ECE4700J Computer Architecture

Summer 2024

Lab #4 First Try Architectural Simulation

Due: 11:59pm (Beijing time) Jun. 25th, 2024

Logistics:

- This lab is an individual exercise.
- Please use the discussion board on Piazza for Q&A.
- All code and generated outputs MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- This lab involves a completely new software for most of you. It takes time to do environmental setup. Do not leave it until the last minute!

Contents

1 Overview

In this lab, we will move to upper-level hardware design. But we will not implement a CPU this time. We will get familiar with a architectural simulator and play with it:)

- Study how to setup Gem5 for simple simulations.
- Study how to write your own processor configuration for your design.
- Study how to generate the test program for your target ISA (optional).

2 Getting Started with gem5!

As we have introduced in the lab lecture, gem5 is a system-level architecture simulator. In this section, we will first set it up and then try to run a program on top of the simulated computer architecture.

2.1 gem5 Installation

2.1.1 Install on JI server

JI IT has provided a virtual machine named "ECE4700J-gem5"(Ubuntu 18.04) for every student. The source code of gem5 has been downloaded at /opt directory. You can log in the virtual machine using your student number. Then you can follow the steps below to compile gem5 and run a simple demo.

1. Go to the directory where gem5 is downloaded and compile the source code. It requires

```
cd /opt/gem5
scons build/RISCV/gem5.opt -j 4
```

You could run into the note that python3 is required but we only have Python 2.7.17 on the server. You should first get python3 on the virtual machine in this case.

```
sudo apt install python3
```

After this, try compile gem5 again using the following command.

```
/usr/bin/env python3 $(which scons) build/RISCV/gem5.opt
```

2. Run a simple simulation!

```
build/RISCV/gem5.opt configs/example/se.py \
--cmd=tests/test-progs/hello/bin/riscv/linux/hello
```

You should be able to see the following output if it succeeds.

```
**** REAL SIMULATION ****
build/RISCV/sim/simulate.cc:192: info: Entering event queue @ 0.  Starting simulation...
build/RISCV/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello world!
Exiting @ tick 3306500 because exiting with last active thread context
```

Figure 1: Hello world output

2.1.2 Local Installation

Recommended OS: Ubuntu 20.04 (WSL also works).
For MacOS users, you can use the JI server "Bitfusion".
For users using other operating system, please have a check on the official website https://www.gem5.org/documentation/learning_gem5/part1/building/.

You can go the https://www.gem5.org/documentation/learning_gem5/part1/building/ for full intructions of installation. But in most cases you can just use the instructions below.

Example Commands:
We use the command in Ubuntu as an example

1. Install all the dependencies.

```
sudo apt install build-essential git m4 scons zlib1g \
zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev \
libgoogle-perftools-dev python-dev python
```

2. Clone the source code.

```
git clone https://gem5.googlesource.com/public/gem5
```

If you have issues downloading the code using the above link, try the following one.

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

3. Build gem5 by Scons. Here we are building the X86 version as you can recognize from the command. You can build similar things like ARM or RISCV versions by changing the ISA in the command below.

```
cd gem5
scons build/X86/gem5.opt -j ${NUMBER_OF_CPUs_ON_YOUR_PC}
```

4. Run a simple simulation!

```
build/X86/gem5.opt configs/example/se.py \
--cmd=tests/test-progs/hello/bin/x86/linux/hello
```

Here the gem5 binary reads in and executes the provided Python script which creates the system under test and executes the simulator. In this example, the script creates a very simple system and executes a "hello world" binary so that you can see an "hello world" printed in the stdout.

2.2 Assignment

Now that you have known the most basic stuff in gem5, let's do a little bit different thing. Given a binary program (in the starter file), please run it under a simulated RISCV system in gem5. You will need to submit a file containing the stdout after you finish the basic simulation (use `tee` to generate a file from stdout).

Hint 1 - command:
Please stick to this command cause the binary we give you will generate something useful for our grading:)
Please note that our target ISA should be RISCV here.

```
scons build/${ISA}/gem5.opt -j ${NUMBER_OF_CPUs_ON_YOUR_PC}
build/${ISA}/gem5.opt configs/example/se.py \
--param 'system.cpu[0].workload[:].release = "99.99.99"' \
--cmd=${PATH_TO_PROVIDED_BIN}/part1_test \
| tee file_to_submit_a.txt
```

After running this command, the program will suspend and wait for you input. At this moment, please type in your student id and push ENTER. Only in this way can the file containing the stdout generated. Please **only type your own student id** and do not change anything since we will use it for grading.

Hint 2 - Sample submission file:
Using the command in hint 1, you can get a file starts and finishes with the following contents respectively.

```
Global frequency set at ... ticks per second
gem5 Simulator System. ...
gem5 is copyrighted software; ...
...
Exiting @ tick ... because exiting with last active thread
context
```

3 Play with Configurations!

As is discussed in the lab lecture, the gem5 binary takes, as a parameter, a python script which sets up and executes the simulation. In the script, you create a system to simulate, create all of the components of the system, and specify all of the parameters for the system components. Then, from the script, you can begin the simulation.

This scripts can be completely user-defined. You can choose to use any valid Python code in the configuration scripts, and can define any valid parameters in the scripts. gem5 also ships with a number of default configuration scripts. Using these scripts, you can save lots of time, and put more effort on the architecture configuration explorations.

3.1 Using the default configuration scripts

In this section, we'll explore using `se.py`, one of the default configuration scripts that come with gem5. All the gem5's configuration files can be found in `configs/`. For the detailed information of all those default configuration scripts, you can always refer to the documentations on the gem5 official website. The configuration script `se.py` allows us to pass our configuration selections to gem5 through command line. We will not discuss about all the options here. To see all of the options, run the configuration script with `--help`, or read the script's source code. Most of the options are found in `Options.py`, and are registered in the function `addCommonOptions`.

First, let's simply run the hello world program without any parameters:

```
build/RISCV/gem5.opt configs/example/se.py \
--param 'system.cpu[0].workload[:].release = "99.99.99"' \
--cmd=tests/test-progs/hello/bin/riscv/linux/hello \
```

However, this isn't a very interesting simulation at all! By default, gem5 uses the atomic CPU and uses atomic memory accesses, so there's no real timing data reported! To confirm this, you can look at m5out/config.ini. The CPU is shown on line 51:

```
[system.cpu]
type=BaseAtomicSimpleCPU
children=interrupts isa mmu power_state tracer workload
branchPred=Null
checker=Null
clk_domain=system.cpu_clk_domain
cpu_id=0
do_checkpoint_insts=true
do_statistics_insts=true
```

To actually run gem5 in timing mode, we can specify a CPU type. While we're at it, we can also specify sizes for the L1 caches.

```
build/RISCV/gem5.opt configs/example/se.py \
--param 'system.cpu[0].workload[:].release = "99.99.99"' \
--cmd=tests/test-progs/hello/bin/riscv/linux/hello \
--cpu-type=TimingSimpleCPU --l1d_size=64kB \
--l1i_size=16kB
```

Now, let's check the config.ini file and make sure that these options propagated correctly to the final system. If you search m5out/config.ini for "cache", you'll find that no caches were created! Even though we specified the size of the caches, we didn't specify that the system should use caches, so they weren't created. The correct command line should be:

```
build/RISCV/gem5.opt configs/example/se.py \
--param 'system.cpu[0].workload[:].release = "99.99.99"' \
--cmd=tests/test-progs/hello/bin/riscv/linux/hello \
```

```
--cpu-type=TimingSimpleCPU --l1d_size=64kB \
--l1i_size=16kB --caches
```

On the last line of our std output, we see that the total time went from 423503000 ticks to 31093000, much faster! That is the target of architecture configuration exploration. By exploring the different combinations of configurations, we can finally find an architecture which is much better than the default one.

3.2 Assignment

In this assignment, you are expected to explore some of the configurations based on the usage introduction given above. The test program has been given to you in starter file as `part2_test`. You can start your exploration based on the following command.

```
build/RISCV/gem5.opt configs/example/se.py \
--param 'system.cpu[0].workload[:].release = "99.99.99"' \
--cmd=${PATH_TO_PROVIDED_BIN}/part2_test$ \
--caches \
--l2cache \
| tee file_to_submit_b.txt
```

You should specify all the following options for the exploration. To make everyone's life easier, we will provide you some choices for the value of each options. You can directly explore different choices and find the best combination. (Hint: Please do try over all these choices, because not all the trend are obvious.)

```
--cpu-type  {RiscvO3CPU | RiscvTimingSimpleCPU}
--l1d_size  {1kB | 2kB | 4kB}
--l1i_size  {1kB | 2kB | 4kB}
--l2_size   {16kB | 32kB}
--l1d_assoc {1 | 2 | 4 | 8}
--bp-type   {TournamentBP | LTAGE}
```

For your submission, please submit the generated file `file_to_submit_b.txt`, with the three gem5 output file `config.ini`, `config.json`, `stats.txt` under the directory `./m5out`.

4 Simulate with More Test Programs! (Optional)

4.1 Intro to Cross-Compilation

Cross-compilation is a process to generate a program running on a platform other than the platform that generates it. In gem5, we know that we can run different programs under different ISAs. But the programs compiled by your own PC may not work for the gem5 platform you built. For example, if you compile a program on your X86 PC, it cannot be run on an Arm machine. So to test different platforms you built for your

research (maybe), you may need a compiler to generate targeted binaries. And this is the cross-compiler.

In this section, you are free to use any cross compiler you like. But to make it not that hart for students without exeriences on cross-compilation, we provide the tool and instruction hints you can use here.

Link to docker cross-compilation tool: https://hub.docker.com/u/dockcross.

Installation of docker: https://docs.docker.com/desktop/.

Compilation commands: Please refer to `${GEM5_ROOT}/tests/test-progs/hello/src/Makefile.${ISA}` as a template to create your own cross-compiling makefile.

4.2 Assignment

Please write a c code and compile it into a RISCV64 binary file so that we can run it with the RISCV gem5 platform we have generated in Section 2. For the standard output of that program, it should print out your student ID.

5 Deliverables

Please compress all the following files into a single `.zip` file named with your student ID, e.g. ECE4700lab4_519370910000.zip

- Section 2.2: `file_to_submit_a.txt` file.
- Section 3.2: `file_to_submit_b.txt`, `config.ini`, `config.json`, `stats.txt` file.
- Section 4.2 (optional): A compiled binary that can be run on a RISCV gem5 platform.

Please don't edit other provided files! They will not be used during grading.

6 Grading policy

Factors	Percentage
Section 2.2	50%
Section 3.2	50%
Section 4.2	(bonus)10%

References

1. UM-SJTU JI ECE4700J SU 2022 Lab6

Acknowledgement

- Haoyang Zhang (ECE4700J SU 2022 TA)
- Gem5