

## Project #4

(Report due is 11:59:59 PM, 11/14/2022)

### Project description:

In previous projects, we went over the cache design issues and cache optimization. This project is different from the previous ones as we will explore the performance of multi-core compute system. Also, in this project, we will learn two different types of simulation mode in Gem5: *syscall* mode and *full system* mode. Please note that full system mode simulation usually takes a long time. So you need to start the project as early as possible. Whenever you face the trouble or problem on this project, please refer to Gem5 document here or contact TAs in WeChat. All the following information have been verified on Ubuntu 20.04.1 with Linux Kernel 5.15.0.

### Submission:

- Please submit (in a report form, both PDF and Word are acceptable) the deliverables for each part in order and clearly defined. Please give a brief description of the results in your report and DO NOT SUBMIT ANY SCRIPTS.
- Please send your report as an attachment at [ca2022fall@163.com](mailto:ca2022fall@163.com). The titles of your email AND attachment should both be Student ID\_Name\_Proj4 (e.g., 123456789\_Keqing\_Proj4).
- DO NOT PLAGIARIZE. After some projects, we will select 10 students randomly and ask them to run their codes in person and answer our questions related to their codes.

### Late policy:

- You will be given 3 slip days (shared by all projects), which can be used to extend project deadlines, e.g., 1 project extended by 3 days or 2 projects each extended by 1 days.
- Projects are due at 23:59:59, no exceptions; 20% off per day late, 1 second late = 1 hour late = 1 day late.

## Part 1: Gem5 Setup and Example Configuration Script Usage

In this project, we plan to explore multiple-core compute system. To support cache coherence for multi-core, we should enable cache coherence protocol in Gem5 (i.e., MOESI). This can be done by recompiling Gem5 with the following command:

```
scons -j8 build/ARM/gem5.opt PROTOCOL=MOESI_CMP_directory
```

In previous projects, we make simple assumption on our computer system, which only considers cpu, cache hierarchy, and interconnect bus. However, once the computer system becomes complex, it is not easy to manually write up the configurations of all components. Fortunately, Gem5 ships with multiple different example configuration files for different purposes. These configuration scripts reside in configs folder. The following figure shows the structure of configs folder:

```
yi@server-node3:~/labs/gem5$ ls configs/  
boot common dist dram example learning_gem5 network nvm proj1 proj2 proj3 ruby splash2 topologies
```

Among these folders, **common** and **example** folders are important in this project.

```
yi@server-node3:~/labs/gem5$ ls configs/common/
Benchmarks.py  cores      FileSystemConfig.py  GPULBOptions.py  MemConfig.py  SimpleOpts.py
CacheConfig.py  cpu2000.py  FSConfig.py         HMC.py           ObjectList.py  Simulation.py
Caches.py       CpuConfig.py  GPULBConfig.py      __init__.py      Options.py     SysPaths.py
yi@server-node3:~/labs/gem5$ ls configs/example/
apu_se.py      garnet_synth_traffic.py  hmc_test.py      memcheck.py      riscv      ruby_random_test.py
arm            gem5_library             hmc_tgen.cfg     memtest.py       ruby_direct_test.py  sc_main.py
etrace_replay.py  gpufs                   hsaTopology.py  noc_config       ruby_gpu_random_test.py  se.py
fs.py           hmc_hello.py            lupv             read_config.py   ruby_mem_test.py      sst
```

### Common:

This directory contains a number of scripts which is commonly used to build simulation system. For example, you can find out L1cache and L2cache configurations in caches.py.

- *Option.py*: this file provides variable options that can be set on the command line. You can search whether the option you want to change already has the command line parameter.
- *CacheConfig.py*: this file contains the options to set cache parameters.
- *MemConfig.py*: it contains the options to set memory parameters.
- *FSconfig.py*: it contains the necessary functions to set full system simulation.

### Example:

This folder contains several example configuration scripts which can be used directly to run Gem5 simulation. Among these files, se.py and fs.py are mostly used and they would be important in this project.

Next, let's explore how to use the example scripts. In part 1, we will use se.py as an example. se.py is an example configuration script for system call emulation (SE) mode. In this mode, rather than simulating whole operating system, only several system calls are implemented to execute the binary file. SE mode provides much faster simulation speed and low simulation overhead. So it is default simulation mode in Gem5. Let's start with a simple SE mode simulation with se.py. You can use the following commands to simulate "hello world" binary on Gem5:

```
build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/linux/hello
```

If the execution completes successfully, you can get the following output:

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

gem5 version 22.0.0.2
gem5 compiled Oct 29 2022 15:27:24
gem5 started Oct 29 2022 15:52:30
gem5 executing on server-node3, pid 2313992
command line: build/ARM/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/arm/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.
build/ARM/mem/dram_interface.cc:690: 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 ****
build/ARM/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulation...
Hello world!
Exiting @ tick 2916500 because exiting with last active thread context
Simulated exit code not 0! Exit code is 13
```

This command has no difference with the commands we used in previous projects, except for option -c which is used to pass the command of executable file. se.py provides various options for users to configure. All of the possible options are printed when you run:

```
build/ARM/gem5.opt configs/example/se.py --help
```

We list the most important options in the following lists:

- `--cpu-type`: select a specific cpu types
- `--num-cpus`: select the number of cpu cores
- `--sys-clock`: Top-level clock for blocks running at system speed
- `--cpu-clock`: Clock for blocks running at CPU speed
- `--mem-type`: type of memory to use
- `--mem-size`: Specify the physical memory size (single memory)
- `--l1d_size`, `--l1i_size`, `--l2_size`, `--l3_size`: define cache size

### Part 1 deliverables:

1. A screenshot that shows you run “hello” successfully.

## Part 2: FS mode

Gem5 is also capable of booting and running a real operating system image. To make FS mode simulation easier (especially for your project), we will use Linux image as its full-fledged operating system. Basically, you need two different extra works for FS mode simulation: i) *operating system* (a copy of the kernel) *binary* and ii) *disk image*. A copy of these two can be downloaded as follows:

```
cd ..  
mkdir fs-images  
cd fs-images  
wget http://dist.gem5.org/dist/v22-0/arm/aarch-system-20220707.tar.bz2  
wget http://dist.gem5.org/dist/v22-0/arm/disks/ubuntu-18.04-arm64-docker.img.bz2
```

Then you need decompress these files:

```
tar -xjf aarch-system-20220707.tar.bz2  
bzip2 -d ubuntu-18.04-arm64-docker.img.bz2  
mv ubuntu-18.04-arm64-docker.img disks/
```

Once you decompressed the files, you can see binaries folder and disk folders. The binary folder include the bootloader for ARM machines (e.g., boot.arm and boot.arm64). The files named with vmlinux (as its prefix) are related to operating system binary. You can use these OS files to simulation an operating system. On the other hand, the disks folder maintains disk image (which means the physical disk layout is formatted by a specific file system). In this project, you will use ubuntu-18.04-arm64-docker.img. On the other hand, the disks folder maintains disk image (which means the physical disk layout is formatted by a specific file system). In this project, you will use ubuntu-18.04-arm64-docker.img.

```
yi@server-node3:~/labs$ tree fs-images/
fs-images/
├── aarch-system-20220707.tar.bz2
├── binaries
│   ├── boot.arm
│   ├── boot.arm64
│   ├── boot_emm.arm64
│   ├── boot_foundation.arm64
│   ├── boot_v2.arm64
│   ├── vmlinux.arm
│   └── vmlinux.arm64
└── disks
    ├── m5_exit_addr.squashfs.arm64
    ├── m5_exit.squashfs.arm
    ├── m5_exit.squashfs.arm64
    └── ubuntu-18.04-arm64-docker.img
```

Now, you have both OS binaries and disk images, which should be given to gem5; you can let gem5 know them by setting the environmental variable, M5\_PATH, with the location of fs-images folder.

```
export M5_PATH=/home/yi/labs/fs-images/ # replace '/home/yi/labs' with your path
```

Now, you can start to run in FS mode, by typing the command:

```
cd gem5
```

```
build/ARM/gem5.opt configs/example/fs.py --kernel=vmlinux.arm64 --disk-image=ubuntu-18.04-arm64-docker.img --num-cpu=8
```

```
yi@server-node3:~/labs/gem5$ build/ARM/gem5.opt configs/example/fs.py --kernel=vmlinux.arm64 --disk-image=ubuntu-18.04-arm64-docker.img --num-cpu=8
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.0.0.2
gem5 compiled Oct 29 2022 15:27:24
gem5 started Oct 29 2022 22:29:23
gem5 executing on server-node3, pid 2330239
command line: build/ARM/gem5.opt configs/example/fs.py --kernel=vmlinux.arm64 --disk-image=ubuntu-18.04-arm64-docker.img --num-cpu=8

Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
build/ARM/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
build/ARM/sim/kernel_workload.cc:46: info: kernel located at: /home/yi/labs/fs-images/binaries/vmlinux.arm64
system.vncserver: Listening for connections on port 5900
system.terminal: Listening for connections on port 3456
system.realview.uart1.device: Listening for connections on port 3457
```

To operate the simulated system, you can use m5term to connect the simulated terminal:

```
cd gem5/util/term
```

```
make
```

```
./m5term *** # replace '***' with terminal port as shown above
```

However, the disk image is in a pristine state and there is no program in it. Thus, you will need to edit the disk file (i.e., ubuntu-18.04-arm64-docker.img) to contain the binary of the program you

want to execute as a multi-threaded program. To this end, you will first mount the img as a filesystem.

```
cd ..  
mkdir imgmnt  
sudo mount -o loop,offset=65536 fs-images/disks/ubuntu-18.04-arm64-docker.img ./imgmnt/  
cd imgmnt  
ls
```

Once you successfully mounted the image file, you can see the following folder list:

```
yi@server-node3:~/labs/imgmnt$ ls  
bin  data  etc  host  init.gem5  lib  media  opt  root  sbin  sys  usr  
boot dev  home init.addr.gem5 init.semi.gem5 lost+found mnt  proc  run  srv  tmp  var
```

In a real FS simulation, you also need to compile the multi-thread program under you target machine (e.g., ARM). As your machine is not an ARM machine (I believe) you actually need to cross compile the multi-thread program you want to execute. However, to reduce the amount of your efforts for this project, we compiled a program, named pthread\_matrixmul, and provide it through the project #4 package.

you can put the precompiled program (i.e., pthread\_matrixmul) under any place of the file system you mounted. You can use the following command line to copy the multi-threaded.

```
sudo cp pthread_matrixmul imgmnt/root/
```

Once it has been done, you can unmount the disk from your host OS:

```
sudo umount imgmnt
```

After that, you can start simulation again, and run pthread\_matrixmul in the m5term as follows:

```
root@aarch64-gem5:~# ls  
pthread_matrixmul  
root@aarch64-gem5:~# chmod 777 pthread_matrixmul  
root@aarch64-gem5:~# /sbin/m5 resetstats  
root@aarch64-gem5:~# ./pthread_matrixmul 2  
root@aarch64-gem5:~# /sbin/m5 exit
```

We also provide an script named test.rcS to help you run pthread\_matrixmul.

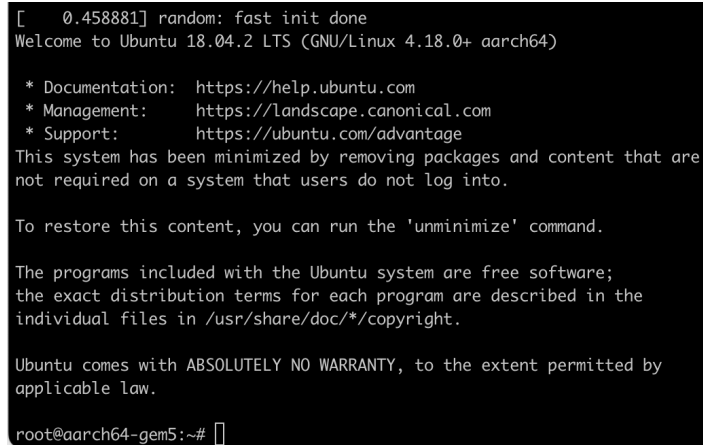
```
#!/bin/sh  
  
chmod 777 pthread_matrixmul  
  
/sbin/m5 resetstats  
  
./pthread_matrixmul 2  
  
/sbin/m5 exit
```

Using this script, you can run your simulation as like this:

```
build/ARM/gem5.opt configs/example/fs.py --kernel=vmlinux.arm64 --disk-image=ubuntu-18.04-arm64-docker.img --num-cpu=8 -script=test.rcS
```

## Part 2 deliverables:

1. A screenshot of m5term that shows you run full system successfully, an example is shown below.



```
[ 0.458881] random: fast init done
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0+ aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@aarch64-gem5:~#
```

2. Simulate “pthread\_matrixmul” workload with FS mode. Please show a plot that offers the speedup of the workload with varying numbers of threads that range from 1 to 8 (1,2,3,..., by change the number after “./pthread\_matrixmul”, i.e., 2 in the example). To get the speedup, you can normalize all the results to that of single thread program execution. To figure out execution time result, you can check simTicks described in the stats.txt file.