ECE552 Lab 2 Report

Marko Ciric (1006723967), Raymond Huynh (1007058238)

Microbenchmark

In our microbenchmark, there are three branches that test predictions from the two-level predictor. The first branches if the loop variable i is not divisible by 8 (i % 8 == 0). This results in a branch pattern that repeats every 8 iterations; NT, T, T, T, T, T, T, T. Since only 6 history bits are being used, the predictor will fail to predict the next action once when the past 6 outcomes are T. It cannot distinguish whether the pattern is the 1st - 6th T or 2nd - 7th T.

The second branch occurs every other time (i % 2 != 0), causing a branch sequence of T, NT. Assuming the 2-bit counters are initially set to WNT, this should cause a misprediction every iteration for the 2-bit predictor, but because the two-level predictor has history, it should cause no mispredictions past the first iteration. Finally, the code is contained in a for loop, which counts as a branch, but will not result in any mispredictions except the very first and last iteration.

Upon execution, we get 126,758 mispredictions, or an MPKI of 9.254. Since the main loop occurs 1,000,000 times, we should expect around 1,000,000/8 = 125,000 mispredictions in total, which agrees with our measurement. The microbenchmark was compiled using flag -O0 to reduce any compiler optimizations.

Branch and MPKI Tables

	2bitsat mispredicted branches	2bitsat MPKI	2level mispredicted branches	2level MPKI	openend mispredicted branches	openend MPKI
astar	3695830	24.639	1785464	11.903	930839	6.206
bwaves	1182969	7.886	1071909	7.146	953538	6.357
bzip2	1224967	8.166	1297677	8.651	1170776	7.805
gcc	3161868	21.079	2223671	14.824	786240	5.242
gromacs	1363248	9.088	1122586	7.484	842147	5.614
hmmer	2035080	13.567	2230774	14.872	1718758	11.458
mcf	3657986	24.387	2024172	13.494	1510133	10.068
soplex	1065988	7.107	1022869	6.819	747443	4.983

Open-Ended Perceptron-based Branch Predictor Implementation

Our open-ended branch predictor implementation focuses on the use of perceptrons, effectively creating a single layer neural network that is trained using branch history. A perceptron makes binary decisions based on a set of weights, computed by training the algorithm with a dataset that can be sufficiently linearly separable. In this implementation, the dataset in question is the global branch history. The global history register has *n* entries, storing the decisions of the past *n* branches. Each unique branch PC address indexes a table of *h* perceptron weight vectors, such that each branch has its own set of *n* weights. In the training process, if the predicted outcome does not match the real outcome, the predictor iterates across all *n* indexes of the GHR. If the *i*th value of the GHR does not match the outcome, the *i*th value of the corresponding perceptron vector will be decremented by 1, denoting a decreased likelihood for the decision represented by the *i*th value of the GHR. Otherwise, the *i*th value of the corresponding perceptron vector is incremented by 1, denoting an increased likelihood for that decision. To determine whether a branch is taken, the dot product between the GHR and the perceptron vector of the calling branch PC is calculated, creating a weighted sum. If this sum is greater than 0, the branch should be taken. Otherwise, the branch should not be taken. Therefore, for any branch, the weighted sum represents the likelihood of a particular decision. This training process occurs while the predicted value does not match the actual value or until a threshold value is met for the weighted sum. We determined the values for *h* and *n* to be 256 and 32 bytes respectively, because using an integer size of 2 bytes per entry leads to a size of 32 x 256 x 2 = 16,384 bytes or exactly 16 KB. The implementation algorithm was based on a paper [1] that provides ideal values for *h* and *n*, however we deviated from those values because they did not result in a cache size of 16 KB or less while having an average MPKI of less than 7.5 over the CPB-4 benchmarks. Using trial and error we found an optimal history length and number of table entries such that the average MPKI was 7.22 for the perceptron-based branch predictor and it required a 16 KB cache size.

Area, Access Latency, Leakage Power Analysis

	Two-Level Predictor	Open-End Predictor
Area (mm^2)	0.00193555497564	0.03073042898
Access Latency (ns)	0.164342	0.269009
Leakage Power (mW)	0.366395	6.07147

Both predictors used the pureRAM.cfg configuration file. For the two-level predictor, the size parameter was changed to 1024B. For the open-end predictor, the size parameter was changed to 16384B, block size to 32B, and bus width to 256B.

Statement of Work

Both members contributed to the high-level design and code development of the altered predictor.cc file and mb.c file. The report was written collaboratively.

Works Cited

[1] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, 2001, pp. 197-206, doi: 10.1109/HPCA.2001.903263.