32KB GL-TAGE branch predictor. The statistical corrector predictor also uses both the global histories and local histories. We use {0, 0, 3, 4, 11} for the history series for global components and {1, 2, 5, 8, 12} for local components. Each entry has 6-bit signed integer. The loop predictor uses 48-entry. It uses skewed associative strategy as performed in ISL-TAGE. Table 2 shows the total budget count and configuration. Total budget count is 262782-bit which is less than 263168-bit. We also try to design GL-TAGE for 4KB configuration. We disabled the loop predictor and the difficulty counter (DC) on this configuration.

The parameters are shown in Table 3 and Table 4. We reduce the saturation counter by 1-bit. Therefore, each entry has 2-bit saturation counter.

Table 5 shows the accuracy of the submitted branch predictors. Our predictors achieve 2.401 MPKI and 3.629 MPKI respectively. This predictor slightly outperforms the accuracy of the predictor without the global-local combined history (2.410 MPKI for 32KB configuration).

Table 1. Configuration of GL-TAGE predictor (32KB)

	# of entry	Tag Width	Global History Length Local History Length	Budget (Kbit)
Base	16K	4 entries share 1 hysteresis bit		20
T0,T1,T2	Share	7-bit tag	0,4,8	Total
	2K	+ 5-bit	0,0,0	24
T3,T4,	Share	9-bit tag	11,15,19,26,34,44,58	Total
…,T9	8K	+ 5-bit	0,0,0,0,0,0,0	112
T10,T11,	Share	11-bit tag	76,100,131,172,226,296,389	Total
…,T16	4K	+ 5-bit	0,2,2,2,2,4,6	64
T17,T18,	Share	13-bit tag	511,670,880	Total
T19	0.5K	+ 5-bit	8,11,14	9
Total				**229**

Table 2. Budget Count for 32KB+1024bit Configuration

Resource	Configuration	Budget
GL-TAGE	See Table 1	229K bit
Statistic Corrector	4096 entries 6-bit counter	24K bit
Loop predictor	48-entry 37-bit / entry	1776 bit
Global history	880 + 16bit path history	896 bit
Local history	64 entry * 14 bit	896 bit
Other counter	UA:5*5*4bit, TICK: 8bit, DC:10bit,UC:5bit,UT: 6bit, WITHLOOP: 7bit RANDOMSEED: 6bit	142 bit
Total	**Limit:263168 bit**	**262782 bit**

Table 3. Configuration of GL-TAGE predictor (4KB)

	# of entry	Tag Width	Global History Length Local History Length	Budget (Kbit)
Base	2K	4 entries share 1 hysteresis bit		2.5
T0,T1,T2	Share	8-bit tag	5, 7, 11	Total
	0.5K	+ 4-bit	0,0,0	6.0
T3,T4,T5	Share	9-bit tag	16,23,34	Total
	1.0K	+ 4-bit	0,0,0	13.0
T6,T7,T8,	Share	10-bit tag	50,74,108,159,233,342	Total
…T11	0.5K	+ 4-bit	1,1,2,4,5,5	7.0
Total				**28.5**

Table 4. Budget Count for 4KB+1024bit Configuration

Resource	Configuration	Budget
GL-TAGE	See Table 4	28.5K bit
Statistic Corrector	1024 entries 4-bit counter	4K bit
Global history	342 bit + 8 bit path history	350 bit
Local history	16 entry * 5 bit	80 bit
Other counter	UA: 4*4*4bit, TICK: 8bit UC: 4bit, UT: 6bit	82 bit
Total	**Limit:33792 bit**	**33792 bit**

6. Conclusion

In this paper, we have proposed the global-local combined branch history that combines global history and local history. This combined history is useful to capture the behavior correlated with local history on tagged predictors. We have applied combined branch history for the state-of-the-art partially tagged branch predictor and implement other optimization effort. The prediction accuracy achieves 2.401 MPKI with 32.125KB hardware budget. Although our contribution for the prediction accuracy is small, we believe that the combined branch history is beneficial for many other tagged predictors which include indirect branch predictor [6], memory controllers and value predictors.

Table 5. Prediction Accuracy for Each Benchmark

	MPKI 32KB	MPKI 4KB		MPKI 32KB	MPKI 4KB		MPKI 32KB	MPKI 4KB
fp1	1.034	1.430	mm5	3.281	4.892	spec08	0.567	0.816
fp2	0.449	0.747	serv1	0.782	4.327	spec09	3.277	4.056
fp3	0.015	0.015	serv2	0.759	4.611	spec10	0.588	1.417
fp4	0.015	0.091	serv3	2.694	4.676	spec11	0.394	0.610
fp5	0.008	0.027	serv4	1.789	4.823	spec12	10.773	11.515
int1	0.121	1.739	serv5	1.535	4.223	spec13	4.912	8.023
int2	4.206	6.366	spec00	1.441	2.369	spec14	0.001	0.001
int3	6.453	9.034	spec01	6.636	7.373	spec15	0.301	0.873
int4	0.510	1.038	spec02	0.370	1.978	spec16	2.715	3.118
int5	0.068	0.263	spec03	0.637	1.086	spec17	2.441	3.414
mm1	6.467	7.385	spec04	7.933	9.338	spec18	0.003	0.063
mm2	8.632	9.561	spec05	4.335	5.154	spec19	0.963	1.406
mm3	0.055	0.087	spec06	0.592	0.795			
mm4	0.872	1.242	spec07	7.422	15.193	avg.	2.401	3.629

7. REFERENCES

[1] Y. Ishii, “Fused two-level branch prediction with ahead calculation,” The Journal of Instruction Level Parallelism, vol. 9, May 2007.

[2] Y.Ishii, K. Kuroyanagi, T. Sawada, M. Inaba, K. Hiraki, “Revisiting Local History to Improve the Fused Two-Level Branch Predictor”, in 3rd Championship Branch Prediction, June 2010

[3] V. Desmet, L. Eeckhout, and K. De Bosschere, “Improved composite confidence mechanisms for a perceptron branch predictor,” J. Syst. Archit., vol. 52, pp. 143–151, 2006.

[4] A. Seznec, “Analysis of the o-geometric history length branch predictor,” in Proceedings of the 32nd annual international symposium on Computer Architecture, ISCA ’05, pp. 394–405, 2005.

[5] A. Seznec, S.Felix, V.Krishnan, and Y.Sazeides, “Design Tradeoffs for the EV8 branch predictor”, in Proceedings of the 29th annual international symposium on Computer Architecture, ISCA 2002.

[6] A. Seznec and P. Michaud, “A case for (partially) tagged geometric history length predictors”, The Journal of Instruction Level Parallelism. vol. 8, April 2006.

[7] A. Seznec, “A 64 Kbytes ISL-TAGE branch predictor”, in 3rd Championship Branch Prediction, June 2010

[8] A. Seznec “A New Case for the TAGE Branch Predictor”, in Proceedings of Microarchitecture, Dec 2012.

[9] D. A. Jim´enez and C. Lin, “Dynamic branch prediction with perceptrons,” in Proceedings of the 7th International Symposium on High-Performance Computer Architecture, HPCA ’01, pp. 197–206, 2001.

[10] L. Porter and D. M. Tullsen, “Creating artificial global history to improve branch prediction accuracy,” in Proceedings of the 23rd international conference on Supercomputing, ICS ’09, pp. 266–275, 2009.

[11] A.Jaleel, K.B.Theobald, S.C.Steely Jr, and J. Emer “High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)” in Proceedings of the 37th annual international symposium on Computer Architecture, 2010.