gem5: Quick Start Guide + Tasklist

ECE 666 – Spring 2021

Goal:

The goal of this assignment is to get familiar with a broad set of gem5 features. The assignment consists of 5 tasks. There are step-by-step instructions for the first four tasks. The last one should be done by consulting gem5 documentation. Please pay attention to the deliverables for each task. You are to upload a single tarball to Brightspace with the files for each task's deliverables.

Cloing gem5.

Use the command: `git clone /home/yara/mithuna2/gem5`

This is the latest version of **gem5**. It has some changes that make it possible to build in the saguaro environment. Do **NOT** download **gem5** directly from the source website.

____

Compiling **gem5**.

You should already be familiar with **gem5** compilation. Use the customized "**build_opts**" file for ECE 666. The default version of **gcc** available on ECN machines is not current enough for **gem5**. As such, you should change your **PATH** and **LD_LIBRARY_PATH** environment variable to point to **/package/gcc/8.3.0/bin** and **/package/gcc/8.3.0/lib64** respectively. (The exact commands to set the environment variables will depend on your shell.)

```
scons-3.6 -j 16 build/ECE666/gem5.opt
```

'**scons**' is a build system like 'make'. On the saguaro machines, the specific version (**3.6**) is part of the filename of the executable; hence the use of '**scons-3.6**'. The "**-j 16**" enables faster compilation by using 16 parallel threads on the saguaro machines. To start over, the equivalent of a 'make clean' is to delete the build directory. (The entire 'build' directory is produced during compilation and all generated files are held in that directory. Deleting it is effectively a clean start.)

TASK #1: Run your code on the simulator with 4 cores

Run your pthreads programs on the gem5 simulator in gem5's System Emulation (SE) Mode. Run both the matrix multiplication and integer sort programs with 4 threads. In the command below, the simulation does not simulate coherence operations, which will be added later.

```
./build/ECE666/gem5.opt configs/example/se.py -n 4 -c <full path to
pthreads binary>
```

Running the program produces outputs in the m5out directory. (You can also customize the output directory using the `'-d'` option.) Examine the stats.txt file for the execution statistics.

Deliverable: Include the stats.txt in your tarball.

TASK #2: HTML Table view of Coherence Protocol

The build options enable creation of a tabular representation of the coherence protocol. After compilation, look in the build directory at the following path.

```
<gem5root>/build/ECE666/mem/protocol/html/index.html
```

This is a key sanity check when modifying coherence protocols. If the cells in the table do not make sense, the coherence protocol will be much harder to debug. The best way to understand coherence protocol implementation in gem5 is to look at the tables and the slicc files that implement the coherence protocol side-by-side.

The current build options implement a simple MI coherence protocol. The slicc files for the MI protocol can be found at <gem5root/src/mem/ruby/protocol>. The relevant files are MI-example-*. The files include one top-level "**`.slicc`**" file and several individual "**`.sm`**" (**`sm`** → state machine) files.

Deliverable: Include the all the HTML files in your tarball.

TASK: #3 Running the Ruby random tester

The table view (from task 2) is a static, compile time sanity check. We now consider ways to do run-time/dynamic debugging of coherence protocols. You will be using two major tools.

- The Ruby random tester performs random writes and reads and checks the expected value. (If you get the sense that this is an *ad hoc* approach, you're right. We are effectively saying, "if we cannot expose a coherence bug in a reasonably long random test, the coherence protocol is bug-free enough.")
- Using the **`ProtocolTrace`** debug flag. (You must be familiar with debug flags from ECE565. This enables tracking of coherence actions for specific blocks to understand bugs.)

- Run the ruby random tester using:

```
build/ECE666/gem5.opt ./configs/example/ruby_random_test.py
```

- Can change number of CPUs via the **`–n`** option.
  - Debugging Strategy: Iterative widening
    - Debug with 1 CPU, then 2, then 4, and so on.
  - For example, to use 4 CPUs, the command would be

```
build/ECE666/gem5.opt configs/example/ruby_random_test.py -n 4
```

- Can change the number of checks (loads) via the **`--maxloads`** option.
    - Debugging strategy: Iterative deepening
        - Debug with 100 checks, then 1000, then 10,000 and so on.

```
build/ECE666/gem5.opt configs/example/ruby_random_test.py -n 4  --
maxloads 100
```

- Remember to turn on the **`ProtocolTrace`** debug flag as shown below. (There are other options to delay the tracing until after a certain Tick. See **`gem5`** documentation.)

```
build/ECE666/gem5.opt --debug-flags=ProtocolTrace
configs/example/ruby_random_test.py -n 4 -maxloads 100
```

Deliverable: Save the ProtocolTrace in a file. Select a short segment of the Protocol trace that shows a cache-to-cache transfer. Save the segment in a file, and include the file in the tarball.

TASK #4: Cache simulation with real applications

Rerun your pthreads programs (matmult + integer sort) with **Ruby** cache simulation enabled. This is done by adding the **'- - ruby'** option as shown below.

```
 ./build/ECE666/gem5.opt configs/example/se.py -n 4 --ruby -c <full
path to pthreads binary>
```

Examine and understand the gathered statistics. Verify that there were no errors and that the simulation ran to completion.

Deliverable: Include the output statistics (stats.txt renamed as ruby.txt).

TASK #5: Full System Simulation with real applications

This is a major task for this homework. You have to follow the gem5 documentation (http://learning.gem5.org/book/index.html) and boot up the simulator with a Linux operating system.

You are then required to run the pi-estimation Monte Carlo pthreads programs in your full system simulator.

Deliverable: Include the output statistics (stats.txt renamed as fullsystem.txt).