# CS6600: Computer Architecture
# Assignment 3: Design your own Branch Predictor

**Bala Dhinesh - EE19B011**
**Vishnu Varma V - EE19B059**

## Introduction:

The assignment aims to design our branch predictor and benchmark it against current state-of-the-art policies like Gshare, Hashed perceptron, Perceptron, and Bimodal. All the branch predictors are warmed up for 50 million instructions and simulated for 50 million instructions and the corresponding results were used for statistical analysis. A detailed description of our branch predictor is given followed by its advantages and limitations. Then our branch predictor is compared with other existing predictors and the results are plotted.

## Our Branch Predictor:

Our branch predictor is an extension using additional techniques to the gshare algorithm. Gshare branch predictor uses a specific hashing mechanism to maintain and utilize the hash table to predict the future branches based on past experiences, our implementation builds upon the same hashing function but with changing the states based on which the prediction is done and also includes two important ideas of confidence factors for taken and not taken, as well as accounting for always taken or always not taken cases specifically. Instead of the existing 4 states from 0-3 in gshare, the number of states is doubled to 8 states (0-7), the division of the states and the basis for decision-making can be observed in the image below.

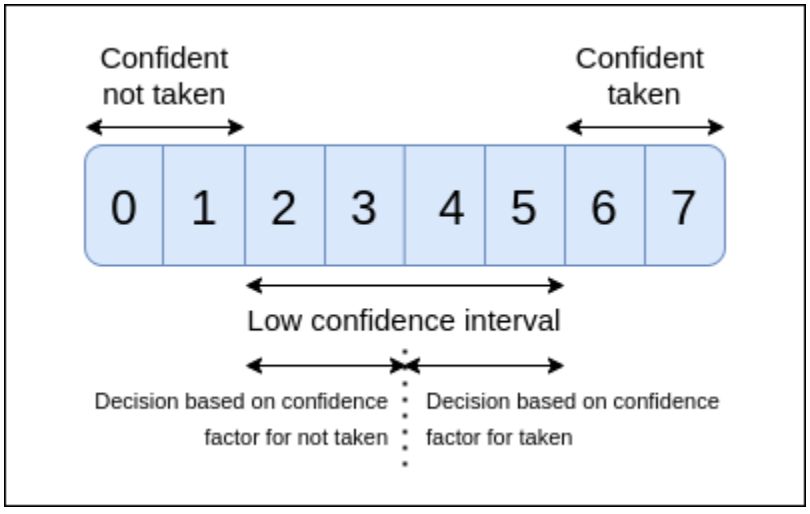


Fig. 1: Correspondence of Hash table values to predictions

The hash table values of 0,1 indicate the prediction to be confidently not taken and similarly, if the value is 6,7 then it is predicted to be confidently taken, but the intermediate region between these two extremes is the low confidence interval which can be taken or not taken. To assert the prediction in this interval we introduce two new variables that keep the track of the accuracy of the previous decisions made and based on that accuracy the decision is taken in this interval. The two variables are cf_t (confidence factor for taken) and cf_nt (confidence factor for not taken) both of which are bounded by a limit specified in our case the limit is given as 10.
The confidence factors are initialized to 0, if a particular branch was predicted to be taken and indeed the branch was taken then the confidence factor for taken is incremented, and similarly for the confidence factor of not taken as well if the branch was predicted to be not taken and indeed the branch was not taken then it is incremented. Suppose the branch was predicted to be taken but actually the branch was not taken then in such cases the confidence factor for taken would be decremented and if the branch was predicted to be not taken but actually the branch was taken then the confidence factor for not taken would be decremented. This is with respect to updating the confidence factors.
The prediction in the low confidence interval is done using the above-mentioned confidence factors. If the Hash table value is 4,5 then it is slightly taken but to decide that the cf_t value is considered if it is greater than half the limit then it predicts that the branch to be taken else not taken, Similarly when the Hash table value is 2,3 cf_nt is considered and if it is greater than half of the limit then it is predicted as not taken else taken.

In addition to the above updations with respect to confidence factors, an additional variable is maintained to keep the track of always taken or always not taken cases - (al_t) varies from + threshold to -threshold. If it reaches +thereshold then the prediction is taken if it reaches -threshold then the prediction is not taken. Setting a reasonable threshold value and in case the branch is continuously taken or not taken for a threshold number of times then the branch is predicted to be taken or not taken respectively based on the above speculation. The setting to track the cases for always taken or not taken is appended along with the previous technique to obtain the final branch predictor.

**Advantages:**

- Provides a two-level abstraction for updating the hash table values using the confidence factors.
- The hash table is updated only if we are confident enough that our predictions are going in the correct direction.
- This method removes the ambiguity in decision-making due to outliers as they change the confidence factors slightly but keep the hash table values safe and since the outliers appear sparsely the change in confidence factor shall not largely impact the prediction.
- Our model is more cautious than the normal gshare, it double-checks with the confidence factor along with the hash table value to make decisions.
- Always taken or not taken cases can be dealt with easily by using the above-mentioned methodology.
- The parameters for confidence factors limit and the threshold for always taken or not taken can be tweaked based on the requirements.

**Limitations:**

- If the traces are such that for most of the time the hash table value signifies extreme values then our branch predictor would deduce to a gshare branch predictor and the usage of the confidence factors will be hindered.
- As our predictor is cautious in shifting the directions of prediction between taken or not taken it maintains some friction in this change by keeping confidence factors in the picture, in situations where the branch is taken or not taken continuously changes for longer durations this friction might hinder the performance slightly as compared to the normal gshare, but this is a tradeoff and this is one possible scenario, If the change in the decision is actually a misdirection then our predictor would perform better by being slightly stringent to the change.

## Comparison of our branch predictor with existing branch predictors:

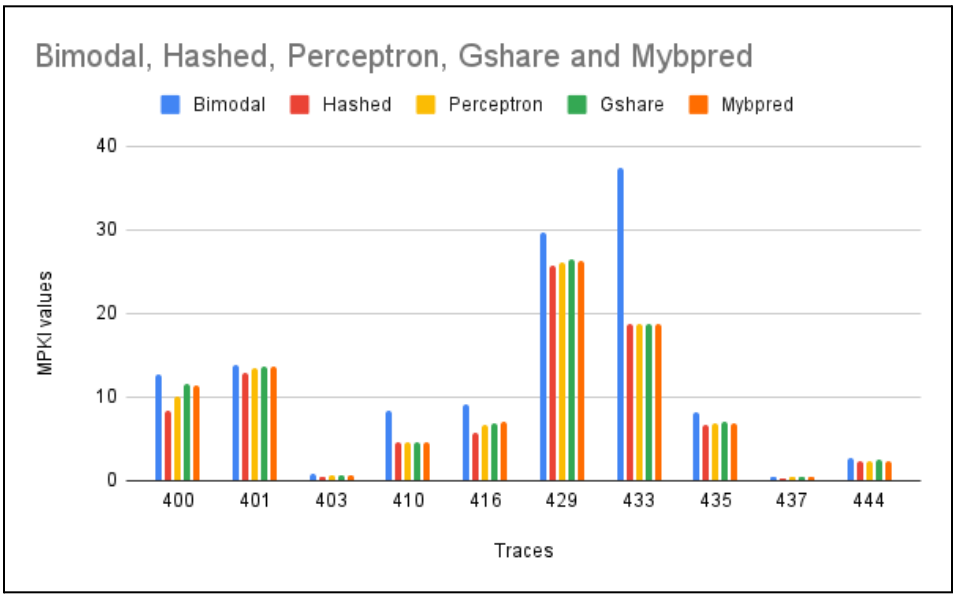


Fig. 2: Comparison of MPKI values of various branch predictors

Based on the above comparison of our branch predictor with other state-of-the-art branch predictors it can be noticed that our branch predictor is performing on par with the existing predictors and it is even better than some predictors like Bimodal, Perceptron and Hashed Perceptron seem to be performing better than others for the given traces, our predictor approaches the performance of these predictors as well for some of the cases namely 433, 437, 444 traces. A detailed comparison of our branch predictor with Gshare can be seen in the below figure, as the hashing mechanism is taken from Gshare comparing the improvement in performance with it would be more meaningful.

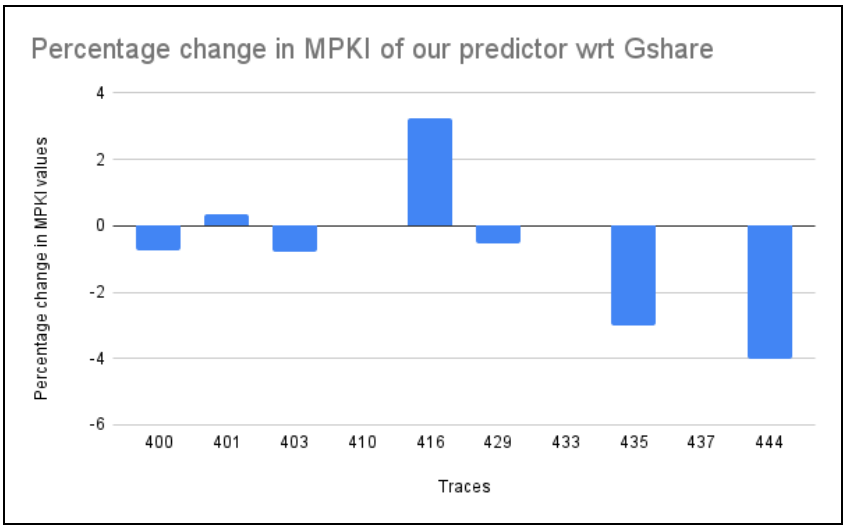


Fig. 3: Percentage change in MPKI of our predictor wrt Gshare

It can be seen from the figure that for all the traces except 416, and 401 the MPKI value for our branch predictor is lesser than that of gshare. As the MPKI value signifies the mispredictions and a decrement in this value for our predictor signifies that our branch predictor is better in comparison to the baseline that we had started with. The percentage change is a positive 3% for only one of the traces 416 and a marginal increment of 0.3% for 401 trace but it has remained the same or decremented as much as 3% for 437 trace, 4% for 444 trace. This implies that our branch predictor overall is performing better than the existing gshare branch predictor for the given traces.

## Conclusion:

- Our branch predictor introduces the low confidence interval states and confidence factors for decision-making in this interval and increases the number of states employed in the hash table methodology.
- Along with it always taken or not taken variable is maintained to keep track of such cases and predict based on continuously taken or not taken branches.
- Our branch predictor performs on par with the existing branch predictors and approaches the high-performing hashed perceptron predictor in some cases while performing better than the bimodal predictor.
- As the baseline is the gshare predictor, upon comparative analysis it showed that our predictor overall performs better than the gshare predictor for the given traces. With the exception of a couple of traces for the remaining ones it either matches the performance or outperforms the existing gshare predictor.

Statistical data:
https://docs.google.com/spreadsheets/d/1ar3cuoQlvGE7-UQOq9bdBpv6Jz_YkrzESNwmPCHk0HA/edit?usp=sharing