Indirect branch predictor ITTAGE implementation

Swathi Changalarayappa

Electical and Computer Engineering Department, Texas A&M University

swathi_c@tamu.edu

***Abstract*—It is widely accepted that a conditional hazard caused by a branch in a pipelined processor can cause a high latency and can significantly affect the overall performance. An indirect branch misprediction causes the same penalty as that of a conditional branch misprediction. This results in the need for a very accurate indirect branch predictor. This paper is an implementation of ITTAGE proposed in [1]. ITTAGE stands for INdirect TAgged GEometric length predictor. ITTAGE was proposed along with TAGE in [2]. TAGE is a highly accurate conditional branch predictor, whose ideas can be very well applied to indirect branches too. ITTAGE prediction uses multiple prediction tables indexed through hash functions of global history and PC. The use of geometric lengths as proposed in [3] allows the prediction to use correlation with large global bit history range. Due to no constraint on the storage budget for the competition, the ITTAGE proposed in this paper uses 4K global history bits and 10 indexed prediction table along with 1 base predictor. The results obtained are highly accurate as compared to the given basic predictor.**

Keywords—ITTAGE, TAGE, indirect, branch, predictor

I. INTRODUCTION

The accuracy of indirect branch predictor depends on 3 main features: (1) where to place the target, (2) how to tag the target, (3) how to replace the targets. The first determines on how we differentiate between multiple branches in the program, second determines on how we differentiate between multiple targets for same PC and the third feature determines how efficiently we use the space. A good branch predictor should capture the flow of program and save it. It must be able to recognize the same flow and predict accordingly for best results. ITTAGE is a unique indirect branch predictor that does the above with utmost accuracy.

II. IMPLEMENTATION

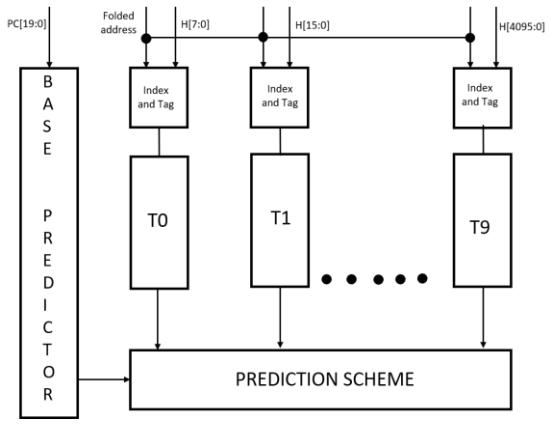
The ITTAGE predictor, similar to TAGE predictor, consists of a tagless base predictor table indexed with 20 bits of PC. This table is kept up to date with the latest target for all the PCs. The efficiency of the ITTAGE comes from 10 tagged predictor tables indexed using hash functions of global path history and PC. The number of bits used to index the table forms a geometric series as proposed in [3]. This lets us exploit the advantages from very long history as well include the correlations between short histories. This paper considers results for the geometric length series: 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096.

Each entry in a predictor table has a tag, target, a 2-bit counter and a useful bit. Tag helps in differentiating between 2 different targets for the same address. 2-bit counter is used to identity how accurate the target is. A value of 3 would mean the entry corresponds to right predictions and a value of 0 means otherwise. The useful bit is used to avoid the entry getting stolen by a new target. If the entry is marked as useful, it means that the entry had served as an accurate predictor in the past and must not be replaced.

A. *Prediction scheme*

For a new PC address, compute the index for all tables. Check through entries if a tag matches. The final predicted target will be from the table of longest history where there is a tag match. If there is no prediction from the prediction tables, the target prediction is provided by the base predictor.

In addition to the predictor component, an alternate prediction, known as altpred, is determined too. Altpred is nothing but the prediction component, that would have been chosen, if there was no pred component. Altpred is the next longest tag matching entry from the prediction tables. Sometimes it is possible that pred, being a new entry in table, might not be the most accurate prediction. In such cases, we choose altpred as the target prediction component. We determine an entry as new when its useful bit is zero and counter is 1. We also use a global USE_ALT_ON_NA 4-bit counter that saturates at 15 when altpred provides accurate results than pred and saturates at 0 otherwise. This is used to choose between pred and altpred.



B. *Indexing the tables*

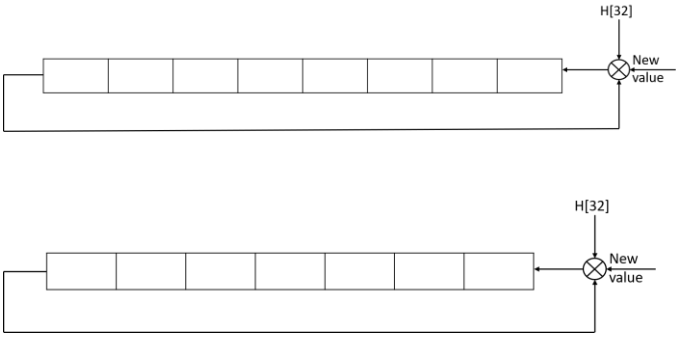
Each table is hash indexed based on history length assigned to it. The index computed is: PC XOR H, where H is the global history. Here, both PC and H are folded onto 8 bits. For

example, for table 1 indexed using 16 H bits and considering PC as 32 bits, the index is obtained as:

PC[31:24] XOR PC[24:16] XOR PC[15:8] XOR PC[7:0] XOR H[15:0] XOR H[7:0]

C. Tag computation

Each entry in the predicted tables should be given a tag, that would correlate with the history. Here, the tag computation used is: PC [7:0] XOR CSR1 XOR CSR2. CSR1 is the global folded history corresponding to the table and CSR2 is also global folded history, but folded through 7 bits instead of 8. The 2 circular shift registers implemented for table2 are shown here.



This is inspired from [3]. It has been observed that the correlation that the tag provides improves prediction capability by a great extent.

D. Update policy

Once a prediction is made, a few entries in the prediction scheme needs to be updated as to predict better in future. The prediction policy is applied as follows:

1. If the prediction was correct, simply increment the counter of prediction provider.
2. If the prediction was wrong:
    a. decrement the counter of prediction provider. If the counter value is 0, replace the target with the new target.
    b. Assign 3 values in the tables (table values greater than the provider table) with the latest target and tag. Assign only on those spots where the useful bit of it is zero.

The useful bit of newly allocated entry is set to null and the corresponding counter is set to weak (which is 1 for 2-bit counter). The newly allocated entries are also not placed in consecutive tables.

E. Reset policy

It is possible that a lot of entries are marked as useful in the tables, hence not allowing new entries to be placed. This can be overcome by having the single global 8-bit counter. This counter increments when useful bits are 0 and decrements when useful bits are 1 during the search to place new elements.

III. ENHANCEMENTS

A number of improvements were tried on the ITTAGE predictor. Some of them were used and some were not. They are listed here:

A. A new reset policy

The given reset policy for useful bits is such that it counts the number of times it can't place a value because of useful bit set to 1 and resets accordingly. A new method was tried, where in, all the useful bits are reset, when the update program was unable to allocate any entry. This method did improve a prediction on some benchmarks, but not all. The improvement on some benchmarks were so insignificant compared to performance loss on others that this reset policy is not incorporated.

B. Altpred usage when altpred counter is greater than 0

ITTAGE uses a simple logic on when to use an altpred. It states that altpred is used when the provider component is null. In this implementation, this definition is extended to saying that an altpred is used when the pred component is new (useful bit is zero and counter value is one) and altpred counter is more than one. This idea is inspired from the observation that an altpred is not a useful entry if its counter value is zero. This reduces the probability of wrong prediction using altpred when pred could have predicted right. This improvement is utilized in the submitted code.

C. Base table improvement

Different benchmarks behave differently with respect to some parameters. One of the observation was on a particular trace SHORT_SERVER-28 that gives the worst prediction for the implemented ITTAGE. On close observation, it was found that this trace has a majority of conditional branches as compared to indirect. These conditional branches are not well correlated and results in a high prediction rate. It can also be observed that the trace has high correlations among indirect branches. Hence, using only indirect branches on global history improves the conditional branch prediction by a good extent. However, this is not a generalized idea, since it worsens the performances of so many other benchmarks, that have a good correlation between conditional and indirect. This is also mentioned in [1]. An alternative is to somehow also save the correlation between only indirect branches.

None of the ITTAGE or related geometrical history length based papers ever talk about the importance of a good base predictor. When the correlation is poor among branches, the predictor goes to a base table, and hence a good base table can improve performance. Here, we propose a base table that is indexed as a hash function of PC and previous target from an indirect branch. The implementation of it PC[19:0] xor previous_target[19:0]. We use 20 bits as our base predictor uses 20 bits to index.

An overall improvement of 5.3% is obtained with this optimization. It can be observed that this enhancement improved performance for all benchmarks and hence can be assumed as a global feature for base table implementation.

The above proposed base table should have improved performance on SHORT-SERVER-28 by a great margin, but it did not. The reason being, as new entries keep filling the predictor tables, the predictor seldom reaches the base table. Hence, more intelligence has be added to this feature to select between base table or predictor table dynamically.

IV. RESULTS

The ITTAGE implementation is run on a set of benchmarks as provided in the competition. A default prediction scheme given with the setup produces a result of 7.156 MPKI on 69 benchmarks.

With the implemented ITTAGE, a result of 0.229 MPKI is observed on the same 69 benchmarks. This is an overall 96.79% improvement as compared to the base prediction scheme provided.

A number of factors effect on how well an ITTAGE can perform. A bad correlation between branches is a nightmare to any predictor, as seen in example of SHORT_SERVER-28. Another such drawback of a program flow that can hinder a branch predictor is a large number of different branches. A branch with multiple targets can be handled well with ITTAGE but different branches with large number of targets fills up the predictor tables exponentially. These values if not referred to frequently can be evicted from the tables giving a misprediction. This is observed on few traces – SHORT-MOBILE-20 and SHORT-MOBILE-21. Also, the values placed and not referred to frequently can affect the prediction scheme. A better replacement scheme, increasing the number of prediction tables or history bits might help improve their performances.

Initially since the predictor tables are empty, we observe a high misprediction rate. Hence, the more instructions get executed (or more precisely, more number of times the branches get executed), the lower is the MPKI.

V. CHALLENGES

Below are some of the main challenges faced while implementing ITTAGE:

- Complexity: ITTAGE is a complex algorithm to understand and implement. It is inspired from many works such as [2], [3] and [4] and hence requires a good understanding of history on branch predictors to accurately implement its algorithm.
- ITTAGE relies on a very accurate update policy using useful bits, counters and resetting them. Any incorrect implementation of it can result is higher mispredictions.
- Some of the enhancements on ITTAGE are targeted to reduce memory usage. Such enhancements are not implemented due to huge storage budget available for the competition.

VI. CONCLUSION

ITTAGE is the best in class indirect branch predictor till date. The accuracy obtained with it can improve the processor performance by many fold. However, mispredictions cannot be attributed to a single cause. Further analysis of branches and their behaviors can be exploited to improve the misprediction rate. There is a potential for improvement especially by working on traces that give a high misprediction rate such as SHORT_SERVER-28, one of which is presented in this paper. An accurate predictor must dynamically be able to tweak its own features so as to provide a good prediction on all traces.

REFERENCES

[1] Andre Seznec. A 64-KBytes ITTAGE indirect branch predictor. *JWAC-2: Championship Branch Prediction,* June 2011.

[2] Andre Seznec and Pierre Michaud. A case for (partially)-tagged geometric history length predictors. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol8)*, April 2006.

[3] A. Seznec. Analysis of the o-gehl branch predictor. In Proceedings of the 32nd Annual International Symposium on Computer Architecture, june 2005.

[4] Pierre Michaud. A ppm-like, tag-based predictor. Journal of Instruction Level Parallelism (*http://www.jilp.org/vol7*), April 2005.