
Project # 2

Project Description:

Using GEM5, in this project, we will study the effect of varying Branch Prediction parameters on different benchmarks.

There are 3 kinds of Branch Predictors available in GEM5 (stable).

1. 2bit_local Predictor
2. Bi_mode Predictor
3. Tournament Predictor

These files are found in the “\$gem5/src/cpu/pred/” folder. We will select each of these Branch Predictors and assess the effect of misprediction of branches when we change the various parameters (details below).

Part 0: Find a (one) teammate

Projects will be carried out in teams of two. You are asked to identify your teammate and send an email to the TA and Professor with the names and UTD NetIDs of the team members by. If you cannot find a teammate, email the TA before the above date and one will be assigned to you (cc me).

Part 1: Setting up the gem5

Step 1: setup Gem5 on your PC or server

Gem5 is the simulator we will use in this project. The gem5 simulator is a modular platform for computer architecture related research. Gem5 Wiki is a good source if you want to know more.

Gem5 Wiki: http://gem5.org/Main_Page

Setup Gem5 on Your PC

Steps to install gem5 are generally broken down into two steps: first, install all dependencies; second, install gem5 itself. Detailed steps can be found in following links.

Install dependencies: <http://gem5.org/Dependencies>

Install gem5 itself: http://gem5.org/Build_System

Particularly, if you choose to install gem5 on Ubuntu system, you have an easier way to install dependencies (sudo for access administrator access):

```
sudo apt-get install python-dev scons m4 build-essential  
g++ swig zlib-dev  
# ('sudo' for gaining  
administrator access)
```

Access Gem5 on Server

A virtual server for the class has been set up for the purpose of carrying out the projects. The server is **ce6304.utdallas.edu**, which is accessible from on-campus machines or via UTD VPN from off-campus machines. You will need your UTD NetID and password to access it. To access the server, you can use your UTD netid and password as follows:

Log on server from Windows:

1. Open PuTTY, and find PuTTY configuration window;
2. Set host name to be "ce6304.utdallas.edu";
3. Select connection type to be "SSH";
4. Click on open;
5. Fill in your NetID and password in coming windows.

Log on server from Mac/Linux:

1. Open Terminal;
2. Type in command "`ssh <your-net-id>@ce6304.utdallas.edu`";
3. Type in your password.

The gem5 simulator is found in "**/usr/local/gem5**". You can find and use all related folders and scripts in this directory. However, you cannot write or modify files within the directory, as it requires administrator access. This means that if you want to output files from the gem5 tool, or modify an existing script, you will need to define an output directory in your own network drive, and copy scripts from the gem5 directory to your local directory.

For this project, 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.

- Using the CE6304 Server:
- You can use the "ce6304" server. 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. Follow the instruction found in GEM5 Website for download:
<http://gem5.org/Download>
- 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
(http://gem5.org/Compiling_M5):

```
% scons build/X86/gem5.opt
```

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

If you are using the gem5 version installed in a UTD server, remember that you do not have write permissions where it is installed. Thus, because you need to re-compile it, you need to copy the gem5 folder to your local directory and work with this copy of gem5. E.g

```
% cp -rf /usr/local/gem5 .  
% scons build/X86/gem5.opt
```

- 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 tidbits from prior experience:

- 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.

After downloading the necessary dependencies, use the “scons” command to build your X86 architecture supporting simulator:

```
% scons build/X86/gem5.opt
```

We will be using the same Benchmarks that we used for Project 1. Download the Pre-compiled SPEC benchmarks from:

```
https://github.com/timberjack/Project1\_SPEC
```

By now, you should all be familiar with how to run these benchmarks as we have done this in Project 1.

****Compiling GEM5 may take some time. So if you plan to use the CE6304 server, I suggest you start early, as you will be needing to compile the simulator multiple times during this project.****

Gem5 Usage

Gem5 needs to be called from the Linux prompt. The gem5 command line has four parts:

- the gem5 binary
- options for the binary
- simulation script
- options for the script.

The options that are passed to the gem5 binary and those passed to the script are handled separately, so be sure any options you use are being passed to the right component.

```
% <gem5 binary> [gem5 options] <simulation script> [script options]
```

Example:

```
cd 401.bzip2
rm -rf m5out
export ARGUMENT=./data/input.program
time $GEM5_DIR/build/X86/gem5.opt -d ./m5out $GEM5_DIR/configs/example/se.py -c $BENCHMARK -o $ARGUMENT -I 500000000 --cpu-type=TimingSimpleCPU --caches --l2cache --l1d_size=128kB --l1i_size=128kB
```

Where EXPORT GEM5_DIR, BENCHMARK and ARGUMENT sets the paths to the gem5 binary, benchmark and argument file in Linux environment variables (more elegant way of doing it instead of passing all the long paths when calling gem5).

To run the file : %sh runGem5.sh in Project1_SPEC folder

Step 2: Benchmarks

We will use a set of CPU benchmarks to simulate your architectural design. You can download the benchmarks at the following link:

https://github.com/timberjack/Project1_SPEC.git

Or you could download the zip file from the following link:

https://github.com/timberjack/Project1_SPEC

There are a total of five benchmarks which we will simulate with:

401.bzip2
429.mcf
456.hmmer
458.sjeng
470.lbm

Within each benchmark directory, you can find a “src” directory that contains the source code and an executable binary, and a “data” directory that contains any necessary input files/arguments. Within the “src” directory, the executable file named “benchmark” is the program that you need to simulate with. You can also open the script “run.sh” to understand how to run the benchmark.

Within “spec” folder, you can also find the script “runGem5.sh”. It is a sample script that runs gem5 on benchmark **429.mcf**. You can use this as an example how to specify the command line to run a benchmark program on gem5.

As some of the benchmark programs can take considerably long time (e.g. bzip2 can take 20 hours, and sjeng can take 10 hours), you can define a max number of instructions to be executed, e.g. a maximum of 5×10^8 (500000000) instructions. The way to specify this constraint is discussed in the following section.

Please feel free to run your own programs or programs from the following suites to test the simulator as well:

<http://euler.slu.edu/~fritts/mediabench/>
<http://groups.csail.mit.edu/cag/streamit/shtml/benchmarks.shtml>
<http://axbench.org/>
<https://asc.llnl.gov/CORAL-benchmarks/>
<http://math.nist.gov/scimark2/>

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. Below are the steps of adding the BranchPrediction support to Timing Simple CPU:

```
$ce6304:> 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.

```
branchPred = Param.BranchPredictor(LocalBP(), "Branch Predictor")  
branchPred = Param.BranchPredictor(TournamentBP(), "Branch Predictor")  
branchPred = Param.BranchPredictor(BiModeBP(), "Branch Predictor")
```

**** It is advised that before recompiling, delete all the files in \$gem5/build/X86 first and then use the “scons” command to recompile in order to reflect the changes without any glitch. ****

After completing the compilation, run the test HelloWorld program:

```
ce6304@~ ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-  
progs/hello/bin/x86/linux/hello
```

```
gem5 Simulator System. http://gem5.org  
gem5 is copyrighted software; use the --copyright option for details.
```

```
gem5 compiled Nov  7 2016 10:48:04  
gem5 started Nov  7 2016 10:49:29  
gem5 executing on cometnet-10-21-46-191.utdallas.edu  
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: DRAM device capacity (8192 Mbytes) does not match the address range  
assigned (512 Mbytes)
```

```
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0. Starting simulation...
Hello world!
Exiting @ tick 415502000 because target called exit()
```

This should generate the “m5out” folder in the \$gem5-stable 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-stable/m5out$ vi config.ini
```

```
86 [system.cpu.branchPred]
87 type=BiModeBP
88 BTBEntries=4096
89 BTBTagSize=16
90 RASSize=16
91 choiceCtrBits=2
92 choicePredictorSize=8192
93 eventq_index=0
94 globalCtrBits=2
95 globalPredictorSize=8192
96 instShiftAmt=2
97 numThreads=1
```

Part 3: Adding extra Resulting Parameter in the Stats.txt file (output file)

In Part 2 we successfully ran the HelloWorld Program with the “BiModeBP”.

In Part 3, we shall add 2 more parameters in the stats.txt file and recompile. The reason for this part is to get you familiarized with the source code of GEM5 and be able to make basic changes to the simulator.

Below is the “stats.txt” file with the modification:

```
./stats.txt:system.cpu.branchPred.lookups          1317      # Number of BP lookups
./stats.txt:system.cpu.branchPred.condPredicted    1317      # Number of conditional
branches predicted
./stats.txt:system.cpu.branchPred.condIncorrect     621      # Number of conditional
branches incorrect
./stats.txt:system.cpu.branchPred.BTBLookups        203      # Number of BTB lookups
./stats.txt:system.cpu.branchPred.BTBHits           202      # Number of BTB hits
./stats.txt:system.cpu.branchPred.BTBCorrect        0        # Number of correct BTB
predictions (this stat may not work properly.
./stats.txt:system.cpu.branchPred.BTBHitPct         99.507389 # BTB Hit Percentage
./stats.txt:system.cpu.branchPred.BTBMissPct         0.492611 # BTB Miss Percentage
./stats.txt:system.cpu.branchPred.usedRAS            8        # Number of times the
RAS was used to get a target.
./stats.txt:system.cpu.branchPred.RASInCorrect       0        # Number of incorrect
RAS predictions.
./stats.txt:system.cpu.Branches                     1317      # Number of branches
fetched
./stats.txt:system.cpu.predictedBranches            210      # Number of branches
predicted as taken
./stats.txt:system.cpu.BranchMispred                621      # Number of branch
mispredictions
./stats.txt:system.cpu.BranchMispredPercent          47.152620 # Percent of Branch
Mispredict
```

The highlighted parameters are not there by default and you need to add them to the source files and recompile to generate the new “stats.txt” file with the parameters.

The **BTBMissPct** is defined as:

$$\text{BTBMissPct} = (1 - (\text{BTBHits}/\text{BTBLookups})) * 100$$

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

Similarly, we also implemented the **BranchMispredPercent** as below:

$$\text{BranchMispredPercent} = (\text{numBranchMispred} / \text{numBranches}) * 100;$$

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 with the settings mentioned in the respective **run.sh** files. (Do not simply run the benchmarks. Pay attention to the parameters used in the **run.sh** files).

We will run the programs for **500 million** instructions. Use the following command to run gem5 (the cache parameters remains the same for this project):

```
$ce6304: time $GEM5_DIR/build/X86/gem5.opt -d ./m5out  
$GEM5_DIR/configs/example/se.py -c $BENCHMARK -o "$ARGUMENT" -I 500000000 --  
cpu-type=timing --caches --l2cache --l1d_size=128kB --l1i_size=128kB --  
l2_size=1MB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --cacheline_size=64
```

where:

```
$BENCHMARK    -> the SPEC Benchmark  
$ARGUMENT      -> Arguments as defined in "run.sh" file for each benchmark
```

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

```
class BranchPredictor(SimObject):  
    type = 'BranchPredictor'  
    cxx_class = 'BPredUnit'  
    cxx_header = "cpu/pred/bpred_unit.hh"  
    abstract = True  
    numThreads = Param.Unsigned(1, "Number of threads")  
    BTBEntries = Param.Unsigned(2048, "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")
```

```
class TournamentBP(BranchPredictor):  
    type = 'TournamentBP'  
    cxx_class = 'TournamentBP'  
    cxx_header = "cpu/pred/tournament.hh"  
    localPredictorSize = Param.Unsigned(1024, "Size of local predictor")  
    localCtrBits = Param.Unsigned(2, "Bits per counter")  
    localHistoryTableSize = Param.Unsigned(2048, "size of local history  
table")  
    globalPredictorSize = Param.Unsigned(4096, "Size of global predictor")  
    globalCtrBits = Param.Unsigned(2, "Bits per counter")  
    choicePredictorSize = Param.Unsigned(4096, "Size of choice predictor")  
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

```
class LocalBP(BranchPredictor):  
    type = 'LocalBP'  
    cxx_class = 'LocalBP'  
    cxx_header = "cpu/pred/2bit_local.hh"  
    localPredictorSize = Param.Unsigned(1024, "Size of local 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(2048, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(2048, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

Deliverables:

At the end of the project, you should be able to discuss the effect of changing different Branch Predictors and what impact you observe by changing their respective parameters.

From Part 1:

Discuss whether you are using the UTD Server (i.e. ce6304) or your own environment. What (if any) were the challenges.

Describe what a branch predictor is and the three different types of predictors used in gem5. How do they work.

From Part 2:

Show the result of your **config.ini** file which shows the **TournamentBP**.

From Part 3:

Please provide the changes you made to the source code in order to add the parameters required in **Part 3**. Present **only changes to the source code** and which file you edited. Do not include the whole source file.

From Part 4:

Generate the **BTBMissPct** and **BranchMispredPercent** for each of the Branch Predictors with the settings provided (highlighted in Part 4) for each of the 5 Benchmarks

Finally, **select Tournament BP**, and vary the sizes of the predictor and run the simulation on all the benchmarks. Discuss your findings and justify the results that you observe.

Important information:

- Projects should be done in groups of two. In exceptional cases single person or more than 2 students groups can be formed, with the explicit consent of the subject lecturer
- Allocate the marks that each group member should get on the front page of your report (e.g. 100/100 = every group member receives the same grade, or 90/110= group member two gets 10% more marks)

Deliverable Format:

- Software package with your solutions

- Power point slides, as if you were presenting to your boss having 10 minutes. These should include
 - Summary of what branch prediction is and why it is important
 - Describe briefly the 3 types of branch predictors in gem5 and their trade-offs.
 - Summarizes how you made gem5 work and problems found and how you resolved them if any.
 - Describe changes made to specify the different branch predictors (where in gem5)
 - Show the results obtained when using different branch predictors and compare all 3 of them.
 - Report how varying the size of the predictor affects the BTB miss and Branch prediction miss for the 5 different benchmarks.
 - Discuss the results
- Marking:
 - Technical contents (deliverables met) = 80%
 - Clarity of results presented= 20% (see marking sheet at the end of this document)

UTD Honor Code

UTD values academic integrity. Therefore, all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures

Group assignments must be completed solely by the members of the group. Cross-group work is not allowed. Moreover, similar assignments have been offered before at UTD and other universities. Any use of information from previous assignments is prohibited. The tutorials are to be taken individually. Failure to respect this rule constitutes dishonesty and is a direct violation of the University Honor Code.