209AS CA1

Branch Prediction - Nathan Portillo, Benjamin Cruz

[1]

***Abstract*—The aim of this paper is to introduce my methodology in creating a system to improve the Average MPKI from ECE 209AS.**

I. Main Method

The main method in trying to improve the MPKI for this assignment focused on trying a hybrid predictor approach, building off of the original code. One of the biggest methods for improvement was incorporating multiple history length tables. An important question in branch prediction is how long the branch history should be?

A long branch history makes it easier to pick branches in difficult scenarios. However, our system's accuracy is reduced for easier scenarios. Therefore, I tried to address this with multiple history tables that feature multiple history lengths. The idea was that the table with the most correct predictions or 'longest history' in the code would therefore be the best table to use for a given branch. Ideally, this would help pick a suitable table for the warming up phase of the predictor and then better out as the history got longer.

This addition gave the biggest improvement in the Average MPKI from the original 6.3 given in the professor's code.

II. Optimization 1

A big issue, even after introducing the tables, was the the initial traces(the warm-ups) were giving too many missed predictions. This was expected. However, to get a lower Average MPKI this had to be improved. So, instead of initializing the predictor with 0's, the predictor was initialized with 2's. In other words, a slightly higher confidence was given to each branch. This resulted in a slight improvement.

The issue with that improvement is that the initialization was uniform (like giving all branches a confidence of 0). So, in an attempt to somewhat randomly distribute the confidence, the function 'targettedBranch' gives a slightly higher confidence to any branch number divisible by 3. This was a somewhat random attempt at improving the score but it yielded positive results, improving the Average MPKI by about .3.

III. Optimization 2

Another optimization that improved the Average MPKI was expanding the confidence counter for a branch being taken (within the update function). By increasing the number from 3 to 5, we give the system a bit more granularity in its confidence. This is also, more than likely, why initializing some values within the predictor with 2 also helped, as we also happened to increase the number of values available to determine its confidence. This change was somewhat successful, reducing the Average MPKI by about .2.

IV. Optimization 3

The final optimizations were 'messing' with the different branch histories per table and number of prediction bits. Of course, these two go hand in hand as one as too many bits without longer history is wasted memory without the historical context and vice versa. Therefore history lengths were chosen to give variety (low to high history lengths) and table bits were picked accordingly to yield the best Average MPKI. This approach was less methodical and more trial and error until the best results were gotten.

A better approach would have been to dynamically code the table bits and history lengths and pick the most optimized values accordingly.

V. Conclusion

Using these three optimization techniques and the main multi-table method, we were able to reduce the Average MPKI from 6.3 to 5.2. The main focus of the approaches were on trying to improve the initial traces as those traces always performed the worst, with the most average missed branch predictions.

These optimizations attempted to use minimal memory compared to the original code. Of course, by introducing multiple tables, history lengths, and increased table bits, the memory consumption increased. However, compared to other approaches like TAGE and Perceptron predictors, memory consumption is less with a somewhat substantial improvement of about 18%.

[1]