ECE 511 Assignment 2

Andrew Smith

February 14^{th}, 2019

1 Introduction

The goal of Assignment 2 was to implement and compare multiple novel branch predictions schemes. I implemented three new branch predictors in the gem5 simulator and compared these new branch predictors to the common branch predictors already packaged with gem5 to evaluate their performance. I added GShare[1], Yags[1], and Perceptron[2] modules to the simulator. To highlight the effectiveness of each predictor, they are tested in a single CPU with the default values of L1 and L2 cache.

2 Implementation

2.1 GShare Branch Predictor

The GShare Branch Predictor uses a simple pattern history table to provide a prediction. The pattern history table is indexed by the xor of the branch address and the history register. This branch predictor uses very minimal hardware resources and is easy to implement. It also performs very well even beating out BiMode and Yags in some instances (Figure 5). The minimal hardware means that there is low latency to arrive at a prediction. An implementation is shown below in Figure 1.

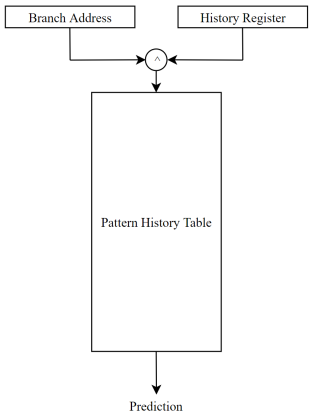

Figure 1: GShare Branch Predictor

2.2 Yags Branch Predictor

The Yags Branch Predictor is very similar to the BiModal branch predictor. It allows a check for an exceptional case where the branch in question might not agree with the choice of the choice history table. The Yags branch predictor utilizes caches to store the taken and not taken predictors. This allows for less aliasing when determining if there is an exceptional case at the branch in question. If there is no hit in any othe the caches, then the choice of the choice history table is use to predict the branch (Figure 2).

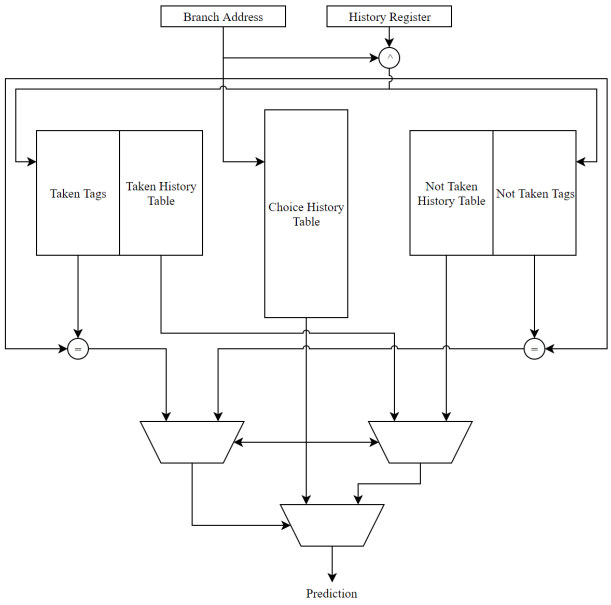

Figure 2: Yags Branch Predictor

2.3 Perceptron Branch Predictor

The perceptron branch predictor is the most clever of all the branch predictors. It uses an array of perceptrons indexed by the branch address to make a classification based on the history register. Then the result is fed back into the training logic and the perceptron is trained. This branch prediction design is incredibly accurate for how simple it is (Figure 5, 4). An implementation is shown below in Figure 3.

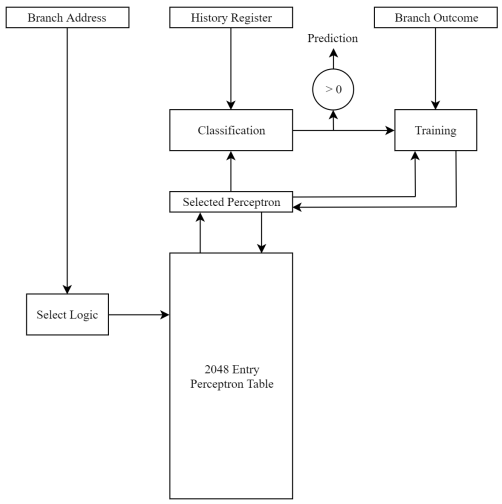

Figure 3: Perceptron Branch Predictor

3 Results

To test the performance of the branch predictors, they were tested on a single CPU arm core with 16kB L1I, 64kB L1D, and 256kB L2 cache. I ran two benchmarks, the Libquantum benchmark which simulates a quantum computer and the Soplex benchmark which uses Simplex Algorithm to solve a linear program.

The parameters of each of the branch predictors were also varied to explore the performance space of the individual predictors.

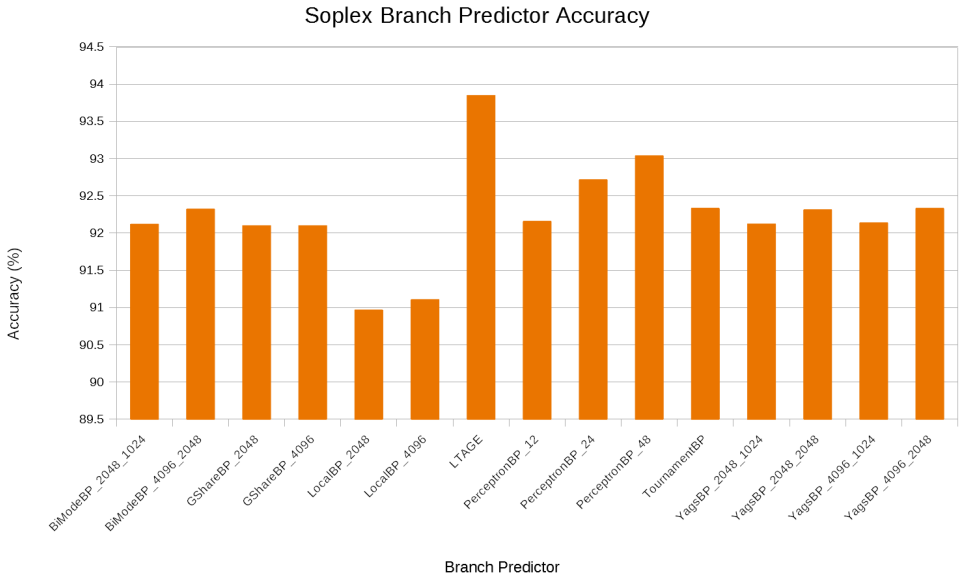

Figure 4: Accuracy of branch predictors for Soplex benchmark

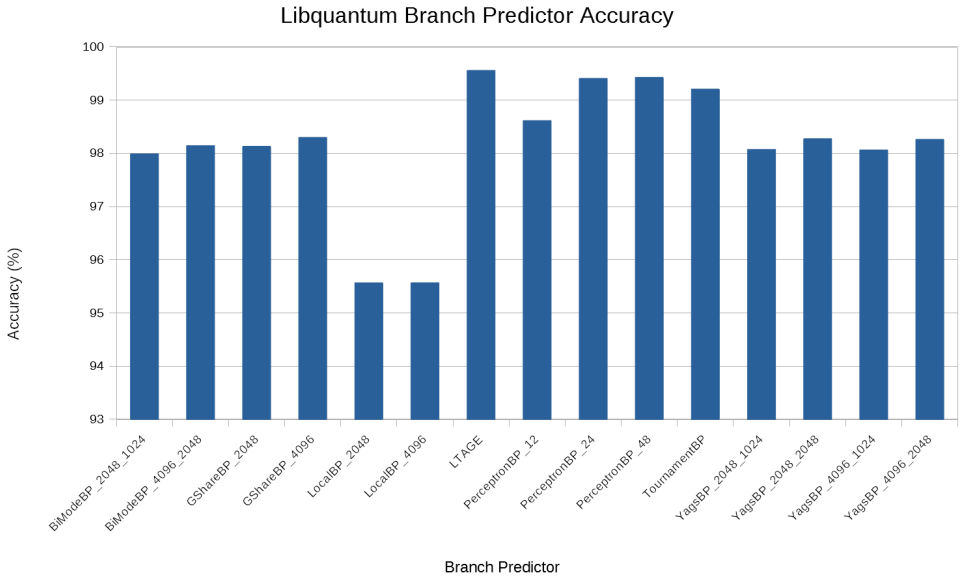

Figure 5: Accuracy of branch predictors for Libquantum benchmark

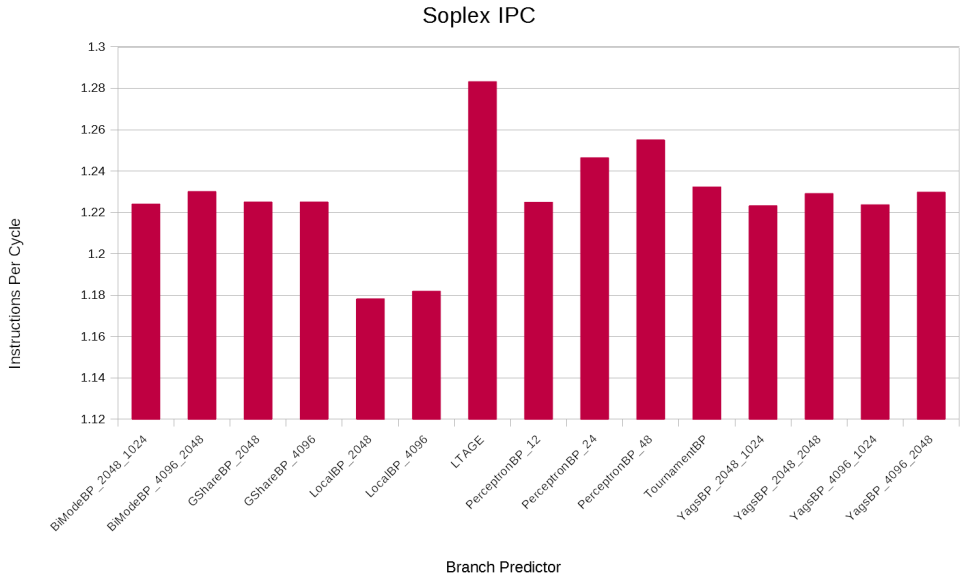

Figure 6: Instructions Per Cycle on the Soplex benchmark

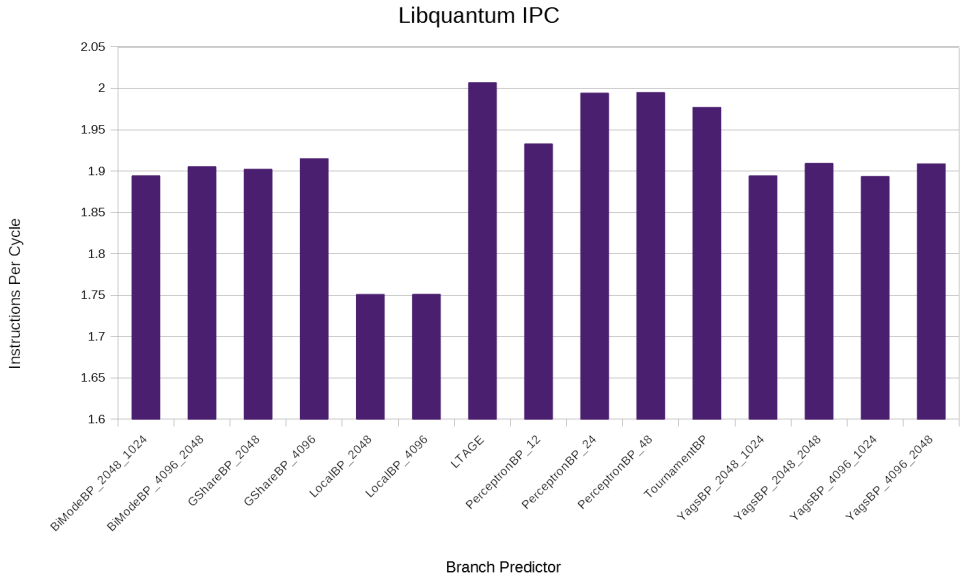

Figure 7: Instructions Per Cycle on the Libquantum benchmark

Overall the branch predictors performed better in the Libquantum benchmark than they did in the Soplex benchmark (Figures 4, 5). This could be due to the Soplex test having very hard to predict irregular branches as it uses the Simplex Algorithm to solve a linear program. Of all the configurations of the branch predictors tested the LTAGE branch predictor was the best performing across both benchmarks followed by the Perceptron branch predictor. The Perceptron branch predictor was the only branch predictor I tested that saw drastic improvements from adjusting parameters. Increasing the width of the weight vector and the history register the performance improved by nearly 1% in the Soplex benchmark (Table 8). Increasing the width from 24 to 48 bits only marginally increased the accuracy of the perceptron in the Libquantum

benchmark. If the Libquantum benchmark has a very predictable branching pattern, then there would be diminishing returns for increasing the width of the history register as it would be holding many taken (or not taken) predictions which wouldn't offer much more new information over what the 24 bit history register could provide. Other branch prediction schemes only marginally benefit from adjusting the size of the predictor tables (Tables 9, 10, 11).

Perceptron		
	Accuracy Soplex	Accuracy Libquantum
PerceptronBP_12	92.161	98.616
PerceptronBP_24	92.719	99.410
PerceptronBP_48	93.040	99.430

Figure 8: Perceptron Accuracy

Gshare		
	Accuracy Soplex	Accuracy Libquantum
GshareBP_2048	92.100	98.134
GshareBP_4096	92.100	98.301

Figure 9: GShare Accuracy

Yags		
	Accuracy Soplex	Accuracy Libquantum
YagsBP_2048_1024	92.124	98.075
YagsBP_2048_2048	92.316	98.277
YagsBP_4096_1024	92.140	98.065
YagsBP_4096_2048	92.335	98.264

Figure 10: Yags Accuracy

BiMode		
	Accuracy Soplex	Accuracy Libquantum
BiModeBP_2048_1024	92.121	97.992
BiModeBP_4096_2048	92.324	98.146

Figure 11: BiMode Accuracy

4 Conclusion

Of all of the branch predictors that were profiled for this assignment the Preceptron branch predictor is the most clever. It uses economical hardware and can achieve better performance than most of the complex designs tested (Yags, BiModal, and Tournament) while being able to adapt adapt to a specific application over time allowing it to perform well in a wide variety of applications.

5 Resources

[1] A. N. Eden and T. Mudge, "The YAGS Branch Prediction Scheme," Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture, 1998.
[2] D. a. Jimenez and C. Lin, "Dynamic Branch Prediction with Perceptrons," HPCA, p. 197, 2001.