Assignment 1 Report

Shuxin Lin (s1469577)

Internal Structure

1. Static Branch Prediction

The main function receives two arguments: trace file name, mode. If mode is “taken” or “not_taken”, call alwaysPredictor function. If mode is “profile”, call profileGuidedPrediction function. Otherwise, print “Error! Cannot recognize the mode.”

alwaysPredictor:
After opening the file, set up branch variables and mispredicton variables. the program set up an integer “predictor” and initialize it according to the mode. Then, the program reads the trace file one line per time until it ends. Everytime set the branch variable “taken” value by getting out the last bit of one line. And then compare it to the predictor and the correct flag increments if they are equivalent. When the trace file comes to the end, the times of wrong prediction divided by the times of total prediction is the misprediction rate.

profileGuidedPrediction:
This technique need two copies of trace file. First, do the same initializing things as alwaysPredictor. Besides, the program initialize a hash table to connect each address with its own predictor. In the first time to go through the trace file, the program scan each line to get the address and the binary digit signifying whether it has been taken or not. And then store the address into hash table if it did not exist and set up a counter related to the address. If the branch is taken, the counter increments. Otherwise, the counter decrements. When the trace file comes to the end, the hash table construction is finished. If the counter of one certain address is positive, it shows this branch is taken 50% or more of the time so it is predicted as taken. If negative, this branch is taken less than 50% of the time so it is predicted as not taken. In the second time to go through the trace file, the program uses the hash table to predict each branch and compare the prediction with the real result, which help the program get the misprediction rate.

2. Dynamic Branch Prediction: Two-Level Correlating Predictor

The main method receives two arguments: trace file name and number of bits. If the number is 4 or 8 or 12 or 16, call twoLevelPredictor method. Otherwise, print the warning.

twoLevelPredictor:
First, read the file into buffer and set up branch variables and mispredicton variables. Then the program initialize the pattern history table and global history register using arrays. The Global

History Register (GHR) is all not taken (0s); and all 2-bit predictors is weakly not taken (01). Then, the program reads the trace file one line per time until it ends. Everytime one branch is about to predict, get PHT predictor indexed with the global history. For example, the global history is 0010, so get PHT[2]. Compare the predictor with branch data to update misprediction rate. Something to notice here is the predictor uses two-bit binary value. So we need to get the decision bit by getting the integer part of predictor/2. After that, the program updates PHT using switch-case statement. This is essentially a statement machine. In addition, the program shifts the GHR to discard the left-most bit and insert the branch data into the right-most bit. When the trace file comes to the end, the times of wrong prediction divided by the times of total prediction is the misprediction rate.

Result Summary

Static Branch Prediction:
Test 1:
Command:
./StaticBranchPrediction gcc_branch.out taken
Display:
Predictor: always taken
Misprediction rate: 23.61

Test 2:
Command:
./StaticBranchPrediction gcc_branch.out not_taken
Display:
Predictor: always not_taken
Misprediction rate: 76.39

Test 3:
Command:
./StaticBranchPrediction gcc_branch.out profile
Display:
Profile Guided Predictor:
Misprediction rate: 6.63

Test 4:
Command:
./StaticBranchPrediction mcf_branch.out taken
Display:

Predictor: always taken
Misprediction rate: 32.24

Test 5:
Command:
./StaticBranchPrediction mcf_branch.out not_taken
Display:
Predictor: always not_taken
Misprediction rate: 67.76

Test 6:
Command:
./StaticBranchPrediction mcf_branch.out profile
Display:
Profile Guided Predictor:
Misprediction rate: 10.94

Test 7:
Command:
./StaticBranchPrediction mcf_branch.out takenOrNot_taken
Display:
Error! Cannot recognize the mode.

Test 8:
Command:
./StaticBranchPrediction data.txt not_taken
Display:
Error! No file exists.
Test 6:
Command:
./StaticBranchPrediction mcf_branch.out not_taken 21
Display:
Usage: ./StaticBranchPrediction <trace file> <mode>

Conclusion and inference

1. The misprediction rate of always taken or not taken is quite high. For the same trace file, the misprediction rate of always taken plus the misprediction rate of always not taken is equal to 100%. So the high misprediction rate is reasonable and predictable.
2. The misprediction rate of profile guided prediction is much lower compared with the first technique. It works very well when addresses reappear several times.

3. The code works perfect to detect errors.

Two-Level Correlating Predictor

Test 1:
Command:
java TwoLevelPredictor gcc_branch.out 4
Display:
Two-Level Predictor:
Misprediction rate: 19.25

Test 2:
Command:
java TwoLevelPredictor gcc_branch.out 8
Display:
Two-Level Predictor:
Misprediction rate: 11.91

Test 3:
Command:
java TwoLevelPredictor gcc_branch.out 12
Display:
Two-Level Predictor:
Misprediction rate: 5.77

Test 4:
Command:
java TwoLevelPredictor gcc_branch.out 16
Display:
Two-Level Predictor:
Misprediction rate: 5.40

Test 5:
Command:
java TwoLevelPredictor mcf_branch.out 4
Display:
Two-Level Predictor:
Misprediction rate: 16.70

Test 6:
Command:

java TwoLevelPredictor mcf_branch.out 8
Display:
Two-Level Predictor:
Misprediction rate: 14.40

Test 7:
Command:
java TwoLevelPredictor mcf_branch.out 12
Display:
Two-Level Predictor:
Misprediction rate: 14.09

Test 8:
Command:
java TwoLevelPredictor mcf_branch.out 16
Display:
Two-Level Predictor:
Misprediction rate: 14.76

Test 9:
Command:
java TwoLevelPredictor mcf_branch.out 13
Display:
Error! Wrong bits.

Test 10:
Command:
java TwoLevelPredictor mcf_branch.out 12 11
Display:
Usage: java TwoLevelPredictor <trace file> <number of bits>

Conclusion and inference

1. The misprediction rate of two-level correlating predictor is much low compared with static branch prediction. The longer the length of history maintained, the better the technique performs.
2. Although the misprediction rate decreases as the length of GHR becomes longer, the improvement becomes smaller or even worse. So we don't need to set the length of GHR to a large number, it will not improve much but take a lot of space. I think 12 is good enough to balance between both.
3. The mispredicition rate of mcf_branch.out is larger than that of gcc_branch.out. And

gcc_branch.out has more lines than mcf_branch.out. So I think the longer the trace file is, the better the program works. So the two-level correlating is a good choice for branch prediction because system branch trace file is always very long.

4. The code works perfect to detect errors.