Exploring Correlation for Indirect Branch Prediction

Nikunj Bhansali Chintan Panirwala Huiyang Zhou
Department of Electrical and Computer Engineering
North Carolina State University
{nsbhansa, cdpanirw, hzhou}@ncsu.edu

Abstract

In this paper, we present a highly accurate predictor for indirect branches. The baseline is a tagged PPM-like (Prediction by Partial Matching) structure. The pattern to be matched includes global taken/not-taken and path history of both conditional and indirect branches. Rather than the conventional way of using the longest matches for prediction, we propose simple yet effective ways to adaptively select from matches with both short and long histories. We also designed an auxiliary predictor to exploit the correlation between the addresses of producer loads and the targets of consumer indirect branches.

1. Introduction

Besides conditional branches, indirect branches with multiple targets present a challenge for processor performance, especially when the targets are dependent on long-latency computation or memory accesses. Mispredictions of such indirect branches result in significant performance loss as well as energy wasted on executing instructions along wrong paths. In this paper, we propose a predictor to achieve high prediction accuracy for indirect branches.

1.1. Related work

Previous studies have shown that history information of control flow carries strong correlation to the targets of indirect branches. To capture such correlation, target caches [1] were proposed, in which the targets of an indirect branch with different history are maintained separately. Cascaded predictors [2] were proposed later on to combine multiple target caches with different history lengths. The Prediction by Partial Matching (PPM) algorithm provides a more generic model to exploit control-flow history information. PPM-based predictors [5] contain multiple Markov predictors with each capturing a different history length and the one with the longest match will be used to make the final prediction. The ITTAGE predictor [7] is an efficient way to implement the PPM algorithm with very long histories. Another way to exploit branch history information for indirect branches is to view each target of an indirect branch as a virtual branch [6]. This way, a conditional branch predictor can be (re)used to predict which virtual branch is taken and the associated target will be the prediction of the indirect branch.

Our proposed predictor leverages previous work based on the PPM algorithm, the ITTAGE predictor in particular, for indirect branches. Our key idea is that *rather than always using the Markov model with the longest match to make a prediction, we adaptively select the Markov models with proper history lengths*. Similar observations that the longest matches may not be the best choice have been made in our previous work [3] on conditional branch prediction.

Another novelty of our work is that we found that *there exists very strong correlation between the producer load addresses and the consumer branch targets*, similar to address-branch correlation explored for conditional branches [4]. We exploit such correlation at the address generation (AGEN) stage of a load instruction to make early correction of mispredicted indirect branches.

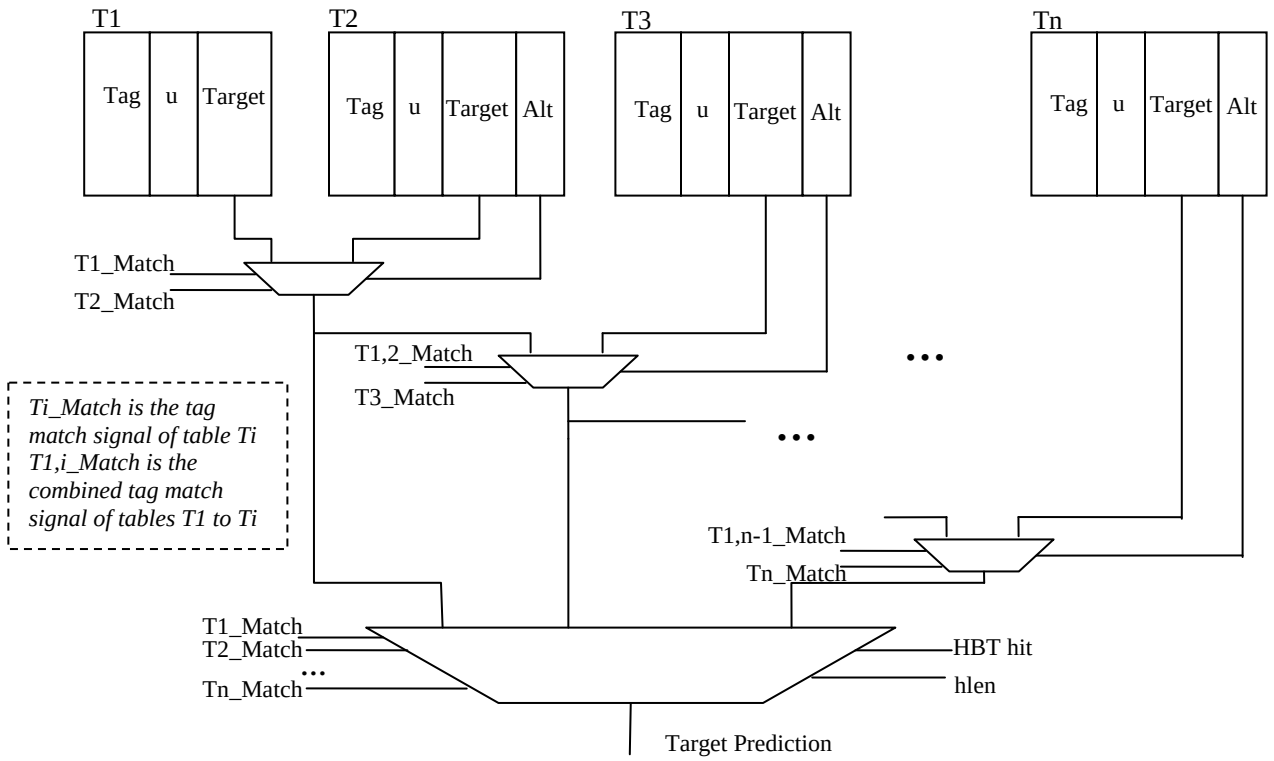
1.2. Outline

In section 2, we present the design of our proposed predictors and discuss its implementation issues. Section 3 reports the detailed information of the storage cost of our proposed design. Section 4 briefly analyzes the experimental results. Section 5 concludes the paper.

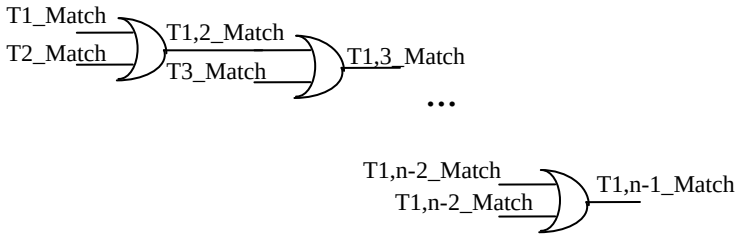
2. Design

2.1. A Main Predictor at the Fetch Stage

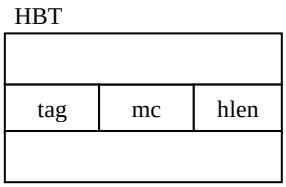
The baseline of our design is the ITTAGE predictor. It has multiple tagged tables with each capturing a specific history length. In our design, we use the same multiple-table structure. The first table (T1) uses the shortest history to generate its indices and tags while the last table (Tn) uses the longest history for its tag and index functions. Our key improvement over the ITTAGE predictor is that rather than using the cascaded design to select the table providing the longest match, we adaptively choose which table to be used for the final target prediction. In addition, we choose not to include a base predictor, which is to be indexed with the program counter (PC) without any history information, due to its



(a) Predictor Structure



(b) Tag Match Signals



(c) A table for hard-to-predict branches

Figure 1. The main predictor to exploit correlation in control-flow history

limited usage and relatively high storage cost. The overall predictor structure is shown in Figure 1a.

As shown in Figure 1a, in all the tables except T1, each entry has 4 fields, tag, u (usefulness for replacement), target and alt. We design two ways to choose a table to provide the final prediction when there is more than one table having tag matches. The first is through the alt field. If this one-bit field is clear, it means that the target from the current entry is preferred for prediction. If the field is set, it means that a table with shorter history is to be used to make the final prediction. Such logic is implemented using the multiplexers, which are controlled with the alt fields, as shown in Figure 1a. As T1 is the one with shortest history, there is no need for the alt field. Initially, the alt fields in all the tables are set to false, which is functionally equivalent to selecting the longest match. During the update phase, if it is found that the table with the longest match fails to make the correct prediction while another table does, the alt field fields will be set for those entries with longer history lengths. The tag match signals shown in Figure 1a and Figure 1b are used to ensure that the tables without a tag match will not be used for final prediction.

Our second way to select the proper table to provide the prediction is to use another multiplexer controlled by a separate structure, called a hard-to-predict branch table (HBT). The HBT is a cache-like set- associative structure and each entry contains a tag, a misprediction counter (mc), and a history length (hlen) field, as shown in Figure 1c. The mc field is used for replacement so that only those indirect branches with high misprediction rates are kept in the table. The HBT is updated based on the prediction made using the longest match rather than the actual prediction depending on the alt fields. Since

the branches in HBT are mispredicted using the longest match, we increase the hlen field, whose value, *x*, is then used to select the table with the x^{th} longest histories. For example, if the hlen of an indirect branch is 2 and tables T2, T4, and T5 have tag matches and their corresponding alt fields are false, we will not select the longest match (i.e., T5) or the second longest match (T4). Instead, the table T2 is selected as a result of the hlen field being 2.

Our proposed predictor is accessed at the instruction fetch stage to make a prediction and it is updated at the retire stage of an indirect branch. Due to the different program state at the two stages, the predictor uses two copies of global branch and path history information, one for prediction and the other for update.

2.2. An Auxiliary Predictor at the AGEN Stage

Our proposed predictor described in Section 2.1 predicts targets at the fetch stage of an indirect branch. To reduce the misprediction penalty, we leverage the execution results of the instructions, upon which an indirect branch is dependent.

From our code analysis, we found that many hard-to-predict indirect branches are dependent upon load values. Since the loaded value will be immediately used by the dependent indirect branches, such execution results are available too late to be useful to reduce misprediction penalty. Therefore, we examine the correlation between the producer load addresses and the consumer branch targets, similar to the address-branch correlation observed for conditional branches [4]. Figure 2 shows such an example from one benchmark.

```
Load  R19 = Mem[R3 +…]  //Addr:  x60848100    x60846ec8
…                                    ⇳             ⇳
Br    R19                 // Target: x60751a64    x607691c9
```

Figure 2. A code example for address-target correlation.

From Figure 2, we can see that the producer load accesses two primary addresses and either address contains a different branch target. As long as the data structure (e.g., a virtual function table) at these addresses is not frequently updated, the addresses of the producer loads are sufficient to determine the targets of their consumer indirect branches. We refer to such correlation as address-target correlation (ATC).

In our design, we capture ATC in a small cache-like set associative structure, called address-target table (ATT) as shown in Figure 3. Each entry in ATT contains a tag and multiple pairs of hashed addresses and the corresponding targets.

ATT is accessed at the address generation (AGEN) stage of a load instruction if it has a dependent indirect branch. The PC of the consumer indirect branch is used for tag match to see whether an entry in ATT has been allocated for it. If so, the hashed address of the producer load will be used to compare with multiple address-target pairs in the entry to find a matching pair to provide the prediction for the consumer branch. If the prediction differs from the one made at the fetch stage of the indirect branch, an early misprediction recovery is initiated to reduce the misprediction penalty. ATT is updated at the execution stage of an indirect branch. Only if it is mispredicted at the fetch stage, we search for its producer load address and then update ATT with the actual branch target. The least-recently-used (LRU) replacement policy is used to select a victim in ATT. Random replacement is used if there are more address-target pairs than what each entry in ATT can maintain.

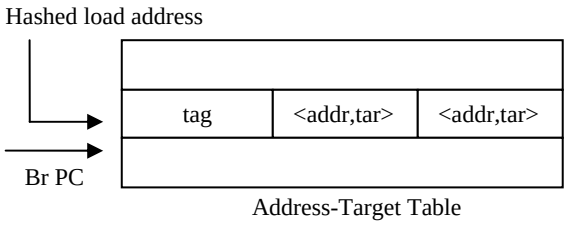


Figure 3. An auxiliary predictor to exploit ATC.

3. Cost of Storage

In our simulator, we adopted the multiple table structure from the L-TAGE predictor [8]. We have a total of 11 tables with the structure described in Section 2. The overall storage cost is presented in Table 1.

4. Experimental Results

Our proposed predictors are implemented in the distributed framework of CBP-3. The performance improvements, measured with reduction in misprediction penalties per 1K instructions across all the benchmarks, of our proposed designs over the baseline ITTAGE predictor are shown in Figure 4.

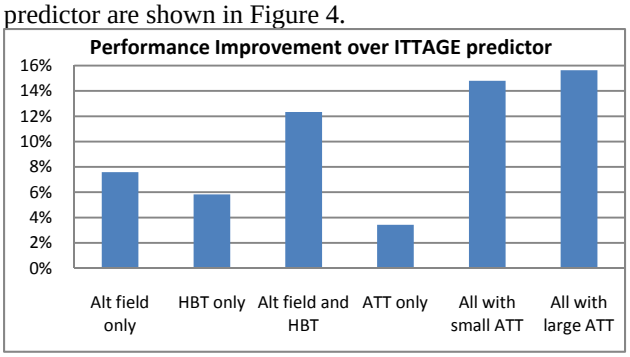


Figure 4. The performance improvement of our proposed design over the baseline ITTAGE predictor.

From Figure 4, we can see that by adding the 1-bit alt field in each entry of the ITTAGE prediction tables, the

performance is improved by 7.6% (labeled 'Alt field only'). In comparison, if we only use HBT to select the proper table instead of the longest matches, the performance is improved by 5.8% (labeled 'HBT only'). The combination of both mechanisms results in an improvement of 12.3% ('Alt field and HBT). Early misprediction recovery using ATT provides a 3.4% improvement ('ATT only'). This relatively limited performance enhancement is due to the fact that the latency between the AGEN stage of a producer load and the EXE stage of its consumer indirect branch is often a small portion of the misprediction penalty, which starts from the fetch stage of the indirect branch. This suggests that for higher performance gains we need to explore address-target correlation beyond the immediate producer-consumer pairs. Nevertheless, ATT provides a highly cost-effective way to improve performance. When all these schemes utilized together, the performance is improved by 15.6% using a large ATT (specification shown in Table 1). For an ATT with much lower cost, an 8-entry ATT with each entry containing 4 address-target pairs, the overall performance improvement is 14.8%.

5. Conclusions

In this paper, we present our design for a highly accurate indirect branch predictor. The key insight includes (a) although control flow history carries correlation to targets, the strength of correlation may either increase or decrease for different indirect branches when we increase the history length; and (b) there exists strong correlation between producer load addresses and consumer branch targets. Our proposed design includes a main predictor to exploit correlation in history and a simple auxiliary predictor to exploit correlation in load source operands. Our results show that our design reduces the misprediction penalty significantly.

6. Acknowledgement

This work is supported in part by an Intel research grant and an NSF CAREER award CCF-0968667.

7. References

[1] P. Chang, E. Hao, and Y. Patt, "Target Prediction for Indirect Jumps", ISCA, 1997.
[2] K. Driesen and U. Holzle, "The Cascaded Predictor: Economic and Adaptive Branch Target Prediction", MICRO-31, 1998.
[3] H. Gao and H. Zhou, "PMPM: Prediction by Combining Multiple Partial Matches", JILP, 2007
[4] H. Gao, et. al, "Address-Branch Correlation: A Novel Locality for Long-Latency hard-to-Predict Branches", HPCA, 2008.
[5] J. Kalamatianos and D. Kaeli, "Predicting Indirect Branches via Data Compression", MICRO-31, 1998.
[6] H. Kim, et. al., "VPC Prediction: Reducing the Cost of Indirect Branches via Hardware-Based Dynamic Devirtualization", ISCA, 2007.
[7] A. Seznec and P. Michaud. "A case for (partially) TAgged GEometric history length branch prediction", JILP, 2006.
[8] A. Seznec, "The L-TAGE Branch Predictor", JILP, 2007.

Table 1. The storage cost the proposed design

Component	Storage Cost
Table 1	2048 entries, each entry has a 7-bit tag, 2-bit u field, 32-bit target address. Total 2048x(7+2+32) = 83968 bits
Table 2	1024 entries, each entry has a 7-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total: 43008 bits
Table 3	4096 entries, each entry has a 8-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:176128 bits
Table 4	2048 entries, each entry has a 8-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:88064 bits
Table 5	512 entries, each entry has a 9-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:22528 bits
Table 6	512 entries, each entry has a 10-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:23040 bits
Table 7	512 entries, each entry has an 11-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:23552 bits
Table 8	512 entries, each entry has a 12-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:24064 bits
Table 9	512 entries, each entry has a 12-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:24064 bits
Table 10	64 entries, each entry has a 13-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:3072 bits
Table 11	64 entries, each entry has a 14-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:3136 bits
Table 12	64 entries, each entry has a 15-bit tag, 2-bit u field, 1-bit alt field, 32-bit target address. Total:3200 bits
HBT	32 entries, each entry has a 32-bit tag, 2-bit u field, 4-bit hlen field Total:1216 bits
History	Global history: 640x2 = 1280 bits
ATT	26 entries, each entry has a 32-bit tag, a 5-bit LRU field, 10*10 bits for storing addresses and 32*10 bits for targets. Total = 11882 bits
Extra	55 bits (path history and few counters)
Overall	532257 bits (64.97kB)