IS2202- Laboration 1

Simulating Caches and RISC-V processor Cores in GEM5

Artur Podobas, Johnny Öberg

Preparations- Setting the system up

Step 0 - Create a Virtual machine for VMWare (or VirtualBox if you prefer that)

- **You will need a virtual machine with 2 processors, 32 GB of Diskspace, and 4096 MB of RAM memory during installation.**
    - **During Runtime, you will need 6.2 GB of RAM memory.**
    - **Create a Shared directory to download files and other materials into.**
- **Use the virtual-disk "ubuntu-22.04.1-desktop-amd64" during VMWare installation.**
    - **Name your virtual machine is2202 (to make it consistent with the instructions)**
    - **Once installation is finished, replace the installation disk with your computers disk.**

STEP 1 - Installing GEM5

Instructions taken from with modifications: www.gem5.org/documentation/general_docs/building

These instructions assuming that you have Ubuntu 22.04 (or equivalent) operating system installed, either natively or using a virtual machine. Native is preferable, as it is much faster than with a virtual machine.

Start by installing dependencies:

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
python3-dev libboost-all-dev pkg-config
```

Next, clone the GEM5 repository:

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

With the GEM5 repository cloned, it is now time to build the simulator for the RISC-V platform. Start by entering the cloned folder:

```
cd gem5
```

then continue to build the RISC-V platform:

```
scons build/RISCV/gem5.opt
```

Above step will take a long time. Once finished, you can either install the cross-compilation toolchain or test the GEM5 RISC-V simulator.

STEP 2 – Create the RISC-V Cross-compilation suite

Instructions taken from with modifications: http://resources.gem5.org/resources/riscv-fs

These instructions assuming that you have Ubuntu 22.04 (or equivalent) operating system installed, either natively or using a virtual machine. Native is preferable, as it is much faster than with a virtual machine.

Start by installing dependencies:

```
sudo apt-get install -y autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev libgmp-dev
gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev emacs
```

Next, clone the riscv toolchain:

```
git clone https://github.com/riscv/riscv-gnu-toolchain
```

Finally, configure it and build/install it into a directory of your choice (e.g. /home/is2202/riscv):

```
cd riscv-gnu-toolchain
mkdir build
cd build
../configure --prefix=INSTALLATION_PATH
make linux
make install
```

Finally, set your path to include INSTALLATION_PATH/bin:

```
export PATH=INSTALLATION_PATH/bin/:$PATH
```

You are now ready crosscompile applications for RISC-V.

STEP 3 - Change the system scripts and download the new image

From CANVAS, download the file “riscv-disk-img” and place it somewhere on your Shared disk. Then mount the shared folder (you will have to mount the Shared disk each time you start Ubuntu, so it might be a good idea to add this line at the end if the .bashrc-file in your root directory):

```
sudo vmhgfs-fuse .host:/Shared /home/is2202/Shared -o allow_other -o uid=1000
```

Copy the riscv-disk-image file there to the Ubuntu root

```
cp Shared/riscv-image-disk ./
```

Next, edit the file “configs/example/gem5_library/riscv-fs.py” in the GEM5 directory.

First, start by adding the following lines to the script:

```
from gem5.resources.resource import CustomDiskImageResource

image = CustomDiskImageResource(
   local_path = "/home/is2202/riscv-disk-img",
   disk_root_partition = None,
)
```

Note: In more recent versions of GEM5, "*CustomDiskImageResource*" is deprecated and you should instead use "*DiskImageResource*".

Next, locate the line with the following content:

```
disk_image=Resource("riscv-disk-img")
```

and change it to:

```
disk_image=image
```

You are now ready to run the labs.

Laboration 1- Instructions

Introduction

The laboration is designed as a walkthrough with questions that the student (read: you) need to answer. Each exercise focuses on a particular subject, and once finished, you are supported to write a laboration rapport that you hand-in on Canvas.

- A tip throughout the lab is to always save outputs from different simulations in-between runs, in order to analyze the results further.
- It is a good idea to Copy instructions from the manual and paste them in the terminal window (Right-click Paste)
- A Virtual Machine is run as a simulation on your physical machine. The gem5 tool is running a simulation emulating RISC-V instructions that are running a UCanLinux system, so in practice, we have a simulation running a simulation, which is very slow, so running Ubuntu Natively on your machine is recommended.
- If you run the Gem5 tool Natively instead of on a VM, things will speed up significantly, so you should increase the computation sets ~20 times.

Grading

The lab consists of a number of directed parts and a number of open ended parts. The directed part is compulsory for a passing grade E, the open ended parts can give you a higher grade.

Directed Part

Exercise 1: Run and analyze statistics

Our first exercise begins with booting the RISC-V system inside GEM5. Go to the base folder of RISC-V, and launch the following command:

```
./build/RISCV/gem5.opt configs/example/gem5_library/riscv-fs.py
```

This will start the simulator and boot an image (the image and kernel will be automatically downloaded). In order to interact with the simulator, open a new terminal and connect to the machine using telnet:

```
telnet localhost 3456
```

The process of booting might take a while (~5-10 minutes).

In the telnet windows, when booting is complete, you will be asked for a username and password, which is ***root//root***. Enter this, and you will be logged in to the system.

In your home directory (/root/), there is a folder called Benchmarks. Enter it and list its content:

```
cd Benchmarks && ls
```

You will see three sub-folders, corresponding to three different benchmarks. These benchmarks come from the Rodinia benchmark suite (https://www.cs.virginia.edu/rodinia/doku.php) and have been crosscompiled for this particular RISC-V system. The three benchmarks are:

1) **HotSpot3D:** is a PDE solver for simulating hotspots (temperature gradients) on 3D stackable silicon chips. Its a type of structured grid application and its used in physics.
2) **SRAD:** Is an image processing algorithms whose computation belong to the structured grid applications.
3) **LUD:** LU Decomposition is a dense linear algebra application for factorizing a matrix into two matrixes (lower and upper triangular matrixes).

Each of above applications have been extended to also interact with the GEM5 simulator, and make sure that the GEM5 simulator only collects statistics from the interesting (the computational) part of the application, ignoring non-essential things such as initialization time, command parsing, etc.

We will now launch the application HotSpot3D. Enter the HotSpot3D folder:

```
cd HotSpot3D
```

Here you will find the hotspot3D application binary called ”3D.riscv” as well as benchmark input data (in the directory ”input”).

Next, run the benchmark with the following command (you can copy the command line from the manual and right-click in the simultation window and select paste) :

Natively:
```
./3D.riscv 512 1 1 input/power_512x8 input/temp_512x8 output.out
```

VirtualMachine
```
./3D.riscv 64 1 1 input/power_64x8 input/temp_64x8 output.out
```

This will start the application, and it will run until completion. This might take a while, depending on your computer. Once the simulation finishes, the simulator will terminate. Next, locate the statistics file that was dumped for this simulation, which can be found under: ./m5out/stats.txt. Open this file and analyze it thoroughly. To understand what the statistics mean, read the following online document (https://www.gem5.org/documentation/learning_gem5/part1/gem5_stats/).

Questions to answer:

Question 1) How long time did the application execution inside the simulator system? How long time did the application execution took outside the simulator system? How much slower was a computer simulation? Why is it this much slower?

Question 2) Under gem5 there is a library called m5out. Inspect the m5out/stats.txt.

a) How many instructions was simulated (in Gem5)?
b) How many host seconds passed (on the VM Ubuntu running Gem5)?
c) How many seconds passed in the simulated machine (UcanLinux)?
d) What was the hostInstRate (on VM Ubuntu running Gem5)?
e) How much slower is it to run the simulation compared to run it Natively (compare hostSeconds with simSeconds)

Question 3) Now, repeat above steps but also for the applications SRAD and LUD. You can invoke the SRAD and the LUD applications with the following arguments:

For SRAD:

Natively or on Virtual Machine (~20-25 minutes)

```
./srad.riscv 2048 2048 0 127 0 127 1 0.5 2
```

For LUD:

Running Natively

```
./lud.riscv -s 8000 -n 1
```

Or on Virtual Machine

```
./lud.riscv -s 256 -n 1
```

You can also time your simulation if you want a quick answer to simulation time (although this also includes setup times);

```
time ./lud.riscv -s 256 -n 1
```

Exercise 2: Checkpoints

By now, you have executed three different benchmarks, and have been forced to reboot the RISC-V system three times. This has taken a long time, and you can imagine that a more complex system would take an even longer time. So for example, if you want to understand the impact on the L1-data cache on an application, and want to simulate 100 cache simulations, then this would take a long time to reboot the system on every simulation. Fortunately, there is a solution to this:

Checkpointing.

Checkpointing means that you save the state of the system at a particular point, with the possibility of returning to that state later. In GEM5, there are multiple ways of performing a checkpoint, with the easiest being making a checkpoint from inside the guest system. To do so, start the simulator and the system as previously.

After you have entered the UCanLinux machine, run the command:

```
./checkpoint.riscv
```

This is a special binary that we implemented for you to make a checkpoint inside the OS. Next, proceed to terminate the GEM5 simulation (CTRL+C will do). You will now have a checkpoint located in the folder m5out (called cpt.???????).

Now, let us resume from the checkpoint we created. Edit the riscv-fs.py file and add the following information:

```
cpoint = CustomDiskImageResource(
   local_path = /home/is2202/gem5/m5out/cpt.???????) #  PATH_TO_OUR_cpt.???? folder
```

Next, modify the following line and add the checkpoint:

```
board.set_kernel_disk_workload(
   kernel=Resource("riscv-bootloader-vmlinux-5.10"),
   checkpoint=cpoint,
   disk_image=image #Resource("riscv-disk-img"),
)
```

You may now launch the riscv-fs.py as usual, and resume from the checkpoint.

Questions to answer:

Question 4: How much faster did it take to launch from the checkpoint compared to booting the entire system?

Exercise 3: Simulating a cache memory hierarchy

Up until now, the system that you have booted did not include a cache. In this exercise, we will attach a simple cache hierarchy to our RISC-V system and explore the impact of the cache.

There are multiple different cache subsystems in GEM5. Some are very detailed, while some are more simple. Generally speaking, the more complex (and detailed) a particular cache subsystem is, the more time it will take to simulate (but the closer to reality the results will be). For the RISC-V system that you are simulating, there are three options that has been tested:

- **NoCache:** There is no cache subsystem.
- **PrivateL1PrivateL2:** Each core has a private L1 and a private L2. This is what we we will use next.
- **MESI_Two_Level:** More advanced protocol. We will not do this option in this lab.

NoCache

To set up a system with no caches, locate the system configuration file 'riscv-fs.py' and edit that one with your favorite editor:

```
emacs configs/example/gem5_library/riscv-fs.py
```

Add this line in the beginning of the system configuration file:

```
from gem5.components.cachehierarchies.classic.no_cache import NoCache
```

Next, locate the line "cache_hierarchy =…" and comment it out. Add a new variable:

```
cache_hierarchy = NoCache()
```

Rerun one of the three applications from exercise #1 of your choice and time it, e.g.:

```
time ./lud.riscv -s 256 -n 1
```

Questions to answer:

Question 5: How much slower did it run without Cache?

PrivateL1PrivateL2

We are going to setup a single PrivateL1PrivateL2 cache subsystem. To add the cache subsystem, locate the place the line "cache_hierarchy = NoCache()". Comment this line out, and uncomment the cache hierarchy you had before. We will now change this to make a cache hierarchy where we have a 2 KiB Instruction cache, a 2 KiB Data cache, and a 64 KiB L2 cache. Do this by changing the line to the following:

```
cache_hierarchy = PrivateL1PrivateL2CacheHierarchy(
    l1d_size="2KiB", l1i_size="2KiB", l2_size="64KiB",
)
```

With the new cache in place, you can now resume your previous checkpoint, and re-run your applications. When the application ends, you can inspect the statistics (`./m5out/stats.txt`) in order to see cache performance.

Questions to answer:

Question 5: Chose one of the three applications of your choice, and run them with the cache (use the checkpoint!). What was the impact on the performance compared to with the case **NoCache**? Why? How many clock cycles did it cost in Latency? Discuss and motivate your arguments with data from the *stats.txt* file. Important numbers:

l1dcaches.overallHits
l1icaches.overallHits
l1dcaches.overallMisses
l1icaches.overallMisses
l2caches.overallHits
l2caches.overallMisses

Exercise 4: Finding out the application cache working set

Your task in this section is to determine the working set size of each of the benchmarks by varying the size of the simple **PrivateL1PrivateL2** cache used in the previous section. Do so by incrementally increasing the L1D data-cache sizes and record the measurements in order to support your claim.

Questions to answer:

Question 6: Identify the cache working set size for each benchmark. Plot a graph showing how the execution time is reduced as a function of increased cache size. For each simulation you do, *save the statistics file for further analyses (see below - copy it to another location and rename it).*

Question 7: Next, for the three applications (and for the best configuration above), repeat the action by doubling the L2 cache size, and monitor the change in execution performance. How did the performance change? How well did the L2 cache absorb misses from the L1? What is the total miss latency the processor experience due to misses to the cache?

Question 8: Which benchmark has the largest working set?

Question 9: In the above measurements, compute the L1 and L2 cache hit rates. Which application has the highest L1 and L2 cache hits? Which has the lowest? Discuss the results.

Exercise 5: Multithreading and Speed-up

So far, you have only been simulating a single-core system. Next, we are going to study multi-core system.

In order to enable a multi-core system, start by opening the system configuration file in your favorite editor:

```
emacs configs/example/gem5_library/riscv-fs.py
```

Locate the file where the processor is defined:

```
processor = SimpleProcessor(
    cpu_type=CPUTypes.TIMING, isa=ISA.RISCV, num_cores=1
)
```

And change the number of cores to two (num_cores=2). You have now a dual-core system, which you can experiment with.

Next, launch the simulator and boot the image. *Note: since the number of cores changes, you will have to reboot the system and take a new snapshot/create a new configuration point.*

Questions to answer:

Question 10: Study the impact on changing the number of cores on the different applications, and plot how the performance improves with the number of cores. Discuss the results.

Changing the number of threads in the application is performed by changing ”?” when launching the benchmark:

For Hotspot:

Running Natively

```
OMP_NUM_THREADS=? ./3D.riscv 512 1 1 input/power_512x8 input/temp_512x8 output.out
```

Running on Virtual Machine

```
OMP_NUM_THREADS=? ./3D.riscv 64 1 1 input/power_64x8 input/temp_64x8 output.out
```

For SRAD:

For SRAD: `./srad.riscv 2048 2048 0 127 0 127 ? 0.5 2`

For LUD:

Running Natively

```
./lud.riscv -s 8000 -n ?
```

Running on Virtual Machine

```
./lud.riscv -s 256 -n ?
```

Question 11: One of the application have not been parallelized, and thus could not benefit from more cores. Which application was this?

Open Ended Portion: Running an application of your choice

You are now going to cross-compile your own application and investigate the impact of caches, cache size, and core count on it. You are free to chose the application, as long as it performs something non-trivial and can exploit parallelism.

Our suggestion is to try something from:

1) Rodinia benchmark suite (https://www.cs.virginia.edu/rodinia/doku.php)
2) PolyBench (https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/)
3) Bots OpenMP suite (https://github.com/bsc-pm/bots)

You will need to use the cross-compiler to compile it on the host (or inside the VM). Remember to statically link libraries (-static) when compiling. When transferring the compiled binary to the RISC-V disk system, you need to mount it, transfer it, and unmount it. For more information, read: https://www.gem5.org/documentation/general_docs/fullsystem/disks

You will need to extend the application in a few ways in order to interact with the GEM5 simulator. Most notably, you need to do the following:

(i) include the "m5ops.h" header file into your application source code,
(ii) locate the section that you want to profile (most often, the computation), and insert the following code before ("m5_work_begin(0,0);m5_reset_stats(0, 0);") and after ("m5_dump_stats(0, 0);m5_work_end(0,0);"), which will make the simulator only collect statistics in your so-called region of interest,
(iii) when compiling the application, you must statically link it against the libm5.a file (to build libm5, follow the instructions on: https://www.gem5.org/documentation/general_docs/m5ops/).

For the application that you chose, provide a report on your finding, including discussion and showing the metrics that support your statements.