Nicholas Kory
SID 862189331
CS 203: Advanced Computer Architecture
March 16, 2020

Final Project Report

Introduction

For my final project, I extended the BranchPrediction lab into a tournament predictor. The iterative design steps can be reviewed on my repository for CS 203 at https://github.com/nmkory/cs203/tree/master/TournamentBranchPrediction. The tournament predictor design was based on the following diagram and on discussions with Dr. Wong in office hours:

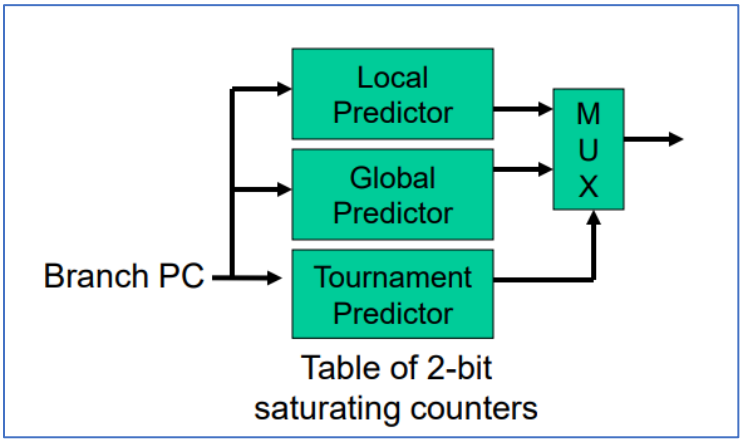

Figure 1: Tournament predictor design.
Credit: https://my.eng.utah.edu/~cs6810/pres/10-6810-08.pdf

Implementation

The implementation was completed as follows:

- Adding a local branch predictor to the implementation. This was done by adding a new class to the original BranchPrediction lab files. The design was based on the following slide from lecture:

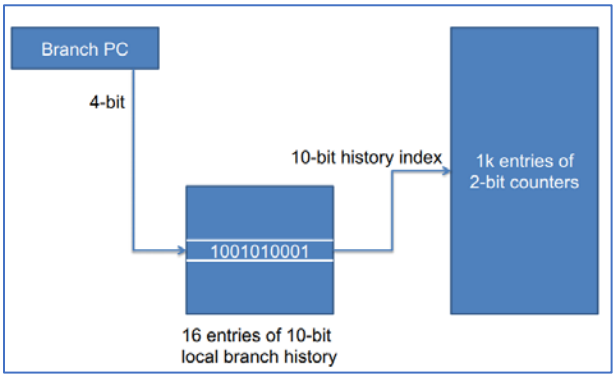

Figure 2: Local predictor design.

The local branch history utilized an array of virtual shift registers that tracked whether the most recent prediction was correct or incorrect. The history table contained indexed entries of 2-bit saturating counters. Each finite state machine utilized the same implementation as the global predictor.

- Adding an additional finite state machine to decide whether to use the local or global branch. To determine which predictor to use, I added a 2-bit saturating counter that changes states based on whether predictors were right or wrong (respectively). The states change based on the following diagram from lecture:

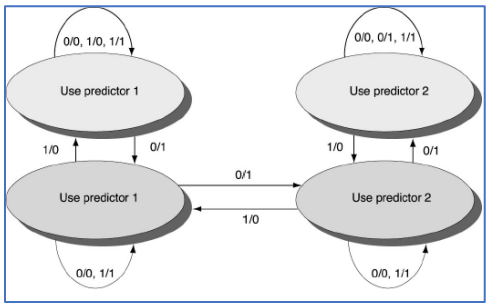

Figure 3: How to pick between the local or global predictor.

- Coordinating the predictors to all run and update simultaneously for each instruction. This is virtualized (I did not implement concurrent operations).

Results

After completing the implementations, I ran multiple tests to find the best hyperparameters for the tournament predictor (holding constant the original hyperparameters for the global predictor from the BranchPrediction lab). After optimization, the misprediction rate of the given traces and predictors from the BranchPrediction lab are as follows:

	Trace	(0,1)	(0,2)	(6,1)	(6,2)
Global only	gcc-10K.txt	30.58%	28.74%	20.29%	26.15%
	gcc-8M.txt	31.23%	28.17%	8.59%	7.31%
Local only	gcc-10K.txt	35.59%			
	gcc-8M.txt	36.6147%			
Tournament	gcc-10K.txt	31.71%	30.25%	20.43%	**24.29%**
	gcc-8M.txt	32.4393%	29.8563%	9.08713%	7.70618%

Figure 4: Misprediction rates of the given traces with local predictor size 128, no XOR gate with the local predictor, and the tournament counter starting saturation set to Strong Local.

The tournament predictor outperformed the local predictor in all cases, consistent with the findings presented in lecture. When compared with the BranchPrediction lab results, the tournament predictor outperformed the (6,2) global predictor for the gcc-10K.txt trace. The tournament predictor fell short in all other cases. This is likely because the trace given for the BranchPrediction lab was designed to produce results with a global predictor implementation.

Additional testing

To determine the hyperparameters, I ran different variations of the tournament predictor and compared their results. The first variation tested whether using a virtual XOR gate with the shift register selected by the current PC would yield lower mispredictions. This was outlined in the following slide from lecture:

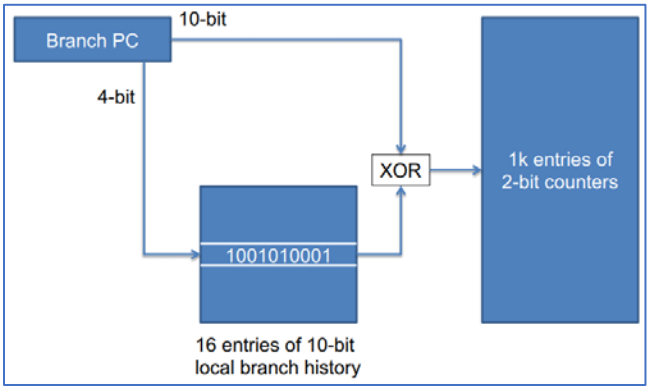

Figure 5: Local predictor design with XOR gate.

The results were as follows (using local predictor size 128, tournament counter starting saturation set to Strong Local):

	Trace	(0,1)	(0,2)	(6,1)	(6,2)
Tournament	gcc-10K.txt	31.71%	30.25%	20.43%	24.29%
w/out XOR Gate	gcc-8M.txt	32.4393%	29.8563%	9.08713%	7.70618%
Tournament w	gcc-10K.txt	32.42%	30.24%	21.09%	26.12%
XOR Gate	gcc-8M.txt	32.871%	30.0992%	**9.07575%**	**7.6984%**

Figure 6: Comparing misprediction rates of the given traces for local predictor with and without XOR gate.

From the results, we can see that the (6,1) and (6,2) global predictor with the XOR gate gcc-8M.txt traces did better than the other cases. The improvement was minimal though, and I chose not to go with the XOR gate because of the respective benefit observed in the XOR-less gcc-10K.txt trace.

The second variation tested the different tournament counter starting saturations. The results were as follows (using local predictor size 128, no XOR gate):

	Trace	(0,1)	(0,2)	(6,1)	(6,2)
Tournament start	gcc-10K.txt	31.72%	30.26%	20.44%	24.3%
strong global	gcc-8M.txt	32.4393%	29.8563%	9.08714%	7.70619%
Tournament start	gcc-10K.txt	31.72%	30.26%	20.44%	24.3%
weak global	gcc-8M.txt	32.4393%	29.8563%	9.08714%	7.70619%
Tournament start	gcc-10K.txt	31.72%	30.26%	20.44%	24.3%
weak local	gcc-8M.txt	32.4393%	29.8563%	9.08714%	7.70619%
Tournament start	gcc-10K.txt	31.71%	30.25%	20.43%	24.29%
strong local	gcc-8M.txt	32.4393%	29.8563%	9.08713%	7.70618%

Figure 7: Comparing misprediction rates of the given traces for different tournament counter starting saturations.

From the results, we see slightly better performance overall when the tournament counter starting saturation is set to Strong Local.

The last variation tested was the local predictor size. It was noted in lecture that there are observable changes in accuracy based on adjustments to the size:

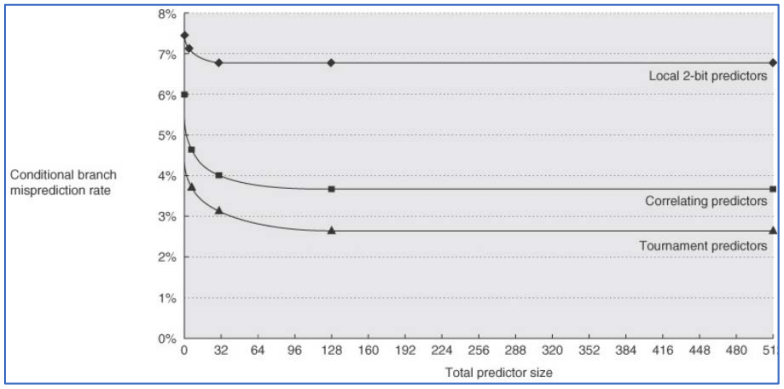

Figure 8: Predictor accuracy by local predictor size.

The results were as follows (using no XOR gate, the tournament counter starting saturation set to Strong Local):

	Trace	32	64	128	256
Tournament (6,2)	gcc-10K.txt	25.82%	25.52%	24.29%	24.88%
	gcc-8M.txt	**7.70234%**	7.67952%	7.70618%	7.72512%

We see that size 128 generally performed better overall (with a slight improvement observed with size 32 in the gcc-8M.txt trace run).

Future work

If I were to extend this project even further, there are several updates I would make:

- Implement concurrent operations to create a more realistic simulation. The local and global predictors are designed to make their predictions in parallel with one another, so using multiple threads to make those calculations would be a prudent update.

- Design a testing harness to automate testing for the hyperparameters. I manually ran all the tests to find the optimal implementation. A testing harness would allow for quicker testing with a wider range of values.

- Run different traces. I used the traces from the BranchPrediction lab to highlight my work for the final project. Different traces would allow for more nuanced testing and a wider range of results for comparisons.