Gem5 Lab 2: Dynamic Branch Prediction

Nov 17, 2020

Due: This lab is to be completed during your time slots. After you are done, submit everything to github classroom.

The purpose of this assignment is to supplement your knowledge about the design of different dynamic branch prediction schemes, alongside intuition about their relative performance.

Cloud Access

You can connect to the cloud Linux server (**2440.cs.uh.edu**) with any ssh client during your time slot. Alternatively, you can use your own computer with Oracle Virtualbox. Keep in mind in the Linux server that we provide, gem5 is installed under /opt/gem5. In the Oracle Virtualbox disk image, it is installed under /home/gem5/gem5.

Windows ssh client
https://mobaxterm.mobatek.net/download.html
https://www.putty.org/
Mac OS
Mac OS has a built-in SSH client.
Android
https://android.md/2019/05/10/5-best-android-ssh-clients/

The Assignment

The default gem5 implemented dynamic branch predictors below for O3CPU:

- BiModeBP

- TournamentBP

- LocalBP

We have added two additional predictors for purpose of study:

- *GshareBP*

- *NeverTakenBP*

Recall that the GShare predictor takes the global history and XORs some bits of the PC to index into its list of counters.

For this lab assignment, you will run **two benchmark applications** (for instance, **bzip2 and mcf**) under two different prediction strategies. You are free to choose two different benchmark applications.

The three prediction strategies are:

- *GshareBP*

- *BiModeBP*

You can change predictor by modifying **6wayo3.py** in gem5 lab 1. Change the line with "cpu.branchPred". The default predictor is TournamentBP. If you connect to the lab server, you don't have permission to modify files under *"/opt/gem5/bin"*, copy to your local home and make necessary changes. A better option is to use the CPU model Python file from the previous gem5 lab assignment (6way cpu model).

```
...
cpu = system.cpu[0]
# use default derivO3 branch predictor
# cpu.branchPred = LocalBP()

cpu.decodeWidth = ...
```

Execute both benchmarks and record the results (tip: save the ***m5out/stats.txt*** file for each one of the experiment cases - Q4 and Q5 tables).

For Gshare, please use **8192** as PredictorSize. The default value is **2048** (read Lab Assignment Appendix).

As a good tradeoff between representative experimentation and execution time, you may, set your maximum instructions to 100 million. Note that even if you do so, sometimes gem5 reports some instructions short or in excess of the value you set. That is fine as long as your results are consistent.

It may take about 15 minutes to complete simulation of 100 million instructions. Only in case it takes too long to complete (for instance, more than 40 minutes), you can pick a different number. If you use a number different from 100 million, please state it in the lab report.

Make sure that you are using the O3CPU model with the following setting, and keep the machine configuration consistent for all your experiments. Please also set **L2 cache size to 8MB**.

After each simulation, remember to ***make a copy of the results*** ***(stats.txt and config.ini)****.* ***The old files will be overwritten when you start to run the simulator again.***

Answer the following questions in the report. Whenever you are required to report numbers/statistics, the responses must be accompanied by discussions.

General branch frequency:

Q1: How often are branches executed in your benchmarks? Report absolute values and the percentage relative to the total number of instructions.

Name of Benchmark 1 (e.g., bzip2):

bzip2

Name of Benchmark 2:

mcf

	Benchmark 1	**Benchmark 2**
Absolute #	7176276	2800
Relative to the total # of instructions	0.07176276	0.2861

Q2: Are there differences between the frequencies of branches in the two programs, what is your hypothesis to explain it?

Different benchmarks were used, therefore, the number of branches executed were also different. When divided by the total number of instructions executed, we get different frequencies.

In **stats.txt**, one may find statistical result below. Q3 is related to understanding of CPU speculative execution model.

Q3: Why are the two numbers, fetch.Branches and commit.branches, different?

```
system.cpu.branchPred.lookups            9030905
...
system.cpu.commit.branches               5990529           # Number of branches committed
system.cpu.fetch.Branches                9030905           # Number of branches that fetch encountered
...
system.cpu.commit.branchMispredicts                        # The number of times a branch was mispredicted
```

Not all fetched instructions are run by the program. Therefore the branches committed are among those branches that are fetched, and run by the program.

Q4: Prediction Performance:

Report the number of correct predictions for all branch prediction strategies, including what the static schemes (*always predict taken or always not taken*) would have achieved. For the static schemes you do not have to run a simulation, just use the branch characterization results in **stats.txt.** As an option, you may apply **NeverTakenBP**.

	Benchmark 1	**Benchmark 2**
Always not taken (**NeverTakenBP)**	29664840	4746
Always taken	2648983	719
Gshare	10090545	4651
BiMode	9162471	4767

Which prediction scheme has the most correct predictions for each program?

Benchmark 1, bzip2 in my case, always has the most correct predixtions for NeverTakenBP, Always taken, Gshare and BiMode.

Q5: Instruction Fetch Results

Report the number of fetched instructions for each setting (line below in **stats.txt**).

system.cpu.fetch.Insts *153612836*

	Benchmark 1	**Benchmark 2**
Gshare	159002108	25841
BiMode	153087773	26811
Always not taken	355263466	33975

What factors will affect the number of fetched instructions? Will a more accurate branch prediction scheme lead to more fetched instructions (or the other way) given all other CPU configurations are the same (cache sizes, issue width, pipelines)? Why or Why not? Please try to support your answer with data.

Branch prediction affects the number of fetched instruxtoons. A more accurate branch prediction scheme will not lead to more fetched instructions, rather, less instructions will be fetched to be executed. According to the above data, BiMode fetches less instructions than Gshare in benchmark 1.

Q6: CPI

Report CPI for all branch prediction strategies.

	Benchmark 1	**Benchmark 2**
Gshare	1.086314	7.352749
BiMode	1.067813	7.400572

Which prediction scheme has achieved the best performance? Why?

Best performance = Lower CPI = BiMode of benchmark 1, where benchmark 1 = bzip2.

Q7: Overhead

Report sizes of the predictors. You can calculate using data reported in **config.ini**. The file lists various table sizes of the predictor. For instance,

```
choicePredictorSize=8192
globalCtrBits=2
globalPredictorSize=8192
localCtrBits=2
localHistoryTableSize=2048
localPredictorSize=2048
```

	Total # of bits used (show your calculation using data from config.ini)
Gshare	15 (bzip) + 15 (mcf) = 30
BiMode	17(bzip) + 17(mcf) = 34

Which prediction scheme has used the most # of bits?

BiMode uses 17 bits.

Does larger branch prediction scheme always deliver better performance (CPI)?

Larger branch prediction = lower CPI = better performance

Submit

Your submission should consist of the following:

- A report containing the answers for the questions, named as: ***gem5_lab2_bp_report.pdf***
- All the statistics output files generated by the simulator (**stats.txt**)
- All **config.ini** files under m5out

Note that you must submit stats.txt and config.ini files for all your experiments. To make TA's grading job easier, label the files that you have submitted using the table below:

	Name of stats.txt	**Name of config.ini**
Benchmark 1 - Gshare	stats_bzip_gshare.txt	config_bzip_gshare.ini
Benchmark 1 - BiMode	stats_bzip_bimode.txt	config_bzip_bimode.ini
Benchmark 2 - Gshare	stats_mcf_gshare.txt	config_mcf_gshare.ini
Benchmark 2 - BiMode	stats_mcf_bimode.txt	config_mcf_bimode.ini

Command line for submitting files to github

```
-    cd [your github repository location]
-    git add "*"
-    git commit -m "Logisim lab2"
-    git push origin master
```

Appendix: Gem5 Branch Prediction Implementation

The source code of branch predictors can be found underlying "*/opt/gem5/gem5/src/cpu/pred*". These include: *bi_mode.cc, gshare.cc, never_taken.cc*. Note that source code is not provided in the Linux disk image in order to save space of compressed file.

The default predictor setting is in "*BranchPredictor.py*". You can modify predictor parameters in your Python CPU model, for instance choosing a different PredictorSize.

```
class GshareBP(BranchPredictor):
  type = 'GshareBP'
  cxx_class = 'GshareBP'
  cxx_header = "cpu/pred/gshare.hh"

  localPredictorSize = Param.Unsigned(2048, "Size of gshare predictor")
  localCtrBits = Param.Unsigned(2, "Bits per counter")

class BiModeBP(BranchPredictor):
  type = 'BiModeBP'
  cxx_class = 'BiModeBP'
  cxx_header = "cpu/pred/bi_mode.hh"

  globalPredictorSize = Param.Unsigned(8192, "Size of global predictor")
  globalCtrBits = Param.Unsigned(2, "Bits per counter")
  choicePredictorSize = Param.Unsigned(8192, "Size of choice predictor")
  choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

The following two lines in a CPU model specifies Gshare as the branch prediction scheme and the predictor table size with 4096 entries.

```
cpu.branchPred = GshareBP()
cpu.branchPred.localPredictorSize = 4096
```