Project #1

(Report due is 11:59:59 PM, 9/25/2022)

Project description:
Nowadays, Gem5 has been the most popular simulation framework that computer architecture research community uses. This project will cover the tutorial of building up a computer system framework by using Gem5 simulator. Also, we will study how different CPU types and main memory systems can impact the whole performance of computer system. For your convenience, we have tried to minimize the hassle of setting up Gem5, but still, it is somewhat tricky. Please prepare Gem5 working early. 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. The titles of your email AND attachment should both be Student ID_Name_Proj1 (e.g., 123456789_Klee_Proj1).
- DO NOT PLAGIARIZE. After each project, 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

The first thing you need to do is to install all the prerequisites of Gem5 by running the following commands:

```
sudo apt-get update; sudo apt-get upgrade

sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev proto
buf-compiler libprotoc-dev libgoogle-perftools-dev python3-dev python-is-python3 libb
oost-all-dev pkg-config
```

scons: Gem5 uses SCons as its build environment. It works like make and uses Python scripts for build process.
protobuf: Protocol Buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. In Gem5, the protobuf library is used for trace generation.

After that, you should clone the codes of Gem5. You can simply check out a copy of Gem5 from the repository using the following command:

```
git clone https://github.com/gem5/gem5.git
```

```
yi@server-node3:~/labs$ git clone https://github.com/gem5/gem5.git
正克隆到 'gem5'...
remote: Enumerating objects: 261450, done.
remote: Counting objects: 100% (3550/3550), done.
remote: Compressing objects: 100% (1125/1125), done.
remote: Total 261450 (delta 2442), reused 3415 (delta 2395), pack-reused 257900
接收对象中：100% (261450/261450), 255.66 MiB | 3.34 MiB/s, 完成.
处理 delta 中：100% (143785/143785), 完成.
```

Gem5 uses scons to manage compiling (it's similar to make). In this lab, we'll compile for ARM ISA. Note that

```
cd gem5
scons build/ARM/gem5.opt -j `grep -c '^processor' /proc/cpuinfo`
```

Building gem5 may take you 30 minutes or more time on your laptop. Gem5 must be built on a Unix platform (e.g., Linux and MacOS). If it completes successfully, it will print the following results.

```
 [     CXX] ARM/base/date.cc -> .o
 [    LINK]  -> ARM/gem5.opt
scons: done building targets.
```

You now have a binary in directory build/ARM called “gem5.opt”.

Part 1 deliverables:

1. A screenshot that shows your Gem5 can compile correctly.

Part 2: Create and Run a Simple Configuration Script

Gem5 simulator is built from a collection of python objects, *SimObjects*. In this section, you need to set up a simple configuration script to describe the *SimObjects* to be instantiated, their parameters, and their relationships. Following the instructions in this section, you are able to model a simple computer system, including a simple CPU core, a single DDR4 memory channel, and a system-wide memory bus connecting CPU core and memory channel.

Let’s get started by creating and opening a new configuration file:

```
mkdir configs/proj1
```

```
vim configs/proj1/simple.py
```

This configuration file is a normal python file. Therefore, you can use any legal grammer or available libraries in Python.

The first thing we should write in the script is to import the class definitions from the m5 modules and all *SimObjects* as follows:

```
import m5

from m5.objects import *
```

In Gem5, to simplify the description of large systems, the overall simulation specification is organized as a tree. Each node in the tree is a SimObject instance. Only the instantiated SimObject which is part of the tree hierarchy will be constructed for the simulation. The program must create a special object root of class *Root* to identify the root of the tree hierarchy. When parsing the configuration program, the tree hierarchy is walked recursively to identify the objects from the root to its descendants. We will create the first object which is the root and specify its parameters:

```
root = Root()

root.full_system = False

root.system = System()
```

Gem5 provides two simulation mode: system call emulation (SE) mode and full system (FS) mode. In FS mode, Gem5 can simulate a complete system with devices and an operating system, while only essential system services are provided by simulator to execute workloads in SE mode. In this project, we will select SE mode as our simulation mode (*root.full_system = False*). FS mode will be discussed in later project. The above codes also instantiate class *System*. The *System* object is the parent of all the other objects in the simulated system. The *System* object contains a lot of functional information, such as the physical memory ranges, the root clock domain, the root voltage domain, etc.
For your information, you don't need to instantiate the python class and then specify its parameters. An alternative way is to pass the parameters as named arguments as follows:

```
root = Root(full_system = False, system = System())
```

Now that a reference to the system for simulation has been created, let's set the clock on the system. We firstly need to create the clock domain, which is an instance of class *SrcCLockDomain* (For more information of class *SrcCLockDomain*, you can refer to src/sim/ClockDomain.py). Clock domain contains multiple parameters such as clock (clock frequency), voltage domain (voltage), etc. Each parameter has its default value defined in the python class. We also can manually set the values in the configuration script as follows:

```
root.system.clk_domain = SrcClockDomain()

root.system.clk_domain.clock = '2GHz'

root.system.clk_domain.voltage_domain = VoltageDomain()
```

Once the clock domain is set up, we can move to create the memory and the memory controller. We need to configure the memory simulation mode and memory range. Since we make configurations for normal simulation, we are going to select *timing* mode for the memory simulation. *timing* mode can be applied in most conditions. In other simulation conditions such as restoring memory system from a checkpoint, you may select *atomic* mode. Then, we will also set up a single memory range of size 2GB. For the memory controller configuration, we should get an instance from various memory controller classes such as DDR4 DRAM (DDR4_2400_16x4), HMC (HMC_2500_x32), etc. All available memory controller classes are listed in src/mem/DRAMInterface.py. Since we define single memory range, we will assign this memory range to memory controller. We also need to create a bus between CPU core and memory, and connect the memory port to the bus, detailed information will be explained later.

```
root.system.mem_mode = 'timing'
root.system.mem_ranges = [AddrRange ('2GB')]
root.system.mem_ctrl = MemCtrl()
root.system.mem_ctrl.dram = DDR4_2400_16x4 ()
root.system.mem_ctrl.dram.range = root.system.mem_ranges[0]
```

So far, we have almost completed memory setup. Now, we can create a CPU. Gem5 provides various types of CPU such as SimpleCPU (simplified CPU timing model), InOrderCPU (in-order CPU), O3CPU (out-of-order CPU), etc (For more information regarding CPU model implementation, please refer to website https://www.gem5.org/documentation/general_docs/cpu_models/). As an example, we choose simple timing-based CPU in Gem5, *TimingSimpleCPU*. This CPU model executes each instruction in a single clock cycle, except for memory requests, which flow through the memory. You can instantiate the object:

```
root.system.cpu = TimingSimpleCPU()
```

Next, we need to create a system bus to connect between cpu and memory controller. The command to instantiate memory bus is as follows:

```
root.system.membus = SystemXBar()
```

For your information, Gem5 provides various system bus classes, which are written in src/mem/XBar.py.

Now that we have created the system bus, the next step is to describe the interconnect logic. For cpu, we need to connect the ICache and DCache ports to the system bus. And for memory, we need to connect memory controller port to the system bus. In addition, we need to create a Interrupt controller on the CPU and connect a system port to the memory bus. This port is a functional-only port to allow the system to read/write memory. (For your information, X86 ISA also requires connecting PIO and interrupt ports to the memory bus)

```
root.system.cpu.icache_port = root.system.membus.cpu_side_ports
```

```
root.system.cpu.dcache_port = root.system.membus.cpu_side_ports
root.system.mem_ctrl.port = root.system.membus.mem_side_ports
root.system.cpu.createInterruptController()
root.system.system_port = root.system.membus.cpu_side_ports
# root.system.cpu.interrupts[0].pio = system.membus.mem_side_ports
# root.system.cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
# root.system.cpu.interrupts[0].int_responder = system.membus.mem_side_ports
```

So far, we have finished the configuration of simulated system. Next, we will set up the process for execution. In this script, we will execute a simple “Hello World” program. You can find out the compiled binary in tests/test-progs/hello/bin/arm/linux/hello. For Gem5 simulation, you also can specify any application built for ARM ISA and that’s been statically compiled. Please refer to Appedix A for compilation environment setup. For your convenience, we also attach several representative workload source codes and their compiled executable files in project #1 package. You can refer to Appedix B to find out the specific commands to execute each workload.
To inform Gem5 to execute our program, we firstly need to set system workload to the executable. Then we create a process and set the process command to the workload we want to run. This is a list structure with the executable in the first field and the arguments passed to the executable in the rest of the list. Then we set the CPU to exeucte the process and finally create the functional execution contexts in the CPU.

```
exe_path = 'tests/test-progs/hello/bin/arm/linux/hello'
root.system.workload = SEWorkload.init_compatible(exe_path)
process = Process()
process.cmd = [exe_path]
root.system.cpu.workload = process
root.system.cpu.createThreads()
```

Finally, we need to instantiate the object hierarchy by calling *instantiate()* function and pass it to the root object of the hierarchy. Once the object hierarchy has been instantiated, the actual simulation can begin. The *simulate()* function invokes the C++ event loop. By default, this function will simulate forever, or until some other factor causes the simulation loop to exit (such as the target program calling *exit()* or a CPU reaching *the max_insts_any_thread* limit). If a positive integer argument is passed to the simulate function, it will simulate at most that number of additional ticks, but may exit sooner if another cause arises first. In any case, the *simulate()* function will return an event object that represents the reason for exiting. The object can be queried via its *getCause()* method for a string explaining that reason. The specific comamnds can be as follows:

```
m5.instantiate()
```

```
exit_event = m5.simulate()

print('Exiting @ tick {} because {}'.format(m5.curTick(), exit_event.getCause()))
```

So far, we have created a simple simulation script, we are ready to run gem5. We can simply run gem5 from the Gem5 root directory as follows:

```
build/ARM/gem5.opt configs/proj1/simple.py
```

If the simulation completes successfully, the output is shown as follows:

```
yi@server-node3:~/labs/gem5$ build/ARM/gem5.opt configs/proj1/simple.py
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 Sep  9 2022 19:42:04
gem5 started Sep  9 2022 21:04:07
gem5 executing on server-node3, pid 617327
command line: build/ARM/gem5.opt configs/proj1/simple.py

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 (32768 Mbytes) does not match the address range assigned (2048 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
build/ARM/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 289397500 because exiting with last active thread context
```

Part 2 deliverables:

1. A screenshot that shows your configuration script can work correctly.

Part 3: Understand Gem5 statistics and output

After Gem5 simulation completes, by default there are three files generated in folder m5out:

```
yi@server-node3:~/labs/gem5$ ls m5out/
config.ini  config.json  stats.txt
```

config.ini: contains a list of *SimObjects* and the values of their parameters for simulatio.
config.json: same with config.ini, but in json format.
stats.txt: detailed statistics generated from Gem5 simulation. Specifically, it contains general information about the execution, which is shown as follows:

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.000289                       # Number of seconds simulated (Second)
simTicks                                    289397500                       # Number of ticks simulated (Tick)
finalTick                                   289397500                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
simFreq                                1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                  0.03                       # Real time elapsed on the host (Second)
hostTickRate                           11208768881                       # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                  2224412                       # Number of bytes of host memory used (Byte)
simInsts                                       4988                       # Number of instructions simulated (Count)
simOps                                         5770                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                 192235                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                   222282                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                         500                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage          1                       # Voltage in Volts (Volt)
system.cpu.numCycles                         578795                       # Number of cpu cycles simulated (Cycle)
system.cpu.numWorkItemsStarted                   0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                 0                       # Number of work items this cpu completed (Count)
system.cpu.exec_context.thread_0.numInsts        4988                       # Number of instructions committed (Count)
```

It contains three columns: left lists the statistics names, middle column lists the values, and the right column shows the explanation. In this project, *simSeconds*, *simInsts* are important. You can calculate your simulated computer system performance (instructions per second) by dividing *simInsts* by *sim*Seconds. You also can find out system.*cpu.numCycles* and calculate IPC (instruction per cycle) by dividing *simInsts* by system.*cpu.numCycles*.
Each *SimObject* in the simulated compute system prints its own statistics. For example, the following figure shows the statistics of memory controller:

```
system.mem_ctrl.avgPriority_cpu.inst::samples      5030.00                # Average QoS priority value for accepted requests (Count)
system.mem_ctrl.avgPriority_cpu.data::samples      1011.00                # Average QoS priority value for accepted requests (Count)
system.mem_ctrl.priorityMinLatency       0.000000017492                # per QoS priority minimum request to response latency (Second)
system.mem_ctrl.priorityMaxLatency       0.000149663992                # per QoS priority maximum request to response latency (Second)
system.mem_ctrl.numReadWriteTurnArounds             0                   # Number of turnarounds from READ to WRITE (Count)
system.mem_ctrl.numWriteReadTurnArounds             0                   # Number of turnarounds from WRITE to READ (Count)
system.mem_ctrl.numStayReadState                12966                   # Number of times bus staying in READ state (Count)
system.mem_ctrl.numStayWriteState                   0                   # Number of times bus staying in WRITE state (Count)
system.mem_ctrl.readReqs                         6091                   # Number of read requests accepted (Count)
system.mem_ctrl.writeReqs                         936                   # Number of write requests accepted (Count)
```

It generates multiple results related to bandwidth and categorizes the simulation results based on read/write operation and instruction/data.

Part 3 deliverables:

1. A screenshot that shows the statistics of CPU in stats.txt.

Part 4: Add Options in the configuration script

In this project, you are required to explore different CPU and memory configurations for different workloads. However, it is panic to edit your configuration script every time you want to test the system with different paramters. To smoothly solve this issue, you can add command-line parameters to your gem5 configuration scripts. Since the configuration script is a Python script, Gem5 can use the Python libraries that support argument parsing (e.g., optparse). For more information of optparse, please search the online Python documentation.
Adding options to configuration script requires several steps. Firstly, after importing SimObjects, you should add the following lines:

```
from optparse import OptionParser

parser = OptionParser()

parser.add_option('--o3', action="store_true")

parser.add_option('--inorder', action="store_true")

parser.add_option('--cpu_clock', type="string", default="2GHz")

parser.add_option('--ddr3_1600_8x8', action="store_true")

parser.add_option('--ddr3_2133_8x8', action="store_true")

parser.add_option('--ddr4_2400_16x4', action="store_true")

parser.add_option('--ddr4_2400_8x8', action="store_true")

parser.add_option('--lpddr2_s4_1066_1x32', action="store_true")

parser.add_option('--wideio_200_1x128', action="store_true")

parser.add_option('--lpddr3_1600_1x32', action="store_true")
```

```
(options, args) = parser.parse_args()
```

parser.add_option() includes several fields. The first field denotes the name of parameter you want to change. You can name it with different styles, such as '--cpu_clock' and '--ddr3_1600_8x8'. Please make sure the name is reasonable and understandable. The second field can define the action types. The most common action is *store*, which tells optparse to store a value in some variable. For example, optparser takes a string from the command line and stores it in an attribute of options. If you don't specify an option action, the default action is store. Another common action is handling boolean options. Optparser sets a variable to true or false when a particular option is seen. The primitives are *action="store_true"* and *action="store_false"*. As shown in above command line codes, you can select a specific memory controller by typing the option name in command line. If you are using *store* action, the third field and fourth field can be stored data type and default value. For example, the value stored in cpu-clock is set as string and the default value is 2GHz.

Next, we will pass cpu type option to cpu in the configuration script. In addition to "TimingSimpleCPU", Gem5 also provides detailed cpu models such as "DerivO3CPU" (out of order) and "MinorCPU" (in order). Unfortunately, these cpu models require the support of cache system. Since the study of cache system and its optimization will be explored in the next project, we simply leverage the modified template configuration file "two-level.py" attched in project #1 package and leave the detailed explanation next time. In two-level.py, we need to add the following command lines to configure cpu types:

```
############### modify cpu type ############################
if options.o3:
        root.system.cpu = DerivO3CPU()
elif options.inorder:
        root.system.cpu = MinorCPU()
else:
        root.system.cpu = TimingSimpleCPU()
############### modify cpu type ############################
```

Then, we need to pass the clock speed option to cpu in the configuration script. Please note that default clock domain for all compute system components is system.clk_domain. To flexibily change the clock speed only for cpu, we need to assign an independent clock domain to cpu. This can be done as follows:

```
############### modify cpu clock domain ############################
root.system.cpu_clk_domain = SrcClockDomain()
root.system.cpu_clk_domain.clock = options.cpu_clock
root.system.cpu_clk_domain.voltage_domain = VoltageDomain()
root.system.cpu.clk_domain = root.system.cpu_clk_domain
```

```
############### modify clock domain ##############################
```

The above code instantiate a new clock *cpu_clk_domain* which is attached under root.system. After define the clock frequency and voltage domain, *cpu_clk_domain* is assigned to root.system.cpu.

Next we need to pass the options to the memory controller in the configuration script. This can be done with simple assignment command lines as follows:

```
############### modify memory controller ############################
root.system.mem_ctrl.dram = DDR4_2400_16x4()
if options.ddr3_1600_8x8:
    root.system.mem_ctrl.dram = DDR3_1600_8x8()
elif options.ddr3_2133_8x8:
    root.system.mem_ctrl.dram = DDR3_2133_8x8()
elif options.ddr4_2400_16x4:
    root.system.mem_ctrl.dram = DDR4_2400_16x4()
elif options.ddr4_2400_8x8:
    root.system.mem_ctrl.dram = DDR4_2400_8x8()
elif options.lpddr2_s4_1066_1x32:
    root.system.mem_ctrl.dram = LPDDR2_S4_1066_1x32()
elif options.wideio_200_1x128:
    root.system.mem_ctrl.dram = WideIO_200_1x128()
elif options.lpddr3_1600_1x32:
    root.system.mem_ctrl.dram = LPDDR3_1600_1x32()
else:
    root.system.mem_ctrl.dram = DDR4_2400_16x4()          #default setting
############### modify memory controller ############################
```

So far, the configuration script has been modified to enable command-line parameters. You can execute Gem5 simulation by typing the following in the shell:

```
./build/ARM/gem5.opt configs/proj1/two-level.py --o3 --cpu_clock=4GHz --ddr4_2400_8x8
```

Next, we will provide some tips which can accelerate your project work.

1. Gem5 simulation is two-/three-folder slower than the native cpu execution. To get results in a reasonable time, you can require Gem5 simulator exit after executing a max

number of instructions. This can be achieved by adding the following command line in the configuration script:

```
root.system.cpu.max_insts_any_thread = 10000000 #set maximum instructions as 10000000
```

2. Please note that the default output directory is Gem5_root_directory/m5out. Running multiple Gem5 instances in parallel will override their simulation results. One solution to this is to change the output directory. This can be done by typing following commands in shell command lines:

```
./build/ARM/gem5.opt --outdir=your-output-directory configs/proj1/two-level.py
```

outdir defines the output directory the output statistics reside in.

Part 4 deliverables:

1. Modify CPU type as TimingSimpleCPU. Modify CPU clock frequency to 1.8GHz. Modify memory controller to DDR3_1600_8x8. Please calculate and show (in table or figure) IPC for each workload we provide. Please refer to Appendix B for the workload execution commands.
2. Repeat the previous experiment, but this time you should decide the dram type to achieve the best performance. It is recommended to collect IPC and memory controller bandwidth as performance metrics. Explain the reasoning behind your design choices for each workload. Present figures showing the trade-off between design choices.
3. Repeat 2) experiment, but this time you should decide the cpu type (either O3CPU or in-order CPU) and cpu clock frequency to achieve the best performance. It is recommended to use IPC and total execution time as performance metric. Please explain the reasoning behind your design choices for each workload. Present figures showing the trade-off between design choices.

Please note that Gem5 simulation may take much longer time than what you expected. You can limit the maximum number of instructions. In this project, 1 billion instructions are recommended.

Appendix A

Q: How to compile an ARM-based program binary?
A: You need to set up a cross-compile tool chain. For Ubuntu user, you can directly download a cross-compile gcc binary by typing:

```
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
$ sudo apt-get install binutils-arm-linux-gnueabi
$ sudo apt-get install libncurses5-dev
$ sudo apt-get install gcc-arm-linux-gnueabi
$ sudo apt-get install g++-arm-linux-gnueabi
```

Then, you can compile the source file XXX.c by using the following command:

```
$ arm-linux-gnueabi-gcc -static XXX.c -o XXX
```

Appendix B

```
process.cmd = ['test_bench/2MM/2mm_base']

process.cmd = ['test_bench/BFS/bfs','-f','test_bench/BFS/USA-road-d.NY.gr']

process.cmd = ['test_bench/bzip2/bzip2_base.amd64-m64-gcc42-nn','test_bench/bzip2/input.source','280']

process.cmd = ['test_bench/mcf/mcf_base.amd64-m64-gcc42-nn','test_bench/mcf/inp.in']
```