A Penalty-Sensitive Branch Predictor

Yue Hu David M. Koppelman Lu Peng
Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803
yhu14@lsu.edu, koppel@ece.lsu.edu, lpeng@lsu.edu

Abstract

Branch predictor design is typically focused only on minimizing the misprediction rate (MR), while ignores misprediction penalty.Because the misprediction penalty varies widely from branch to branch, performance might get improved by using a predictor that makes a greater effort to predict high-penalty branches, at the expense of the other, even if the total number of mispredictions doesn't change.

A penalty-sensitive predictor was developed based on this idea. It includes a penalty predictor to predict whether a branch is high or low penalty. Then, a two-class TAGE predictor is developed to favor high-penalty branches at the expense of low-penalty branches. Experiment shows although the overall performance improvement is limited, the penalty-sensitive mechanism successfully decreases the MR of the high-penalty branches while increasing the MR of the low-penalty branches by a small amount.

1. Introduction

Branch predictor design is typically focused only on minimizing the misprediction rate, while ignores misprediction penalty [1-7]. Penalty, the amount of time the system is not fetching along the correct path, includes pipeline refilling time plus any delay in evaluating the branch condition, perhaps due to dependencies. Because the misprediction penalty varies widely from branch to branch, performance might get improved by using a predictor that makes a greater effort to predict high-penalty branches, at the expense of the other, even if the total number of mispredictions doesn't change. The design of such a penalty-sensitive branch predictor is presented here.

The rest of this paper is organized as follows. In section 2 we introduce our design which is composed of three sub-predictors. Then, in section 3 some experiment results about our branch predictor are presented. Next, the storage requirement is listed in section 4. Finally, section 5 concludes the paper.

2. Design Overview

As shown in Figure 1, the overall design is composed of three sub-predictors: a penalty predictor, a two-class TAGE predictor, and a loop predictor.

The two-class TAGE predictor is our main predictor. It is connected to the penalty predictor which can predict whether a branch is high penalty or low penalty. After accessing the penalty information, the two-class TAGE predictor is able to favor high penalty branches, while only provide normal operations for low-penalty ones. The loop predictor works as an assistant predictor to the two-class TAGE predictor. It gives the final prediction only when it is beneficial to the overall prediction.

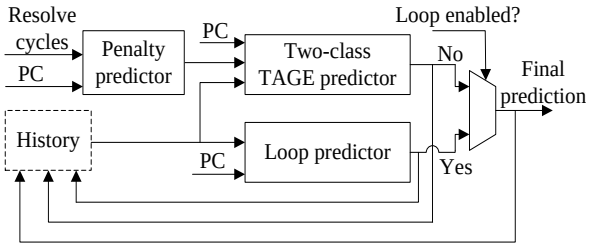

Figure 1. Overall structure of our design

2.1. Penalty Predictor

The penalty predictor is used to determine, when a branch is mispredicted, whether to make a normal or a high-penalty allocation for this branch. The predictor was designed to make a high-penalty prediction for a branch that has more than one high-penalty recovery out of every eight low-penalty ones. Once a branch is predicted to be high-penalty it will keep that prediction for over a hundred executions.

The penalty predictor uses a PC-indexed penalty table; each entry holds an 8-bit penalty counter and a state bit. The penalty counter is incremented by 8 for a high-penalty branch and decremented by 1 otherwise. A branch is regarded as a high-penalty branch if the time it takes for this branch to flow from the fetch stage of the pipeline (when gives a prediction) to the retire stage (when the branch is resolved) exceeds a threshold, 120 cycles for the competition configuration. The state bit is set to high-penalty when the counter reaches 192, and will not reset to low-penalty until the counter reaches zero. The table size was 1024 entries for the competition.

2.2. Two-class TAGE Predictor

A two-class TAGE predictor provides higher prediction accuracy to branches that predicted high-penalty than to the other branches. The TAGE predictor [1-3], a former CBP winner, can easily be made into a two-class predictor because it uses multiple tables and these multiple tables can predict for the same branch simultaneously.

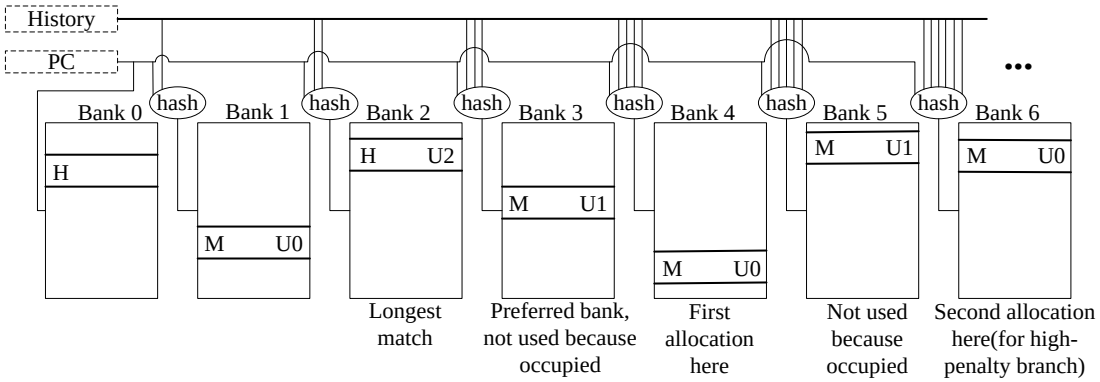

Figure 2. Allocation illustration of the two-class TAGE predictor

The two-class TAGE predictor consists of several banks, each containing a history-indexed table. The history consists of branch outcomes and addresses. Higher numbered banks are indexed using successively longer history. Figure 2 shows the tables with entries highlighted that are at the index for the current branch/history pair. Only two (in Bank 0 and 2) of the highlighted entries are actually for the current pair, indicated by an H (hit), the others, marked M (miss), are for other pairs; they are ignored when making a prediction and might be overwritten when performing an allocation. Table entries hold a 3-bit counter for predicting branch direction, a varied-length tag for detecting hits, and a 2-bit useful counter, described below. Table entries are allocated on a misprediction; the text beneath the banks refers to an allocation for the current branch, which was mispredicted by the entry in Bank 2.

A branch/history pair that was not found in any table would be allocated in Bank 0 (a bimodal predictor), a branch that had a mispredicted entry in an existing bank would have a new entry allocated in a higher-numbered bank, where the longer history might avoid the misprediction [1-3]. The latter situation is illustrated in the figure. Bank 2 holds the entry used to make the prediction while the entries in higher-numbered banks are allocation candidates.

The useful counter is initialized to zero on an allocation and then later incremented when the entry was used to make a useful and correct prediction. A counter value is shown after the U in each highlighted entry. An entry is considered useful only when the useful counter is positive, otherwise it is considered vacant for purposes of allocation.

The bank in which to make a normal allocation is based on the largest bank number in which a matching entry for the predicted branch is found; call that bank number b; it is Bank 2 in the figure. The preferred bank in which to allocate a new entry is Bank (b+r), where r is 1, or 2, or 3, randomly chosen. The entry is placed in the preferred bank if it is not occupied by a useful entry. Otherwise, the entry is placed in the smallest numbered bank in which the entry has a useful count of zero, Bank 4 in the figure. For further details see [1-3].

The TAGE predictor is made into a two-class predictor by allocating two entries (rather than one) for high-penalty branches. The entries are allocated in different banks, the first bank is chosen in the same way as for the normal low-penalty branches, the second bank is chosen as close to the first bank as possible. This double allocation increases the chance that a new entry will survive long enough to establish its usefulness.

For a mispredicted high-penalty branch, the second entry would be allocated if two conditions are met: First, there must be at least one more bank with a vacant entry. Second, the last two allocations at the preferred bank had to be normal allocations; this is called the spacing restriction. These restrictions were carefully chosen so that a high-penalty allocation does not induce more mispredictions than it avoids, including for the cases where all branches double allocate. In the figure, the second allocation is made to Bank 6.

2.3. Loop Predictor

1) Prediction

The loop predictor is enabled to give the final prediction only when three conditions are all satisfied:

a) On the whole, the loop predictor is beneficial to the main two-class TAGE predictor (it is beneficial when the beneficial counter, named WITHLOOP in program is positive, otherwise negative);

b) The coming PC hits the loop table, at the same time, the hit loop entry is high confident (the loop pattern has already been detected);

c) At fetch stage, the loop branch has been synchronized. Or in another word, a not-taken branch of this loop has been detected.

2) Update

At retire stage, the actual result of the current branch and its two predictions made by the loop and two-class TAGE predictors at fetch stage are used to update the loop predictor. Since both the fetch and retire stages are in-order, this can be implemented in hardware by simply adding two prediction bits to each entry of the reorder

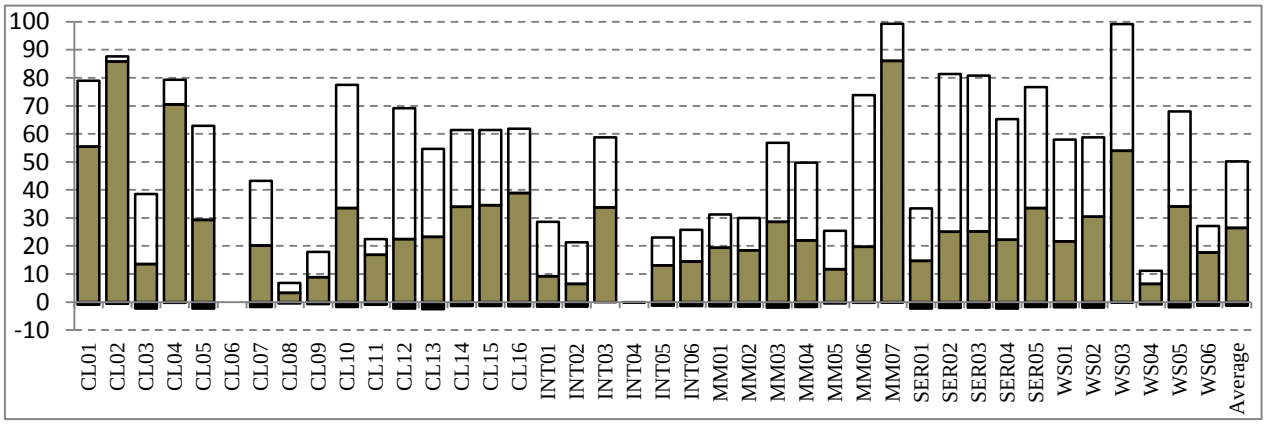

Figure 3. Penalty prediction statistics of the given 40 benchmarks
(CL: Client, INT: Integer, MM: Multi-media, SER: Server, WS: Workstation)

buffer (ROB). The update of the loop predictor can be divided into the following two classes.

a) Loop pattern detection:

A loop pattern is detected, as known as being high confident when the loop has already been successively executed three times with the same number of iteration.

b) When the loop predictor is enabled:

If the loop prediction is different from the two-class TAGE prediction, increase the beneficial counter WITHLOOP when the final prediction is right; or decrease it otherwise. Once a loop prediction is found to be wrong, the entry corresponding to that branch will be cleared immediately.

3. Experiment Results

3.1. Penalty Predictor Performance

The penalty predictor performed well on the competition benchmarks. Its performance on individual benchmarks together with their average appears in Figure 3. Bar height above the *x* axis indicates the percentage of branches that was predicted to be high-penalty by our penalty predictor, while the height of the lower gray segment shows the percentage of branches that was actually high-penalty among all branches. Below the *x* axis, the depth of the dark bar indicates the percentage of high-penalty branches among those predicted low-penalty.

On average, 50.2% of executions were predicted high-penalty and about 27% of executions were actually high-penalty. Only 1.3% of the executions which was predicted low-penalty turned out to be high-penalty. Overall, according to our statistics, among branches that were falsely predicted by our penalty-sensitive predictor, the average penalty of branches that predicted high-penalty was 212 cycles, while the average penalty of branches that predicted low-penalty was 121 cycles, nearly half.

3.2. Two-class TAGE Predictor Performance

The two-class TAGE predictor was designed so that the prediction accuracy of the high-penalty branches would improve at the expense of the low-penalty branches. To verify this, the misprediction rate (MR) was collected separately for branches predicted to be low-penalty and high-penalty. This data appears in Table 1. On average, the MR was 0.03213 for the low-penalty branches and 0.03508 for the high-penalty branches on the base LTAGE predictor. When the two-class TAGE predictor was used, PSLTAGE, the average MR of the low-penalty branches increased by 0.00003, while that of the high-penalty decreased by 0.00006. The overall MR dropped since about half of the branches were predicted high-penalty. Compared with the LTAGE, our design-PSLTAGE's performance was improved, although only a small amount.

Table 1. Performance statistics

Predictor	Low-latency Bran.		High-latency Bran.	
	MR	Penalty	MR	Penalty
LTAGE	0.03213	121.13	0.03508	212.12
PSLTAGE	0.03216	121.02	0.03502	213.64
LTAGE2x	0.03151	121.81	0.03461	212.76

To provide a reference on what sort of performance improvement to expect, the performance of a larger LTAGE predictor was measured, one in which all tables except the bimodal and loop predictor, were twice as large, shown as LTAGE2x in Table 1. For LTAGE2x, the MR was 0.03151 and 0.03461 for low- and high-penalty branches respectively. Looking again at the drop of 0.00006 in the MR of the high-penalty branches, we see that it is 12.8% of the decrease that one would get by doubling the table sizes. Though this certainly is not a large fraction of the potential it is respectable given that there was no increase in the size of the TAGE predictor.

4. Storage Requirement

The storage requirement of our predictor is as follows.

4.1. Two-class TAGE Predictor

Table 2 shows the storage budget and configuration of the two-class TAGE predictor. The base table is used

Table 2. Storage budget and configuration of the two-class TAGE predictor

Table Name	Base	T1,2	T3,4	T5	T6	T7	T8,9	T10	T11	T12
History Length	0	5,8	13,21	33	53	85	136,218	350	561	900
Number of Entries	32K	2K	4K	4K	4K	2K	2K	1K	1K	0.5K
Tag Width	0	9	10	11	12	12	13	14	15	16
Storage Budget (bits)	40K	28K	60K	64K	68K	34K	36K	19k	20K	10.5K

by a bimodal predictor. With the help of hysteresis technique, here each four entries share the same lower bit of the conventional bimodal entries. Therefore, its budget is (1+1/4)*32K = 40 Kbits. According to the section 2.2, we can see for each table Ti (i=1-12), its storage requirement is equal to (3 + 2 + Tag width)*(Number of Entries). So, the total budget listed in Table 2 is 503.5 Kbits.

Apart from the storage budget listed in Table 2, the two-class TAGE predictor also has the following extra budget. There are two 900-bit global history, two 16-bit path history, a 4-bit "using alternate prediction on newly allocated" counter, a 21-bit periodic reset counter and a 2-bit seed counter used for random number generation. Therefore, these extra budget is equal to (2*900 + 2*16 + 4 + 21 + 2) = 1852 bits.

4.2. Loop Predictor

The loop predictor has 64 entries. For each entry, 14-bit detected iteration counter + two 14-bit current iteration counters individually for the fetch and retire stage + 14-bit tag + 2-bit confidence + 8-bit age + 1-bit prediction enable = 67 bits. In addition, there is a 7-bit WITHLOOP counter to monitor whether the loop predictor is globally beneficial or not. So, it requires 64*67 + 7 = 4295 bits.

4.3. Penalty Predictor

The penalty predictor has 1024 entries. Each entry consists of an 8-bit penalty counter and one penalty state bit. Besides, for each two-class TAGE table Ti (i= 1-12), there is a 2-bit counter to record the number of single-entry allocations between each double-entry allocation. So, 1024*(8+1) + 12*2 = 9240 bits in total.

4.4. Others and Overall

In addition to the up-mentioned storage budget, at retire stage, the two-class TAGE predictor and loop predictor also need three prediction bits that made at the fetch stage for update. There is one bit from the loop predictor, two bits from the two-class TAGE predictor (one for TAGE prediction, the other for alternate TAGE prediction) [1-3]. According to the implementation method introduced in section 2.3, we can implement this by adding three bits to each entry of the ROB whose size is 256 in the competition. So, it requires 256*3 = 768 bits.

Therefore, the total storage budget of our branch predictor is 503.5*1024 + 1852 + 4295 + 9240 + 768 = 531747 bits which is no larger than the given storage budget 65KB (65 * 1024 * 8 = 532480 bits).

5. Conclusion

The design for a penalty-sensitive branch predictor has been presented, consisting of a penalty predictor, a two-class TAGE predictor and a loop predictor. Experiment shows the penalty predictor successfully identifies high-penalty branches whose average misprediction penalty is nearly as twice large as that of the low-penalty branches. The two-class TAGE predictor has been developed which is successful at improving the prediction accuracy of the high-penalty branches at the expense of the low-penalty one.

By one measure, it realizes 12.8% of the potential improvement in MR. A natural question is whether more of this potential can be realized. The high-penalty allocation mechanism was designed conservatively so that the second table entry is allocated only when the entry to be replaced is useless. A variation, in which this restriction was removed, so that the second entry can be allocated every time, resulted in worse performance. The double allocation mechanism gives high-penalty branches an advantage when allocating new entries, but does not make their established entries any less vulnerable. In a possible design, alternative table entries can have a class bit which indicates whether the branch is high or low penalty; the bit can be used to prioritize allocations. This class bit consumes storage, and so reduces the performance when all branches fell into a single class. For that reason it was not tried.

References

[1] A. Seznec, "The L-TAGE Branch Predictor," Journal of Instruction Level Parallelism, May 2007.

[2] A. Seznec, "A 256 Kbits L-TAGE branch predictor," 2nd Championship Branch Prediction, Dec 2006.

[3] A. Seznec, "A case fro (partially) Tagged Geometric history length branch prediction", Journal of Instruction Level Parallelism, Feb 2006.

[4] H. Gao and H. Zhou, "Adaptive Information Processing: An Effective Way to Improve Perceptron Predictors", 1st Championship Branch Prediction, Dec 2004.

[5] S. McFarling, "Combining Branch Predictors", Technical Report, DEC, 1993

[6] R. Amant, D. Jimenez and D. Burger, " Low-power, High-Performance Analog Neural Branch Prediction", MICRO, Dec 2008

[7] H. Kim, O. Mutlu, J. Stark and Y. Patt, "Wish Branches: Enabling Adaptive and Aggressive Predicated Execution", MICRO, Jan, 2006