Accuracy of Different Branch Predictor Implementations

A. Emis, *Student, UCLA*

I. Gshare Base Implementation

I was tasked with implementing a branch predictor with the goal of attaining the lowest amount of mispredictions as possible. I was given a base implementation, which was a gshare branch predictor. This worked by having a table of 2-bit counters that would determine if we predicted taken or not taken. If this counter had a value of 2 or 3, predict taken. If the counter had a value of 0 or 1, predict not taken. When updating this table, if the actual behavior was taken, the counter would be incremented. If the actual behavior was not taken, the counter would be decremented. This table was indexed by the XOR'd value of the global history and the program counter. This hashing is meant specialize the prediction, so it is dependent on the address and the history, as opposed to just the address. This was a pretty solid start and returned a MPKI of 6.305.

II. Bimodal Predictor

A ghsare predictor is good if the branch's behavior relies more on the global history of the entire program, compared to the local history of that specific branch. I decided to use a bimodal counter to implement local branch prediction. I had two tables: a history table indexed by the program counter's address and a counter table that was indexed by the history of that branch. I followed the implementation instructions from Reference [1]. Once the correct counter is obtained, a similar procedure with the gshare predictor is followed: determining if it is greater than 1 or less than 2. If the branch was actually taken, that counter (specific to the address and history) would be incremented. If the branch was actually not taken, that counter would be decremented. I updated the history similarly to how I did for gshare, but instead of one global history being updated, I updated the history for that one entry in the history table, which corresponds to the address. This returned a MPKI of 6.692.

III. Tournament Predictor

Some branches are more predictable using global history compared to local history, and vice versa. I decided to implement a tournament predictor that would determine which approach to use based on how that branch behaved before. I used a table of counters that was indexed by the branch's address. If the counter at that address was less than 2, use gshare's prediction. If not, use the bimodal prediction. The next step is to update the counter at that address depending on which implementation was correct. If gshare predicted correctly and bimodal predicted incorrectly, the counter would decrement if possible. If bimodal predicted correctly and gshare predicted incorrectly, the counter would increment if possible. This was a better approach than the previous two because it taken into account that different branches have different tendencies for if they follow local history or global history. The tournament predictor approach accounts for this and stores the preferences for lots of different branches. This implementation returned a MPKI of 4.990. While this was my best accuracy yet, there were some tradeoffs. For example, this approach required multiple tables that were large in size. If there were more constraints with the amount of memory allowed to be used, this implementation might not have been valid.

IV. Refining Parameters

At this point, the only other improvement I could think of was refining the parameters by changing the HISTORY_LENGTH and TABLE_BITS globals. They were both set to 15. I knew that HISTORY_LENGTH had to be less than or equal to TABLE_BITS, so I made both of them equal and tested different numbers until I got the smallest number that yielded the best accuracy. This number would be 24, where the MPKI was 4.415. I then tested if increasing the TABLE_BITS steadily. The smallest number that yielded the best accuracy was 28, which returned an MPKI of 4.382. This was the most accurate I was able to get the branch predictor to be.

V. Other Attempts

I wanted to briefly discuss my other attempts before I landed on tournament predictor. First, I just adjusted the parameters of the gshare implementation to see which combination would be most accurate. When HISTORY_LENGTH was 14 and TABLE_BITS was 20, an MPKI of 5.342 was returned. I thought it was interesting that changing just two parameters improved the accuracy by that much. After, I tried implementing a TAGE predictor, using references [2] and [3] as resources. I was able to get the MPKI to a value of 5.954, which is better than the base implementation, but I was not satisfied with that result, so I did more research and tried the tournament predictor instead.

References

1. *B. Childers, University of Pittsburg.* https://people.cs.pitt.edu/~childers/CS2410/slides/lect-branch-prediction.pdf
2. *A. Seznec, "A 256 Kbits L-TAGE branch predictor."* https://www.irisa.fr/caps/people/seznec/L-TAGE.pdf
3. *Sdesai.* https://github.com/2Bor2C/tage/blob/master/predictor.cc