CHALMERS
UNIVERSITY OF TECHNOLOGY

EDA284 Lab 1: Introduction to gem5

Contacts: Mustafa Abduljabbar, Jeckson Souza and Miquel Pericàs,
Emails: musabdu, jeckson, miquelp@chalmers.se

February 14, 2020

`gem5` is a modular and extensible platform that enables you to simulate one or more computer systems in multiple ways. It is written in C++, but most of its functionality is provided through Python wrappers, which are what you will generally deal with in the labs. `gem5` features multiple interchangeable CPU models, event-driven memory system, mutiple ISA support, and other architecture-related features (for more details, refer to https://www.gem5.org/about). The learning book for `gem5` is available on http://learning.gem5.org/book/.

By the end of this lab, you should be able to:

- understand basic `gem5` usage using a simple ARM CPU and memory system.
- read and interpret the relevant parts of the output and stat files that are generated after simulation.
- in light of a prebuilt GEneralized Matrix Multiplication (GEMM) that is optimized on different stages, co-design your hardware to benefit from the optimization phases.
- run `gem5` in full-system mode simulating a parallel BIG.little architecture, draw the roofline model of the given system, and locate your experiments on the plot.

1 Hello World with `gem5`

To save you "build" time in the lab, you are given a prebuilt `gem5` container available on https://www.dropbox.com/s/r7vuluhqjsq7twy/gem5.tar.gz?dl=0, that is tested on your lab's machines (ED3507). Using the menu on the right, click "Download" to obtain the tar file as shown in Figure 1.

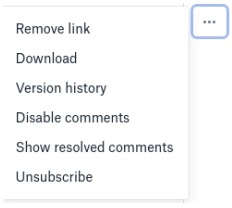

Figure 1: Downloading `gem5` container

Now, you should be able to simulate systems that target ARMv8 ISA. As mentioned, `gem5` has been built and dynamically linked on the lab machines. If you still face dependency issues when running the "Hello World" simulation on the lab machines, or if you are curious to start a clean build on your own linux system, use the instructions provided here
http://learning.gem5.org/book/part1/building.html. From now on, replace `GEM5_PATH` with the absolute path to `gem5`

Tasks:

(a) Name 4 available target ISAs that can be simulated in `gem5`.

(b) Using Part 1 of the learning book, follow the instructions to create a simple hardware configuration script. Run it using the provided `gem5` build system `./build/ARM/gem5.opt -d GEM5_PATH/m5out/YOUR_STATS_OUTPUT_FOLDER your_python_config_script`, and report the observed standard output. Notice that the -d parameter allows you to redirect the output of the simulation to a non-default folder. It can be convenient if you want to automate your simulations, as by default gem5 always overwrites the content in m5out.

(c) What are other available CPU models that can be used with your ARM system, and how do they differ?

(d) Read further on understanding the output and stat files that are generated after each simulation run. As you might have noticed, the basic configuration scripts are already available in `configs/learning_gem5/part1/`. However, it is strongly recommended to incrementally develop these scripts to help you understand the system being used.
Now, equip your system with L1/L2 instruction and data caches following the instructions in Part 1 of the book.
Use two different CPU models with your cache-enabled system and document 2-3 differences. Also, double CPU frequency in one of the model and write your observations.

(e) Attempt another memory model (instead of DDR3). Document 2 differences on the stats output.

2 Architecture-Aware Programming

You are supplied with double-precision (using 64-bit double matrix elements) GEMM binaries that are built using an ARM g++ compiler
(`aarch64-linux-gnu-g++`) and that underwent several optimization phases. The general usage of all binaries is as follows:

```
./gemm_### matrix_dim_size <0:non-transposed,1:transposed> matrix_tile_size number_of_threads
```

Note that tile size and dim size contain counts of elements (not counts of bytes).
The code, binaries and Makefile are downloadable at:
https://www.dropbox.com/s/7uzprnqdocf029w/matmul.tar.gz?dl=0. If you choose to rebuild on your linux system, you will need to download the ARM `gnu-g++` compiler
(`aarch64-linux-gnu-g++`), which is available in most linux distributions' package repositories.

Note that for all bins, transposed and non-transposed versions are available.
The differences between the binaries are listed below:
Version 1: `gemm`: is the basic naive GEMM implementation.
Version 2: `gemm_tiled`: is a tiled GEMM implementation.
Version 3: `gemm_threaded`: is an OpenMP threaded naive GEMM implementation.
Version 4: `gemm_threaded_tiled`: is an OpenMP threaded and tiled GEMM implementation.

It is advised to run always with matrix dim size 128/256 to reduce simulation time.

Tasks:

(a) On your basic system developed in Section 1 - Part b, replace the binary path in the Python script to
- run Version 1 with and without transposition. Document your observations on the stats output.
- run Version 1 and Version 2 (both with transpose enabled). Should there be differences on the stats output? why or why not?

(b) On 2-level cache system developed in Section 1 - Part d, replace the binary path in the Python script to
- run Version 2 with 2 different cache sizes. Note the impact on the stats output.
- configure your hardware with new appropriate L1 and L2 cache sizes. Observe and justify the impact in 5 cases:
1) tile size $<$ L1 cache.
2) tile size $=$ L1 cache.
3) L1 cache $<$ tile size $<$ L2 cache.
4) tile size $=$ L2 cache.
5) tile size $>$ L2 cache.

3 Your First Parallel ARMv8 BIG.little Architecture

So far, you have been using `gem5` in application-only (non-full-system) mode, where gem5 can execute a variety of architecture/OS binaries with Linux emulation on the host linux system. Using full-system mode, gem5 can model up to 64 (heterogeneous) cores of a Realview ARM platform, and boot unmodified Linux and Android with a combination of in-order and out-of-order CPUs. The ARM implementation supports 32 or 64-bit kernels and applications. For the purpose of this lab, you are provided with a linux image that is pre-loaded with the needed binaries in its file system. If you choose to create your ARM linux image to attempt your enthusiastic codes, read the documentation on http://old.gem5.org/ARM_Kernel.html and https://www.gem5.org/documentation/general_docs/fullsystem/disks. However, this step is not needed for this lab.

You will also use a modified script originally provided by ARM called fs_bigLITTLE.py. There are a few important parameters in this script that will be used in your simulations, which are the following:

- cpu-type: CPU simulation mode (more on that later)
- kernel: path to the Linux kernel file
- disk: path to the disk image file
- big-cpus: number of out-of-order cores in the system
- little-cpus: number of in-order cores in the system
- bootscript: a script to be executed after the system initialization
- caches: use caches in the simulation
- restore: restore simulation after a checkpoint (more on that later)

`gem5` allows different simulation modes that exploit various levels of details of the CPU. For instance, in the **atomic model**, no timings are counted when executing the applications (latencies and pipelines are ignored), while in the **O3CPU model** the entire out-of-order structure, along with caches and memory latencies are simulated. Needless to say, the O3CPU is more complex, and will take more time to simulate the same amount of instructions. However, if you want to evaluate the behavior of a real processor, you must use detailed CPU models. Thus, to reduce our simulation time, a helpful strategy is to simulate only the regions of interest of the application using complex (but detailed) models, and executing the rest with the fast ones.

In this lab, you will use `gem5`'s checkpoint feature to fast-forward the Linux boot operation. This operation has two main advantages:

(a) The entire boot operation can be simulated using the fast atomic model, as timings and processor state are not important during this process. From there, you can restore the simulation using a complex model, which can be used to simulate user applications and extract important statistics.

(b) After a checkpoint is created, you can restore the simulation from that point again as many times as you want. Therefore, the boot operation has to be executed only once per processor configuration.

Note: If you want to change basic configurations on the processor (such as the number of cores available in the system or the issue order), you will have to boot the system again and create a new checkpoint.

3.1 System boot, checkpoint, and restoration process.

First of all, set your `M5_PATH` to the location containing the disks, binaries and bootloader files (boot.arm and boot.arm64). To set `M5_PATH` in our case

```
export M5_PATH=GEM5_PATH/imgs/aarch-system-20170616/
```

Booting up the full system is a lengthy process, so to speed up experiments, we created a checkpoint for you that already stores the booted state of the system. We will put the steps required for initializing the system in the appendix, should you want to create a checkpoint on your own.

Now, we restore the simulation using a detailed CPU model using the following command:

```
build/ARM/gem5.opt -d GEM5_PATH/m5out/YOUR_STATS_OUTPUT_FOLDER \
configs/example/arm/fs_bigLITTLE.py \
--cpu-type exynos \
--kernel GEM5_PATH/imgs/aarch-system-20170616/binaries/vmlinux.arm64 \
--disk GEM5_PATH/imgs/aarch-system-20170616/disks/linaro-minimal-lab-aarch64.img \
--big-cpus 4 \
--little-cpus 0 \
--bootscript GEM5_PATH/imgs/aarch-system-20170616/Script.rcS \
--restore="GEM5_PATH/checkpoint/cpt.3372247250500/" \
--caches
```

Notice that in the restore/fast-forward phase here, the parameter for cpu-type is "exynos" (exynos are the detailed models for the Samsung Exynos ARM processor in the fs_bigLITTLE.py script), and there are two extra parameters:

- caches: enable caches in our simulations (required for the detailed CPU models).
- restore: restore the simulation from the given checkpoint

You will notice that your terminal will start outputting information about gem5 state. Meanwhile, open a new terminal and type:

```
telnet localhost 3456
```

The system will be available for bash usage after logging some messages. Afterwords, you will see the message:

```
INIT: no more processes left in this runlevel
bash-4.2#
```

After a full system simulation, the **stats.txt** file in your output folder will be updated with the information from that simulation. This includes cycles taken to execute OS operations such as a change directory command (cd). One way to isolate the stats of the target application is to use "m5 resetstats" before executing the app and "m5 dumpstats" after the execution, for example:

```
m5 resetstats; GOMP_CPU_AFFINITY="0,1,2,3" /home/root/gemm_### args; m5 dumpstats.
```

Notes:
– The pre-compiled GEMM binaries are located in the /home/root/ directory of the booted linux kernel's file system.
– For multithreaded runs, setting the affinity `GOMP_CPU_AFFINITY="0,1,2,3"` is required to get the desired speed-up .
– `m5 resetstats` will reset all the currently collected statistics of the system (cycles, instruction

count, memory misses...), while `m5 dumpstats` will write in the stats.txt file the current system statistics. For more special commands, type "m5" in your telnet terminal.
– When you close the system using `m5 exit`, the stats file will be appended by the another snapshot of stats. What matters is the first snapshot (i.e., the one that was generated after you called `m5 dumpstats`.

Tasks:

(a) Develop a roofline model for the system being used. Make sure you explain how you arrived at the different parameters that constitute your model.

(b) Run Version 3 of the code with 4 threads on the system image you are connected to, close and collect your stats file and plot the point on the model.

(c) Run Version 4 of the code with 4 threads on the system image you are connected to, close and collect your stats file and plot the point on the model. Comment on the bounds of your algorithm given your observations and the roofline model.

Report Submission

- You can submit the report as a group of two. Write down CIDs of each partner.
- You will get a PASS only if all the tasks are properly addressed in the report.
- The reports should be submitted on Canvas, no later then **Thursday, February 27th, 2020 23:59**

Appendix

To initialize the gem5 simulation:

```
build/ARM/gem5.opt -d GEM5_PATH/m5out/YOUR_STATS_OUTPUT_FOLDER \
configs/example/arm/fs_bigLITTLE.py \
--cpu-type atomic \
--kernel GEM5_PATH/imgs/aarch-system-20170616/binaries/vmlinux.arm64 \
--disk GEM5_PATH/imgs/aarch-system-20170616/disks/linaro-minimal-lab-aarch64.img \
--big-cpus 4 \
--little-cpus 0 \
--bootscript GEM5_PATH/imgs/aarch-system-20170616/Script.rcS
```

This command will start the simulation of the Linux kernel using the processor described by the fs_bigLITTLE.py script with two out-of-order cores.

Observe that in this stage you will use –cpu-type **atomic** to initialize our system. Make sure that the parameters –**kernel**, –**disk**, and –**bootscript** are pointing to the right directory.

You will notice that your terminal will start outputting information about gem5 state. Meanwhile, open a new terminal and type:

```
telnet localhost 3456
```

Using this telnet connection, you will now see the output of the Linux boot process. After a few minutes (the boot time vary depending on the host machine, but typically takes between 5-10 minutes using the atomic mode), the system will be available for bash usage. You will see the message:

```
INIT: no more processes left in this runlevel
bash-4.2#
```

This is the point you want to fast-forward when simulating with complex CPU models. To create a checkpoint of the current system state, type the following command:

```
m5 checkpoint
```

You will notice in the terminal of the simulator state the following message:

```
Dropping checkpoint at tick XYZ
```

XYZ represents the current simulation tick when the checkpoint creation was requested. The name of your checkpoint will be cpt.XYZ and its files will be found at your simulation output directory (in this case at '/home/YOUR_USER/gem5/m5out/lab1/config1').

You can now use the following command in your simulation terminal:

```
m5 exit
```