Optimized for Mac

Enoch Huang
Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA
eihuang@g.ucla.edu

Sudarshan Seshadri
Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA
sseshadri26@g.ucla.edu

***Abstract*—This document describes a modified implementation of the TAGE branch predictor for Computer Assignment 1 in the ECE209AS course on Secure and Advanced Computer Architecture at UCLA. Like TAGE, this predictor is a partial-tag based predictor utilizing global history and draws from predictors such as PPM. In our implementation, there are 4 tables using varying history lengths. A few changes exist from the TAGE branch predictor as originally described when introduced. In particular, a change was introduced to the update policy that removed probabilistic allocation of entries to the TAGE table, which had resulted in more variance in the final mispredictions per 1000 instructions (MPKI) than desired.**

***Index Terms*—branch prediction, TAGE, partially tagged, path history, folded history, global history**

I. Introduction

The TAGE branch predictor was introduced in 2006 [1]. Inspired by prior predictors like PPM [2] and O-GEHL [3]. In particular, it combines the tagging and indexing methods of PPM and the geometric sequence length derivation from O-GEHL, and introduces some new contributions of its own, such as within its update policy. Our implementation mostly aligns with TAGE as originally described.

II. Design Methodology

A. History Length

Our implementation uses a 5-component TAGE. The specific geometric sequence is derived from the formula of $L(i) = \left(\text{int}\left(\alpha^{i-1} \times L(1) + 0.5\right)\right)$. From the TAGE paper, an optimum value of $L(i)$ was 5, and α was chosen such that the final sequence matched the Global History Register (GHR) length of 131. Path History was kept at 16 bits in accordance with the TAGE paper.

B. History Compression

In order to compress the 131-bit global history to match the lower-bit index, the same method of history folding was employed as described in the PPM paper. For example, if the index was 10 bits and the history was 30 bits, the compressed history would be equal to h[0:9]^h[10:19]^h[20:29]. A more optimized method is described as follows from the PPM paper and is implemented.

In the above graph, a 20 bit history is compressed into 8 bits. To do so, the compressed history is shifted right, the newest bit in the GHR is XOR'ed with the oldest (soon to be discarded) bit in the compressed history, and the oldest bit in

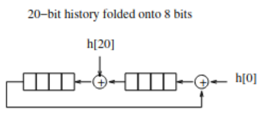

Fig. 1. Example of Compressed History

the GHR is XOR'ed with the bit at position (geometric length % compressed length).

C. Table information

The Bimodal Table is used as a base predictor and has 2^{14} entries. Indexing is based on the PC modulo with the number of entries, in a somewhat similar manner to the GShare predictor. The values in the bimodal table are 2-bit, ranging from 0 to 3, with values 0 to 1 indicating NOT TAKEN and values 2 to 3 indicating TAKEN. Initializing the values to 1 (weakly NOT TAKEN) instead of 2 (weakly TAKEN) provided a small decrease in MPKI of around 0.04.

The TAGE tables were implemented as described in the TAGE paper. Each entry in each TAGE table contains a tag derived from the PC and two compressed global history buffers, a counter used for prediction, and a useful bits counter. The tag was 9 bits for the tables indexed based on longer history, which was optimal per the TAGE paper, and 8 bits for tables indexed based on shorter history. It is computed from the PC and two compressed history buffers XOR'ed with each other, as outlined in the PPM paper. The prediction counter was 3 bits, with values ranging from 0-7 and values 4 or above resulting in a prediction of TAKEN and values 3 or below resulting in a prediction of NOT TAKEN. Subsequently, 7 can be described as strongly TAKEN, 4 as weakly TAKEN, 3 as weakly NOT TAKEN, and 0 as strongly NOT TAKEN. The useful bits counter was 2 bits and incremented based on whether or not the entry historically provided a useful prediction, with a correct prediction resulting in an increment and an incorrect prediction resulting in a decrement. Indexing the TAGE table was computed via a hash combining the PC, compressed global history based on the geometric sequence, and path history register.

D. Prediction Policy

Our prediction policy follows that oulined in the TAGE paper. If a matching tag is found for a given instruction in the TAGE tables, the primary prediction (primePred) comes from that using the compressed index based on the longest global history, with the alternate prediction (altPred) coming from that using the second longest global history. There are circumstances, however where the altPred may prove more useful, particularly with regards to newly allocated entries in the TAGE tables; if that is the case altPred is chosen. Furthermore a 4-bit counter keeps track of whether or not the alternate prediction is better; if this counter is above 8 altPred is chosen. This counter is updated whenever a new allocation to the TAGE tables is made and increments based on whether or not altPred is correct.

E. Update Policy

Our update policy largely follows the update policy outlined in the TAGE paper, with one major difference outlined in this section.

The useful counter is updated when primePred and altPred is not equal; if primePred is correct its usefulness counter is incremented, and if it is incorrect its usefulness counter is decremented. Next, absed on the result of the actual correct direction, the prediction counter becomes more strongly or weakly taken.

Next, the TAGE predictor will attempt to allocate a new entry if the prediction is incorrect and came from a table that was not indexed based on the longest history. In particular, it will attempt to allocate in a table based on a longer history than the one that the prediction came from. If a suitable candidate for allocation is not found (i.e. one whose usefulness counter is 0) the usefulness counters of all entires in the tables will be decremented. Otherwise it will attempt to decrement on the table corresponding to the longest history possible. This deviates from the update policy outlined in the TAGE paper; in the paper if multiple tables with available allocation spots were found, the one with the longer history would be chosen with double the probability of the one with shorter history. In our implementation the one indexed based on longer history will always be chosen. Our update policy resulted in generally higher MPKIs; while the randomness introduced by probabilistic allocation sometimes resulted in a lower MPKI compared to a pre-chosen allocation, most of the time the resulting MPKI was higher, and could peak at 4.6 or more. As a result, allocating to the one based on a longer history 100% of the time was done.

F. Reset Policy

Our implementation largely follows the reset policy outlined in the TAGE paper. Periodically, the MSB of the useful counters are reset to 0, which occurs on a clock tick occuring every certain number of instructions. Then, the LSB of the useful counters are reset on the next clock tick. In the original paper, the reset occured every 256K instructions; our implementation conducts a reset every 512K instructions instead.

III. Tradeoffs

Notably, our predictor utilizes 2^{13} entries per TAGE table. This doubles the size of each table, but provided a notable 0.2 increase in MPKI that allowed our predictor to be strongly competitive in the in-class competition. According to the TAGE paper, the change in update policy introduces the ping-pong phenomenon which can lead to higher hardware utilization. However, hardware implementation concerns were not considered for the sake of this assignment; decreasing MPKI was the sole factor in this decision to change the update policy.

IV. Storage Justification

The Bimodal table consisted of 2^{14} entries with 2 bits per entry for a total of 32KB. The Global History Register consisted of 131 bits and the Path History Register of 16 bits. The counter to keep track of whether or not the alternate prediction is better is 4 bits. The 4 TAGE tables each had 2^{13} entries of 14 bits each, totaling 458KB of storage. Thus the total space taken up is slightly under 492KB of storage, well under the implied 1MB limit for storage in this assignment.

V. Additional Notes

The MPKI result on the leaderboard of 4.134 were obtained by running the code on a 13-inch MacBook Pro (Late 2013) with an Intel Core i5-4258U and 8GB of RAM on macOS Sequoia using the clang 16.0.0 compiler. Noticeably, this deviates from the MPKI of 4.17 obtained on a Thinkpad T14s Gen 1 (AMD) with an AMD Ryzen 7 Pro 4750U and 16GB of RAM on Ubuntu 24.04 LTS using the gcc 13.3.0 compiler. We cannot find an explanation for this difference other than something inherent to either machine, the choice of compiler or OS due to the fact that the code was shared using a GitHub repo, and any differences in the code would cause merge conflicts. Depending on the choice of evaluation machine, results may differ.

References

[1] Seznec, André & Michaud, Pierre. (2006). A case for (partially) TAgged GEometric history length branch prediction. Journal of Instruction-level Parallelism - JILP. 8.

[2] Michaud, Pierre. (2005). A PPM-like, tag-based branch predictor - JILP. 7.

[3] Seznec, André. (2005). Genesis of the O-GEHL branch predictor - JILP. 7.