Cameron Calv and Nicholas Sica
ECEC 621 High-Performance Computer Architecture
Project Two – Implementing a perceptron-based branch predictor and evaluating its performance using ML workloads.

Implementing the Perceptron Predictor

In describing the perceptron, it is best to run through how it executes within our implementation. The perceptron algorithm uses three hardware resources, namely the global history table, the global counters and of course the perceptron itself. The size for each piece of hardware in bits is decided by the user and such values are referred to as the configuration values for the duration of the report. The global counter values act as states of a finite state machine representing an outcome choice of the perceptron prediction. The highest bit of the counter determines whether the outcome is taken or not taken for that counter. The number of counter bits represents the maximum value of the counter since these counters have a saturation point. The perceptron itself has a particular threshold value that determines how accurate the change in algorithm will be as it trains itself. The counters act as the input to the perceptron algorithm and its prediction is based on those counter values. The number of perceptrons is of course equal to the perceptron size.

The perceptron is a learning algorithm that goes through a series of steps to predict the direction of branch instructions (either Taken or Not Taken). When an instruction is passed to the predictor, the first step is to obtain what the global prediction is for that particular instruction. This is a result of whether the instruction outcomes so far match an entry already in the global history table. Depending on whether the instruction is taken or not taken, the counter that corresponds to the instruction's address is updated either incrementing if the branch is taken or decrementing if the branch is not taken. Based on the counter inputs the perceptron computes a value corresponding to a positive or negative value. A positive value corresponds to a taken prediction and a negative value corresponds to a not taken prediction. The perceptron is a two-stage process that first takes the sum of the counters inputs and then steps through the values adjusting internal weights that change based on what inputs it processes over time. The final step in the process is training the perceptrons with the input values and the accuracy of its prediction to allow it to adjust its internal weights to attempt to make a better prediction next time. Each step of the process outlined above has its own function present in our implementation that computes the various values of the branch prediction algorithm.

Evaluating Perceptron Predictor

We evaluated the performance of the perceptron predictor algorithm my playing around with the configuration values and determining which settings would produce the highest accuracy. We then used these parameters to test the predictor on the ML workloads in order to compare them to the predictor algorithms of project one.

Table 1: Perceptron algorithm accuracy when tested on the 531 Deepsjeng workload with various configuration properties.

Global PredictorSize	Global CounterBits	Perceptron PredictorSize	Correctness
1024	12	2048	83.567001%
2048	12	2048	83.733139%
4096	12	2048	83.859222%
8192	12	2048	84.137253%
16384	12	2048	84.265701%
32768	12	2048	84.324341%
8192	12	4096	84.949265%
8192	22	4096	83.939293%
8192	28	4096	83.902840%
8192	32	4096	83.621582%
8192	62	4096	83.887680%
8192	12	8192	85.563461%
8192	12	16384	85.890808%
8192	12	32768	86.004433%
32768	12	32768	86.162704%
65536	12	65536	86.213913%
131072	12	131072	86.224182%

Upon attempting to select the optimal values for configuring the perceptron, a few observations were noticed. When run on the ML Workload 531 Deepsjeng the accuracy of the program seemed to decrease when increasing the number of global counter bits until it hit a value of 32, after which the accuracy began to increase again up to 62 bits. It seemed that the optimal value for the ML workloads was 12 bits for the global counter. Either increasing the global predictor size or the perceptron size correlated to an increase in accuracy with the algorithm. For this reason, we tested the algorithm on the workloads using the configuration values highlighted in green above.

Comparing to Two-Bit Local, Tournament, and gShare Predictors

Table 2: Branch prediction accuracy for each predictor and each workload.

	Sample Traces	531 Deepsjeng	541 Leela	548 Exchange2
Two-Bit Local	84.759995%	86.989662%	83.000610%	82.570908%
Tournament	82.290001%	90.746185%	83.784431%	94.983772%
gShare	75.320000%	85.289474%	77.581154%	91.617897%
Perceptron	84.010002%	86.224182%	83.330025%	82.503082%

In our case, it seemed that perceptron algorithm does not outperform the other predictors. This may be because of our particular implementation of the algorithm or it may be due to the configuration values of the perceptron not being as optimal as it could be for each machine learning algorithm. It is worth noting that the perceptron came very close the leading predictor for at least the sample traces and ML load 541.