Experiment Report

Introduction

This experiment's purpose is to test which of these branch prediction strategies: always taken branch predictor; standard 2-bit branch predictor; gshare branch predictor and profiled branch predictor predict the actual branches taken/not taken situation more accurately. The four branch prediction simulator simulating these four branch prediction strategies were developed, then in order to test their performance, the branch trace files used as benchmark were inputted into those simulators, simulators outputted the misprediction rate, the result shows that the profiled branch prediction simulator have comparatively lowest misprediction rate.

Materials

The BranchTraces.zip file including many dynamic branch trace file; the Branch Prediction Simulator folder including Sim.java file, TwoBitPredictor.java file, AlwaysTaken.java file, Gshare.java file and ProfiledPredictor.java file andPredictor.java file; The README.txt file showing how to run those simulators.

Design

Four simulators have one common argument: the absolute path of the trace file. Besides, for the 2-bit branch predictor and profiled branch predictor, there are one more argument to run the predictors: n, the value of 2^n defines the entries of the branch prediction table is defined. (In profiled branch predictor, the value of n won't influence the misprediction rate, normally, there should be no other argument besides the file path in this predictor, there n is parameter because I attempted to combine profiled algorithm and 2-bit algorithm, but it turned out that in this condition, the misprediction rate is higher than the normal profiled branch predictor, so I deny this strategy.) For the Gshare simulators, there are two more arguments to run it: m and n, the value of 2^m defines the entries of branch prediction table, n is the number of bits in the global branch history register.

Use the chosen benchmarks’ absolute path as the arguments of the branch prediction simulators, respectively input them to the four branch prediction simulators, and keep the other parameters among different simulators same (in practice, I kept n of 2-bit branch predictor to be 8 all the time and kept the m and n of Gshare branch predictor respectively to be 14 and 8 all the time), collect every time’s misprediction rate, put these data into forms to compare different branch prediction simulators’ performance and analyse.

Adjust the value of n and m, keep the inputted file same, run the simulators and collect the misprediction rate to compare and analyse.

Analysis

It can be seen from the following table and figure, in general, for the 2-bit branch predictor, the more entries the BP table has, the lower misprediction rate the predictor causes. For some 2-bit branch predictor, the misprediction rate keep same when the size of BP table entries is changed, it is because there is a periodic and regular changes in the result of taken or not taken of all the branches. For example, in file jpegtran.out, all branches basically follow this regulation: after 8 branches are taken, next branch is not taken. In file linpack.out, taken or not taken of branches are alternative in order.

File Size of entries	4096	2048	1024	512
ack.out	5.81%	5.87%	5.98%	6.34%
bubbleSort500.out	12.33%	12.36%	12.54%	13.21%
coremark.out	9.78%	10.01%	10.97%	11.95%
fib100.out	4.65%	5.31%	6.24%	7.81%
g++.out	6.63%	7.72%	9.25%	11.46%
gpg.out	5.11%	5.11%	5.11%	5.11%
jpegtran.out	10.72%	10.72%	10.72%	10.72%
jpegtran2.out	5.14%	5.18%	5.29%	5.60%
linpack.out	0.59%	0.59%	0.59%	0.59%
LoopCondition.out	8.88%	9.02%	9.34%	10.48%
matrix_mult.out	9.28%	9.40%	9.75%	10.66%

pythonsort.out	5.71%	6.99%	8.27%	10.36%

Table 1. The misprediction rate of 2-Bit branch prediction simulator-form

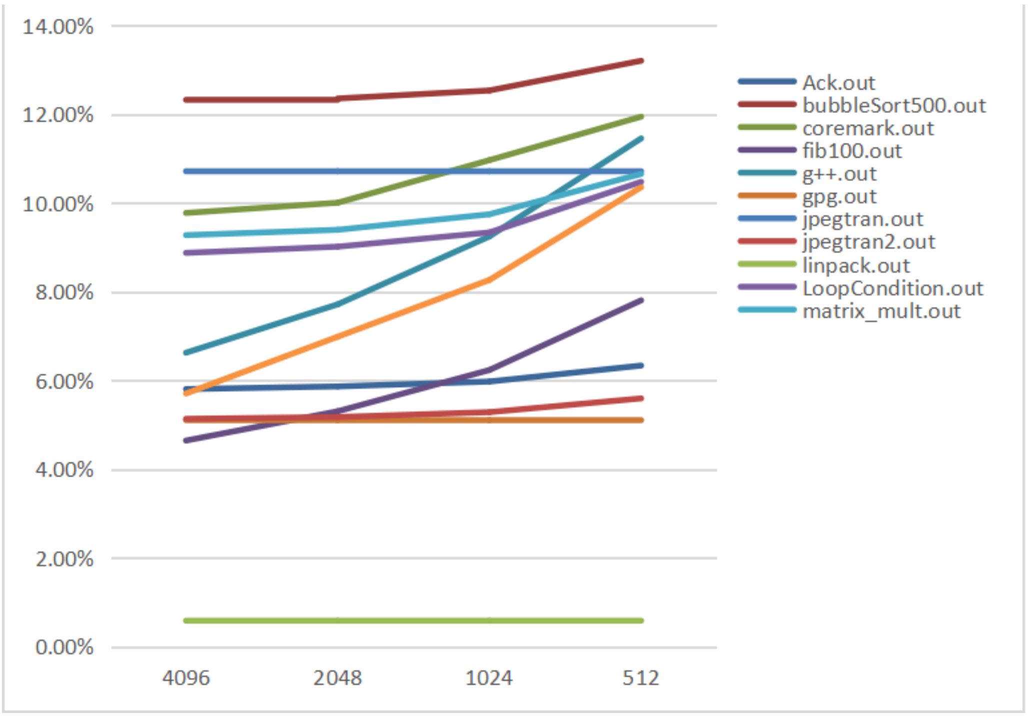

Figure 1. The misprediction rate of 2-Bit branch prediction simulator-graph

As illustrated in the following table and figure, broadly, the sort of predictors in misprediction rate order is: Alwaystaken branch predictor > Standard 2-bit branch predictor > Gshare branch predictor > Profiled branch predictor. (The pythonsort.out file is too huge, so I made the modulus operation using length of the file and number 4 to divide the file into four sub-files and got the first file as the argument.)

For the branches in Echo_hello.out, jpegtran.out, matrix_mult.out and quicksort.out file, the Gshare predictor's misprediction rate is higher than or equals to standard 2-bit predictor's misprediction rate. This situation occurs because the outcome of most recent executed branch doesn't have an apparent influence to the outcome of the next branch.

In general, profiled branch predictor's misprediction rate is much lower than the other predictors' misprediction rate, but for the branches in ack.out, bubbleSort500.out, linpack.out and yes.out, profiled branch predictor’s misprediction rates are not the lowest among the four predictors. The closer the real ratio of number of taken to number of not taken of a branch gets to 1, it means the probability of taken or not taken of this branch is similar, the higher the profiled branch predictor’s misprediction rate will be.

file Branch predictor	Alwaystaken	2-bit	Gshare	Profiled
ack.out	32.90%	6.34%	6.02%	9.59%
bubbleSort500.out	31.10%	13.21%	13.12%	18.33%
curl.out	49.56%	10.62%	4.08%	3.59%
Echo.out	48.84%	10.57%	10.56%	6.34%
Echo_hello.out	49.45%	11.23%	11.67%	6.97%
ffmpegenc.out	51.67%	10.48%	7.46%	7.12%
fib100.out	53.34%	7.81%	7.17%	3.50%
g++.out	38.70%	11.46%	5.95%	5.31%
jpegtran.out	17.93%	10.72%	10.72%	10.72%
jpegtran2.out	47.68%	5.60%	5.47%	4.95%
linpack.out	49.47%	0.59%	0.45%	0.53%
matrix_mult.out	50.94%	10.66%	11.12%	5.94%
pythonsort.out(huge)	56.25%	11.68%	9.22%	5.52%
quicksort.out	47.70%	10.51%	10.91%	6.34%
yes.out	38.32%	6.54%	2.05%	2.44%

Table 2. The misprediction rate of four branch prediction simulators

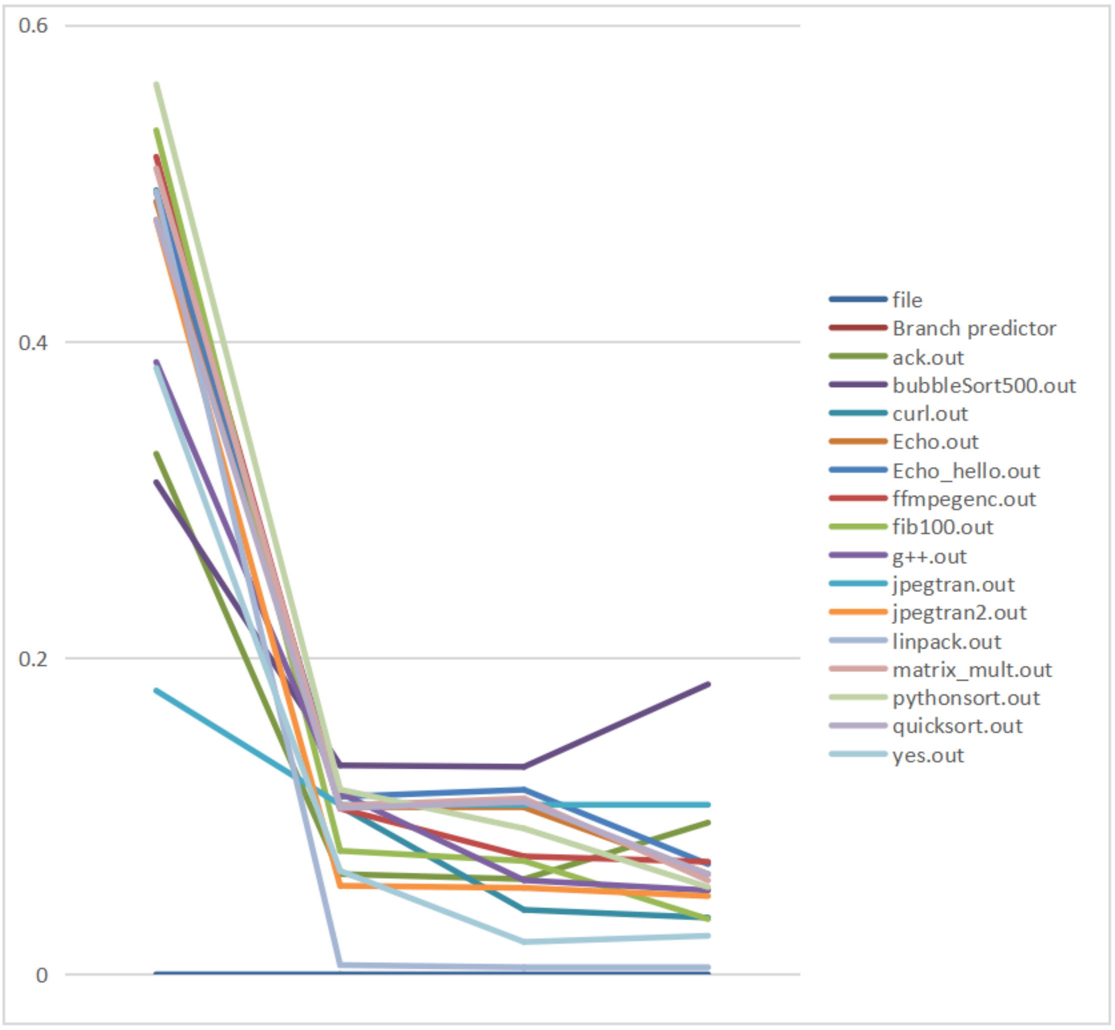

Figure 2. The misprediction rate of 2-Bit branch prediction simulator

Reference

Lawrence Oks. (2021) *branch-predictor-simulator. Available from:* https://github.com/ljoks/branch-predictor-simulator