EE524/CPTS561 Advanced Computer Architecture, Fall 2020

Programming Assignment 2

Due date: Monday, 30 November 2020, 11:59 pm (on Blackboard)

Introduction

Using gem5, in this assignment, we will study the effect of varying Branch Prediction parameters on different benchmarks.

There are 11 kinds of Branch Predictors available in GEM5 (stable). You can find these branch predictor options by running the following command in gem5.

```
$ ./build/X86/gem5.opt ./configs/example/se.py --list-bp-types
```

1. LocalBP
2. BiModeBP
3. TournamentBP
4. MultiperspectivePerceptron8KB
5. TAGE_SC_L_8KB
6. TAGE
7. MultiperspectivePerceptron64KB
8. MultiperspectivePerceptronTAGE8KB
9. TAGE_SC_L_64KB
10. LTAGE
11. MultiperspectivePerceptronTAGE64KB

These files are found in the “$gem5/src/cpu/pred/” folder. We will select three of these branch predictors and assess the effect of misprediction of branches when we change the various parameters (details below). You may experiment with additional branch predictors for additional bonus points.

Part 1: Set up Gem5

For this assignment, you **DO NEED TO COMPILE THE GEM5 SIMULATOR.** Every time you make a change to the “source” files, you need to compile to see the result.

There are 2 ways in which you can compile the GEM5 simulator in your own environment.

1. Using the sig servers:

Four servers have also been setup for the purposes of running gem5 and completing the programming assignments. The servers are:

```
sig1.eecs.wsu.edu

sig2.eecs.wsu.edu
```

```
sig3.eecs.wsu.edu

sig4.eecs.wsu.edu
```

To access the server, you can use your EECS id and password as follows:

Log on server from Windows:

1. Open PuTTY, and find PuTTY configuration window
2. Set host name to be "sig#.eecs.wsu.edu", where # is to be replaced with 1, 2, 3, or 4
3. Select connection type to be "SSH"
4. Click on open
5. Fill in your EECS ID and password

Log on server from Mac/Linux:

1. Open Terminal;
2. Type in command "ssh <your-eecs-id>@sig1.eecs.wsu.edu";
3. Type in your password.

- You can use the "sig1--4" servers to build gem5. In the server all the necessary dependencies are installed. You simply need to Download the gem5 to your "Home Directory (i.e. ~/)" or any other directory of your choice in the server and build gem5 using the "scons" command. The gem5 source code is available on Blackboard along with the write up of this assignment. Once you download the source file tarball to your computer, transfer it to the server using tools like WinSCP, FileZilla or other SFTP software. On the server, run the following command to untar the archive

```
$ tar xvf gem5.tar.gz
```

Once successfully downloaded, we now need to compile (build) the system for X86 Architecture. In the GEM5 folder, you do need to run this command

```
$ cd gem5
$ scons build/X86/gem5.opt -j4
```

This would build the "gem5.opt" executable which we will use to run the simulations.

In some cases gem5 build may fail using the above coming due to system running out of memory. In such cases, remove the -j4 part from the command and rerun the build.

2. **Using your own Machine/system:**

You also have the choice of installing the GEM5 on your own system of choice. Note that the TA will not be able to help troubleshooting much with this one, as everyone's system is different. But here are few tips:

- If you are using Windows, you need to use VM to install Linux/Unix and then install the dependencies needed to run GEM5 in your system (Details are in the GEM5 website; Link above)
- If you are using Linux/Unix, same thing for the dependencies and then install gem5.
- If you are using MAC OSX, then, using MACPORTS to install the dependencies actually works better rather than HOMEBREW. The reason is, MACPORTS needs sudo permission and install the binaries in the default locations (unlike HOMEBREW, which uses the – prefix). So, unless you are making changes to your MakeFile (or SConscript File), by default it will look for the binaries in the default location, which HOMEBREW does not install to.

The steps for installation of dependencies and gem5 are outlined at the following link:

https://www.gem5.org/documentation/general_docs/building

If you are using an Ubuntu system, you can follow the following steps:

1. Install the dependencies:
   ```
   sudo apt-get install python-dev scons m4 build-essential g++ swig
   zlib-dev
   ```
2. Build:
   ```
   scons build/X86/gem5.opt [-j<#cpus>]
   ```

Even if using your own machine, please download the source from Blackboard.

Step 2: Benchmarks

We will the same set of CPU benchmarks as in PA1 to simulate the architectural designs. The benchmarks have been set up at: https://github.com/gmbhat/EE524_CPTS561

Use the following command to download the benchmarks using git:

```
$ git clone https://github.com/gmbhat/EE524_CPTS561
```

Or you could download the zip file from the following link: https://github.com/gmbhat/EE524_CPTS561. By now, you should all be familiar with how to run these benchmarks as we have done this in Assignment 1.

Note: You need to change the gem5 path in each of the runGem5.sh file to the directory in which you built the gem5 binary. In the last assignment we used pre-built binaries since we did not do any changes in the source code. For this assignment, you need to point it to the gem5 binary you built.

Compiling GEM5 may take some time. So if you plan to use the one of the sig servers, I suggest you start early, as you will be needing to compile the simulator multiple times during this assignment.

Part 2: Adding Branch Predictor Support to Timing Simple CPU:

By default, GEM5 TimingSimpleCPU does not have the BranchPredictor Support. You need to add the support of your Predictor. There are two methods by which you can add branch predictor to the simulations.

1. Specify the branch predictor as a command line option. This can be done using the command option below:

```
$ ./build/X86/gem5.opt ./configs/example/se.py --bp-type=TYPE
```

where you need to replace TYPE with one of the options in the introduction.

For example, we can specify a bi-mode branch predictor using the command below:

```
$ ./build/X86/gem5.opt ./configs/example/se.py --bp-type=BiModeBP -c
./tests/test-progs/hello/bin/x86/linux/hello
```

In the above example, we ran the hello world program while using the BiMode BP. The output of the simulation is:

```
warn: CheckedInt already exists in allParams. This may be caused by the
Python 2.7 compatibility layer.
warn: Enum already exists in allParams. This may be caused by the Python
2.7 compatibility layer.
warn: ScopedEnum already exists in allParams. This may be caused by the
Python 2.7 compatibility layer.
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 20.0.0.3
gem5 compiled Oct 26 2020 10:37:50
gem5 started Oct 26 2020 10:55:04
gem5 executing on sighup.eecs.wsu.edu, pid 15391
command line: ./build/X86/gem5.opt ./configs/example/se.py --bp-
type=BiModeBP -c ./tests/test-progs/hello/bin/x86/linux/hello

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file
and pdf.
warn: DRAM device capacity (8192 Mbytes) does not match the address range
assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 5985500 because exiting with last active thread context
```

This should generate the “m5out” folder in the gem5 directory. Now, if your compilation was successful, you should be able to check the Branch Predictor being used in the config.ini file

```
$ gem5/m5out$ vim config.ini
```

```
[system.cpu.branchPred]
type=BiModeBP
BTBEntries=4096
BTBTagSize=16
RASSize=16
choiceCtrBits=2
choicePredictorSize=8192
eventq_index=0
```

```
globalCtrBits=2
globalPredictorSize=8192
indirectBranchPred=Null
instShiftAmt=2
numThreads=1
```

2. The second method is to modify the source to choose the appropriate branch predictor. Below are the steps of adding the BranchPrediction support to TimingSimpleCPU by modifying the server. First, login to the server, go to the gem5 directory and execute the following steps:

```
$ cd gem5/src/cpu/simple
```

Edit the file named "BaseSimpleCPU.py" and at the bottom you should find the line:

```
branchPred = Param.BranchPredictor(NULL, "Branch Predictor")
```

Change the "NULL" to your predictor of choice each time and then recompile gem5. In this assignment we will choose one of: BiModeBP(), TournamentBP(), and LocalBP()

After completing the compilation, run the test HelloWorld program:

```
$ ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-progs/hello/bin/x86/linux/hello
```

```
warn: CheckedInt already exists in allParams. This may be caused by the Python 2.7
compatibility layer.
warn: Enum already exists in allParams. This may be caused by the Python 2.7
compatibility layer.
warn: ScopedEnum already exists in allParams. This may be caused by the Python 2.7
compatibility layer.
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 20.0.0.3
gem5 compiled Oct 25 2020 11:35:01
gem5 started Oct 25 2020 11:41:38
gem5 executing on sighup.eecs.wsu.edu, pid 25874
command line: ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-
progs/hello/bin/x86/linux/hello

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned
(512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 5985500 because exiting with last active thread context
```

This should generate the "m5out" folder in the gem5 directory. Now, if your compilation was successful, you should be able to check the Branch Predictor being used in the config.ini file (I used BiMode in this example).

```
gem5/m5out$ vim config.ini
```

```
[system.cpu.branchPred]
type=BiModeBP
BTBEntries=4096
BTBTagSize=16
RASSize=16
choiceCtrBits=2
choicePredictorSize=8192
eventq_index=0
globalCtrBits=2
globalPredictorSize=8192
indirectBranchPred=Null
instShiftAmt=2
numThreads=1
```

NOTE: If you're already comfortable building gem5, then you can use the first method for Part 2. Otherwise, you should practice modifying the source code of gem5 and building before you move on to the next part.

Part 3: Calculate BTB Miss Rate and Branch Misprediction Rate From Stats.txt

In Part 3, we will calculate the BTB miss rate and branch misprediction rate from the stats.txt file. Specifically, you need to identify the parameters in the stats.txt file that will allow us to get the two miss rates.

The **BTB miss rate** is defined as:

BTBMissPct = (1 - (BTBHits/BTBLookups)) * 100

where: BTB Hits -> total number of BTB Hits
BTBLookups -> total number of BTB References

Similarly, **branch misprediction rate** is defined as:

BranchMispredPercent = (numBranchMispred / numBranches) * 100;

where: numBranchMispred	->	total number of mispredicted Branches
numBranches	->	total number of branches fetched

Part 4: Running the benchmarks with the changes and comparing the different Branch Predictors

You need to run all 5 benchmarks that you downloaded in Part 1 using the new runGem5.sh files. Note that the runGem5.sh files in each benchmarks have been updated for the second assignment. Therefore, you need to update them using 'git pull' command or downloading the files again. The cache parameters are not to be changed in this assignment.

We will run the programs for **500 million** instructions.

Change the following parameters (highlighted with the new numbers):

You should change the **"src/cpu/pred/BranchPredictor.py'** file to make these changes.

```
class BranchPredictor(SimObject):
    type = 'BranchPredictor'
    cxx_class = 'BPredUnit'
    cxx_header = "cpu/pred/bpred_unit.hh"
    abstract = True

    numThreads = Param.Unsigned(Parent.numThreads, "Number of threads")
    BTBEntries = Param.Unsigned(4096, "Number of BTB entries")
    BTBTagSize = Param.Unsigned(16, "Size of the BTB tags, in bits")
    RASSize = Param.Unsigned(16, "RAS size")
    instShiftAmt = Param.Unsigned(2, "Number of bits to shift instructions
by")

    indirectBranchPred =
Param.IndirectPredictor(SimpleIndirectPredictor(),
      "Indirect branch predictor, set to NULL to disable indirect
predictions")

class LocalBP(BranchPredictor):
    type = 'LocalBP'
    cxx_class = 'LocalBP'
    cxx_header = "cpu/pred/2bit_local.hh"

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

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

    localPredictorSize = Param.Unsigned(2048, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
    localHistoryTableSize = Param.Unsigned(2048, "size of local history
table")
    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")


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")
```

You have to a do a design space exploration to understand the change in branch predictor accuracy with change in size or other parameters. You can choose any power of 2 for the parameters. It is open ended so that you can explore the design space by yourself. However, you need not go too high or low. Consider 512 on the lower end and 8192 on the higher end. As long as you get the trend in the behavior, exploring additional parameters won't add too much.

Part 5: Experiment with Additional Branch Predictors (Optional)

You experimented with three common branch predictors in Part 4. In this part, you can choose any two branch predictors that were not used in Part 4. Modify the appropriate parameters for the chose branch predictors and report the results. You can complete this part for additional bonus points. We can discuss this part during office hours if you're interested in doing the optional part for bonus points.

Deliverables:

At the end of the assignment, you should be able to discuss the effect of changing different Branch Predictors and what impact you observe by changing their respective parameters. Write a report that discusses the following for each part.

From Part 1:
Discuss whether you are using the sig servers or your own environment. What (if any) were the challenges.

From Part 2:
Show the result of your **config.ini** file which shows the **BiModeBP.** State whether you rebuilt gem5 or used the command line option.

From Part 3:
Please provide the parameters that you identified in the stats.txt file to get the BTB and branch miss prediction rates.

From Part 4:
Generate the **BTB miss rate and Branch misprediction rate** for each of the Branch Predictors for each Benchmark. Discuss your findings and justify the results that you observe. Also discuss how the different branch predictor choices will affect the CPI of each benchmark.

From Part 5:
Briefly discuss the properties of the branch predictors you choose for additional experiments. Then, present the **BTB miss rate and Branch misprediction rate** for each of the Branch Predictors with the settings provided (highlighted in Part 4) for each Benchmark. Discuss whether the new branch predictors offer any advantages over the branch predictors simulated in Part 4.

Include the following files in your submissions
1. PDF file of your report
2. m5out folder for each of your simulations. The m5out folders help me in understanding your simulation parameters

Grading rubric:
Part 1: 10 points
Part 2: 15 points
Part 3: 25 points
Part 4: 50 points
Part 5: Up to 30 bonus points