Brent Lee

Kunal Sharma

ECEC 621

Project 2

2 Bit Local Predictor Results

localPredictorSize	localCounterBits	Deepsjeng Accuracy	Leela Accuracy	Exchange2 Accuracy
2048	1	82.20	78.01	72.23
2048	2	85.87	82.69	82.54
4096	2	86.65	82.81	82.56
8192	2	86.99	83.00	82.57
16384	2	87.06	83.01	82.58
32768	2	87.06	83.01	82.58
65536	2	87.06	83.01	82.58

Tournament Predictor Results

Local History Table Size	Global Predictor Size	Choice Predictor Size	Deepsjeng Accuracy	Leela Accuracy	Exchange2 Accuracy
2048	8192	8192	91.21	84.41	95.45
4096	8192	8192	91.50	84.55	95.47
4096	16384	16384	92.25	85.24	95.68

gShare Predictor Results

Predictor Size	Deepsjeng Accuracy	Leela Accuracy	Exchange2 Accuracy
8192	87.70	79.73	92.82
65536	92.41	81.57	95.01
524288	94.23	87.61	96.28

Perceptron Predictor Results

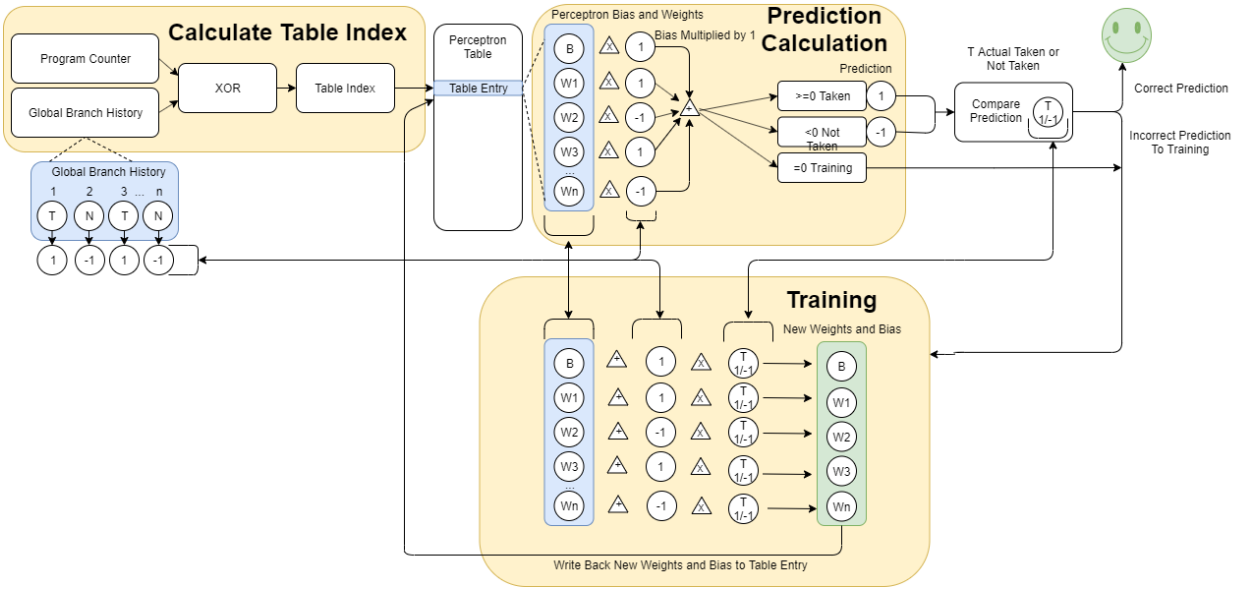
Perceptron History Length	Deepsjeng Accuracy	Leela Accuracy	Exchange2 Accuracy
19	94.33	87.18	95.80
39	94.96	89.01	97.09
62	95.15	89.41	97.84

3. Compare your branch predictor against the two-bit local, tournament and gShare predictor. Does the perceptron-based branch predictor out-perform them?

The paper provided suggested that history lengths 12 to 62 would work best, so history lengths of 19, 39, and 62 were tested. The perceptron predictor with a history length of 62 yielded the highest overall accuracy of any branch prediction method. Even with a history table length of 19, the smallest history length tested, produced accuracies higher than almost every other configuration. The sole exception was the gShare predictor using a predictor size of 524,288 for the Exchange2 cpu trace. However, that gShare configuration has a significantly higher hardware cost compared to the perceptron.

1. Report how you design the perceptron-based branch predictor with a diagram and pseudo-code.

Diagram of the perceptron-based branch predictor. A PDF copy of the diagram is included with the submission. It looks a lot nicer and more readable as a PDF and Not Taken is fully in the white box.



Pseudo-code of the perceptron-based branch predictor

```
//Calculate table address and get weights from table
perceptron_table_addr = branch_addr XOR global_history_register
weights = perceptron_table[perceptron_table_addr]

//calulate sum
//dot product and sum weights and global history
sum = 0
for I = 0 to size(weights):
   //BIAS weight
   if I == 0:
      sum += weights[i] * 1
   //Not BIAS weight
    else:
      sum += weights[i] * global_history_register[i]
   end
end

//Sum <0 Not Taken, Sum >=0 Taken
//Not Taken
if sum < 0:
   prediction = -1
//Taken
else:
   prediction = 1
end

//Training if prediction is wrong or sum is 0
if !(prediction == Actual_Taken_or_Not_Taken) OR sum == 0
   for I = 0 to size(weights)
      weights[i] += prediction * global_history_register[i]
   end

   //Writeback new weights
   perceptron_table[perceptron_table_addr] = weights
end
```