CDA-5106 Project Phase 1

Installation:

System specifications:
Processor: Intel Xeon® CPU-E5-1650 v2 @ 3.50GHz × 12
System Memory: 8GB
Hard Drive: 160 GB HDD
OS: Linux Ubuntu 16.04 LTS

The package gem5 was installed twice, once with the 'opt' binary and the other with the 'debug' binary. The 'opt' binary is optimized with some debugging options not omitted from the binary. The 'debug' is has all optimizations turned off, hence it runs significantly slower than other binaries. The build time for the gem5.opt was 17 minutes whereas gem5.debug took about the same time. Gem5 was also installed on a virtual machine created on Windows 10 running on an Intel i7-4510u with 8gm RAM as well but that was quite laggy and took a long time to complete (~50 minutes for the build) due to which it was deemed unsuitable for benchmarking.

Installation instructions for both were taken from "http://learning.gem5.org". However, it fails to list Boost libraries as a dependency as a result of which initial builds for gem5 were failing. After installing Boost using the Ubuntu Package Manager (the apt command), the build completed successfully.

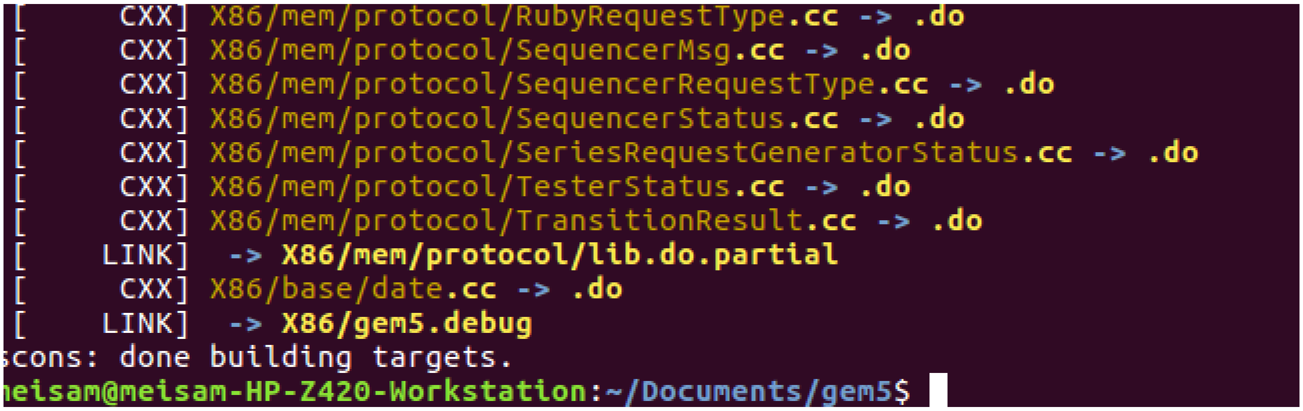
The installation process uses "*scons*" which is a build tool in which the configuration files are written in Python. This makes the build process easier as it allows the developer to deal with build issues using the power of a high level programming language. It must also be noted that gem5 currently only works with Python 2.6 or Python 2.7. Python 3.X is currently not supported so its good practice to create a virtual environment if the default Python version on one's system is Python 3.X.

In order to trivialize this process, I wrote a Bash script that runs all the required commands so the user only has to run one script. The script can be found in Appendix 1.

The user can replace the "*build/X86/gem5.opt*" with " *build/X86/gem5.debug*" to change the binary version for the build. The flag "-j4" tells *scons* to use 4 cores of the machine. It can be changed to potentially speed up the build process by parallelizing tasks.

The install command also gives the user the option to build gem5 for other architectures apart from x86. For the purpose of this project, gem5 was also compiled for the ARM architecture.

The user should get the output shown in Figure 1 when the gem5 has completed building without errors.



```
[      CXX] X86/mem/protocol/RubyRequestType.cc -> .do
[      CXX] X86/mem/protocol/SequencerMsg.cc -> .do
[      CXX] X86/mem/protocol/SequencerRequestType.cc -> .do
[      CXX] X86/mem/protocol/SequencerStatus.cc -> .do
[      CXX] X86/mem/protocol/SeriesRequestGeneratorStatus.cc -> .do
[      CXX] X86/mem/protocol/TesterStatus.cc -> .do
[      CXX] X86/mem/protocol/TransitionResult.cc -> .do
[     LINK]  -> X86/mem/protocol/lib.do.partial
[      CXX] X86/base/date.cc -> .do
[     LINK]  -> X86/gem5.debug
scons: done building targets.
meisam@meisam-HP-Z420-Workstation:~/Documents/gem5$ 
```

Figure 1: Successful build completion

Test Cases:

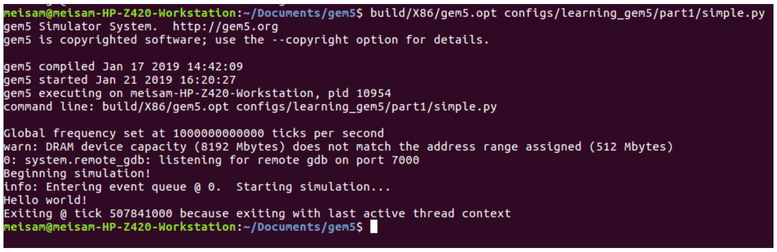
Running gem5 requires a command comprising of 4 parts.

```
<gem5 binary> [gem5 options] <simulation script> [script options][1]
```

The gem5 binary is the binary that was compiled during the installation phase; *gem5.opt* or *gem5.debug* in this case. Running the gem5 binary with the “-h” flag gives a list of options available to the gem5 binary as shown in Figure 2.

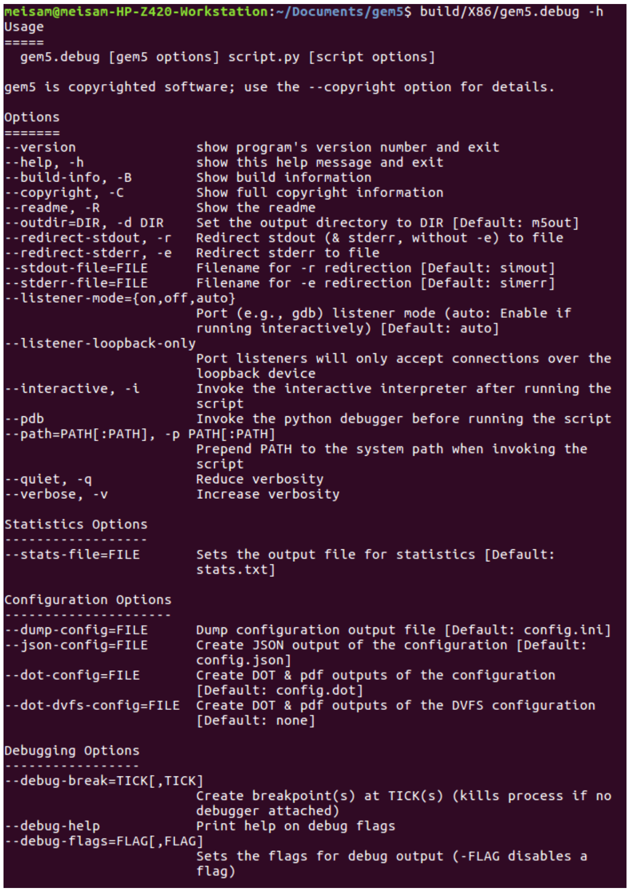
The simulation script is a Python script that creates a gem5 system and invokes a binary to be simulated on the system. Using print commands, the user can get an idea of the speed of execution of the binary.

The bash script in Appendix 1 runs the basic script written by the authors of gem5 (and is available in the gem5 github repository). The output of the command is shown in Figure 3.



```
meisam@meisam-HP-Z420-Workstation:~/Documents/gem5$ build/X86/gem5.opt configs/learning_gem5/part1/simple.py
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jan 17 2019 14:42:09
gem5 started Jan 21 2019 16:20:27
gem5 executing on meisam-HP-Z420-Workstation, pid 10954
command line: build/X86/gem5.opt configs/learning_gem5/part1/simple.py

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: listening for remote gdb on port 7000
Beginning simulation!
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 507841000 because exiting with last active thread context
meisam@meisam-HP-Z420-Workstation:~/Documents/gem5$ 
```

Figure 3: Successful run of gem5



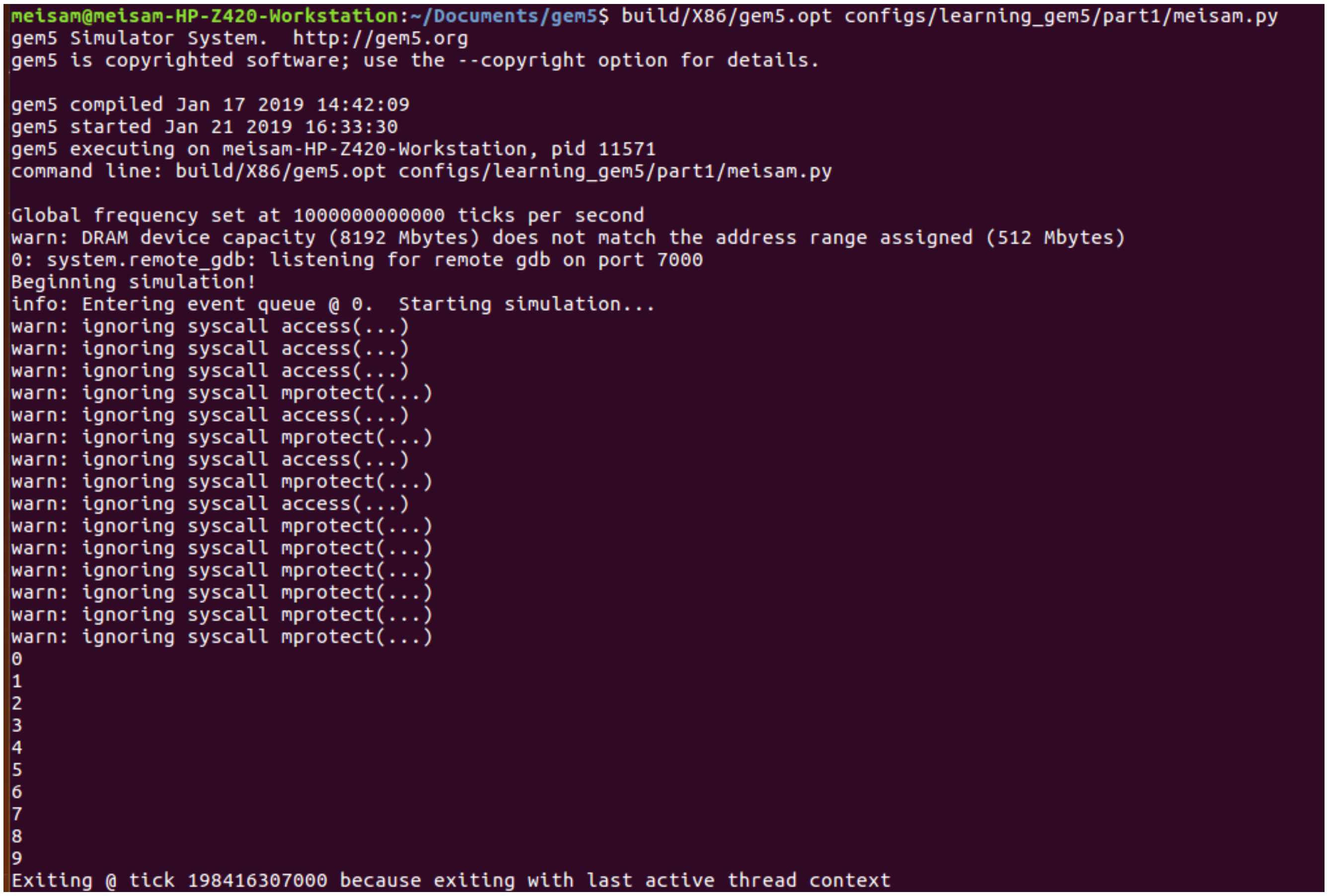
```
meisam@meisam-HP-Z420-Workstation:~/Documents/gem5$ build/X86/gem5.debug -h
Usage
=====
  gem5.debug [gem5 options] script.py [script options]

gem5 is copyrighted software; use the --copyright option for details.

Options
=======
--version               show program's version number and exit
--help, -h              show this help message and exit
--build-info, -B        Show build information
--copyright, -C         Show full copyright information
--readme, -R            Show the readme
--outdir=DIR, -d DIR    Set the output directory to DIR [Default: m5out]
--redirect-stdout, -r   Redirect stdout (& stderr, without -e) to file
--redirect-stderr, -e   Redirect stderr to file
--stdout-file=FILE      Filename for -r redirection [Default: simout]
--stderr-file=FILE      Filename for -e redirection [Default: simerr]
--listener-mode={on,off,auto}
                        Port (e.g., gdb) listener mode (auto: Enable if
                        running interactively) [Default: auto]
--listener-loopback-only
                        Port listeners will only accept connections over the
                        loopback device
--interactive, -i       Invoke the interactive interpreter after running the
                        script
--pdb                   Invoke the python debugger before running the script
--path=PATH[:PATH], -p PATH[:PATH]
                        Prepend PATH to the system path when invoking the
                        script
--quiet, -q             Reduce verbosity
--verbose, -v           Increase verbosity

Statistics Options
------------------
--stats-file=FILE       Sets the output file for statistics [Default:
                        stats.txt]

Configuration Options
---------------------
--dump-config=FILE      Dump configuration output file [Default: config.ini]
--json-config=FILE      Create JSON output of the configuration [Default:
                        config.json]
--dot-config=FILE       Create DOT & pdf outputs of the configuration
                        [Default: config.dot]
--dot-dvfs-config=FILE  Create DOT & pdf outputs of the DVFS configuration
                        [Default: none]

Debugging Options
-----------------
--debug-break=TICK[,TICK]
                        Create breakpoint(s) at TICK(s) (kills process if no
                        debugger attached)
--debug-help            Print help on debug flags
--debug-flags=FLAG[,FLAG]
                        Sets the flags for debug output (-FLAG disables a
                        flag)
```

Figure 2: Gem5 options

	Gem5.opt	*Gem5.debug*
Build time	1080	1080
Number of ticks	507841000	507841000
Runtime	1.010	1.601

Table 1: Statistics for x86

After running the hello world program included in the repository, I altered one of the files to run of my own binaries that I had compiled myself. I ran the binary with both the “gem5.opt” and “gem5.debug” binaries to observe the number of ticks and total runtime required. The output for the “gem5.opt” is shown in Figure 4.

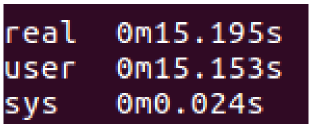


```
meisam@meisam-HP-Z420-Workstation:~/Documents/gem5$ build/X86/gem5.opt configs/learning_gem5/part1/meisam.py
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jan 17 2019 14:42:09
gem5 started Jan 21 2019 16:33:30
gem5 executing on meisam-HP-Z420-Workstation, pid 11571
command line: build/X86/gem5.opt configs/learning_gem5/part1/meisam.py

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: listening for remote gdb on port 7000
Beginning simulation!
info: Entering event queue @ 0.  Starting simulation...
warn: ignoring syscall access(...)
warn: ignoring syscall access(...)
warn: ignoring syscall access(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall access(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall access(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall access(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
0
1
2
3
4
5
6
7
8
9
Exiting @ tick 198416307000 because exiting with last active thread context
```

Figure 4: gem5.opt output

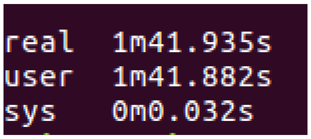
The number of ticks required for gem5.opt and gem5.debug were the same, possibly because the binary itself did not include any debug statements like “assert”. That is because the number of ticks being simulated does not depend upon the debug statements.

Next, I ran the two binaries using Ubuntu’s built in *time* command which gives the time a command took to execute. Here we see that the gem5.opt ran considerably faster than gem5.debug (refer to Figures 5 and 6) as was expected.



```
real	0m15.195s
user	0m15.153s
sys	0m0.024s
```

Figure 5: gem5.opt time



```
real	1m41.935s
user	1m41.882s
sys	0m0.032s
```

Figure 6: gem5.debug time

The “hello.py” script also run for the ARM architecture. The build time, number of ticks and the runtime is given in Table 2. All units are in seconds. The build times are estimates with the important point being that they took approximately the same amount of time.

	Gem5.opt	*Gem5.debug*
Build time	1080	1080
Number of ticks	372284000	372284000
Runtime	0.315	0.562

Table 2: Statistics for ARM

It was not possible to run a custom binary for the ARM version since gem5 currently does not support execution of cross compiled executables.

Appendix

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler
libprotoc-dev libgoogle-perftools-dev python-dev python
sudo apt install sudo apt-get install libboost-all-dev
git clone https://gem5.googlesource.com/public/gem5
cd gem5
scons build/X86/gem5.opt -j4

build/X86/gem5.opt configs/learning_gem5/part1/simple.py
```

Appendix 1: Script to build and run gem5

References

1. *http://gem5.org/Running_gem5*