Rohan Jhaveri

Reading Assignment 1

Topic: Dynamic Branch Prediction with Perceptrons

Daniel A. Jiminez and Calvin Lin. 2001. **Dynamic Branch Prediction with Perceptrons**. In Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA '01). IEEE Computer Society, Washington, DC, USA.

With the growth in technology, pipelining and instruction level parallelism (ILP) has become one of the prime features of the computer architecture. With the increase in the complexity of software as well as hardware, accuracy and performance of branch predictors has become a very interesting topic of discussion. Traditional branch predictors used two-bit counters, in which branch predictor keeps records of whether branches are taken or not taken. When it encounters a conditional jump that has been seen several times before, then it can base the prediction on the history. The prediction is based on pattern history table (PHT) which has a major disadvantage that its accuracy decreases as the history length increases. To overcome this, the authors introduced a new approach of branch prediction which replace the two-bit counters by a basic neural network called perceptrons which uses global history table (GHT) instead of PHT. The working of a perceptrons is pretty straight forward. It has one input unit which is connected to one output unit via a hidden layer which has multiple vector elements with different weights based on which it makes its prediction. The output of a perceptron is given by

$$y = w_0 + \sum xi.wi$$

The inputs to the perceptrons are either taken (1) or not-taken (-1). After the output y of a perceptron is computed, the next step is to train the perceptron. According to the training algorithm, the weight becomes negative with large magnitude when there is mostly disagreement or it becomes positive with large magnitude when there is mostly agreement.

One disadvantage of this perceptron based branch prediction in this paper is the aliasing problem in which different branches points to the same entry in the weight table. This could be addressed by learning branch tags. This method is called pseudo tagging which is suggested by Seznec. Another way to address this problem is to use a concept called weight caching. In this there is a small weight table which allows fast access time and it is backed up by a large weight cache which is comparatively slower. In this, the predictor performs the calculation of weights using the weight table and compares the weight table tag with the program counters branch tag simultaneously. If it misses i.e. the branch is taken, the weight table requests the weight of the new program counter and sets the corresponding entries to zero. This makes it fast. When there is a hit in the weight cache, the perceptron is returned and overwritten in the weight table. If there is a miss in the weight cache, it uses the weight table contents for prediction and it makes the mispredicted weights zero and hence avoiding aliasing. Also, another way is to perform the weight calculation and disregard or eliminate the branches with smaller weights or use victim caching for more accuracy.

Another problem addressed in this paper is that perceptron based branch predictor works well only if the prediction is linearly separable i.e. all the true predictions can be separated from all the false predictions. This can be a problem when there are large number of branches. It could be solved by partitioning the output dataset. In this, we divide the large dataset into smaller independent sets which could make it easy to linearly separate the perceptron outputs. The more the partitions, more linearly separate dataset could be achieved.