COEN-4730 Computer Architecture
HW #4

Dept. of Electrical and Computer Engineering, Marquette University
Cristinel Ababei

OBJECTIVE:

Learn about gem5 full system simulator and McPAT power calculator. We'll use them in hw#5, for design space exploration of chip multiprocessors (CMPs) for which the communication between cores is done via the traditional Bus.

1. GEM5 SIMULATOR

GEM5 (https://www.gem5.org/) is a computer system simulation platform. Unlike the processor architecture simulator SimpleScalar, GEM5 can perform simulations for a complete multicore platform (that is processor, memory system, operating system itself). For processors, GEM5 is capable of simulating a number of ISAs, including Alpha, ARM, MIPS, and X86.

There are two modes for GEM5 to simulate a system:

- Syscall Emulation (SE): in this mode there is no operation system (OS). All the system calls in the application is emulated by GEM5.
- Full System (FS): in this mode a complete system is simulated, including the OS and all peripherals. This is what we'll mostly use.

In GEM5 SE mode, the support for multithreaded applications is limited. Therefore, we are going to use the FS mode in this assignment and the next. In addition, we'll use the Alpha and ARM ISA's as the target architectures.

==========

1.1 Install Ubuntu 18.04

==========

Because of the changes to gem5 and perhaps to also the latest versions of Ubuntu, installing the latest gem5 and compiling builds for ARM and Alpha ISA pose memory and space challenges when working on VirtualBox. Even when the virtual HDD is 20GB and Memory 10GB building the latest version of gem5 crashed during installation due to lack of space.

So, for this assignment, we'll use a slightly older version of Ubuntu, i.e., 18.04 and also an older version of gem5. First, in your VirtualBox, delete the previous version of Ubuntu and install – following the same steps you did in hw2 – an older version of Ubuntu, that is 18.04. Download the desktop .iso from here:
https://releases.ubuntu.com/18.04/

Make sure you allocate at least 20GB of disk space and as much memory as you can afford on your computer; in my case I allocated 10GB of memory. Also, do not say yes when asked to upgrade to more recent versions of Ubuntu!

==========

1.2 Install GEM5

==========

Install prerequisites software:

Before installing GEM5, you need to first install a couple of things on your Linux Ubuntu machine. So, in a terminal, do:

> sudo apt-get update

> sudo apt-get install emacs
> sudo apt-get install build-essential
> sudo apt-get install mercurial
> sudo apt-get install python-dev
> sudo apt-get install scons
> sudo apt-get install swig
> sudo apt-get install zlib1g-dev
> sudo apt-get install m4
> sudo apt-get install protobuf{-c,}-compiler
> sudo apt-get install libgoogle-perftools-dev
> sudo apt-get install git

Download GEM5:

An older but stable version of gem5 is included with the files for this hw assignment (also placed on D2L).

The archive file is: **gem5-stable_2015_09_03.tar.gz**

Place it in some specific folder and extract it (using gunzip and then tar xvf). In my case, I have done that inside **/user/cristinel/hw4/**. You should have now created a **gem5-stable_2015_09_03/** folder with all the files of gem5. To make working with it easier, then rename it from gem5-stable to gem5.
>mkdir hw4
>cd hw4
Have **gem5-stable_2015_09_03.tar.gz** copied into hw4 folder.
>gunzip gem5-stable_2015_09_03.tar.gz
>tar xvf gem5-stable_2015_09_03.tar
>mv gem5-stable_2015_09_03 gem5

One more thing to do:
Open with a text editor (we installed emacs earlier) and edit the following file: gem5/src/SConscript
>cd gem5
>emacs src/SConscript
Search inside that file "Werror=True" and replace with Werror=False". Save your changes.

==========

1.2 Building and running FULL SYSTEM (FS) mode for ALPHA architecture

==========

Download pre-compiled Linux kernels:

In FS mode, *Linux itself is booted on the simulated architecture during the GEM5 simulation*. That is, the OS itself is simulated as well. To be able to do that we need first to download and install the full-system binary and disk image files. Linux kernels, disk images, and boot loaders for Alpha, ARM, and x86 are available on the Download page of gem5. However, for the gem5 version we use in this class, these files are included in the archive files provided for this hw assignment.

The file is: **m5_system_2.0b3.tar.bz2**

Let's download the pre-compiled Linux kernels, PALcode/Console code, and a filesystem for the Alpha architecture, and save that into a new folder, which I called **full_system_images_ALPHA** inside **gem5/**.
Expand the just downloaded archive:
>cd gem5
>mkdir full_system_images_ALPHA
>cd full_system_images_ALPHA
Place m5_system_2.0b3.tar.bz2 into folder full_system_images_ALPHA.

>tar vxfj m5_system_2.0b3.tar.bz2
>rm m5_system_2.0b3.tar.bz2
>mv m5_system_2.0b3/* .
>rm -rf m5_system_2.0b3/
>chmod –R 777 *
You should have now **binaries/** (i.e., Linux kernel) and **disk/** folders inside **full_system_images_ALPHA/** folder.

The path to these files is specified/determined in **configs/common/SysPaths.py**. That's how GEM5 simulator will know about and use them. There are a couple of default paths which are hard-coded inside this script file. Therefore, to use the newly downloaded kernel and disk files, one could do one of the following:

1-- Simply place the system files at one of those hard-coded paths. I have not done that because I created my own self-explanatory folder **full_system_images_ALPHA**

2-- Place the system files in some new folder that you create and then override the hard-coded paths in that script file by setting the **M5_PATH** environment variable. That's what I have done.
To permanently set the **M5_PATH** variable, edit your **.bashrc** file to add this line to it:
export M5_PATH=/home/cristinel/hw4/gem5/full_system_images_ALPHA

3-- Place the system files in some new folder that you create and then edit **SysPaths.py** to change those pre-defined paths to specify your new folder. That's what I have done also, just in case. You should do it too.
To change the **configs/common/SysPaths.py** I replaced:
path = ['/dist/m5/system', '/home/guoqi/simulators/gem5/dist/m5/system']
with:
path = ['/dist/m5/system', '/home/cristinel/hw4/gem5/full_system_images_ALPHA']

NOTES:
1-- If this is not done correctly you will get an error of this sort:
"ImportError: Can't find a path to system files."
when you first attempt to run the GEM5 simulator in full-system mode.
2-- If you go for option 1 above, you need root (sudo) privileges because you need to create a new folder: **/dist/m5/system** where the system files must be copied. The advantage of this option is that the files are in a place where they can be shared by multiple users.
To do that, you can follow this example to put the system files at the default location:
> sudo mkdir -p /dist/m5/system
> cd /dist/m5/system

Building:

Example 1: Now, let's build an ALPHA target of the GEM5 simulator in order to use it to be able to run FS simulations:
> scons FULL_SYSTEM=1 build/ALPHA/gem5.opt -j4

The above will create a binary which we can run to do FS simulations. If the processor architecture during your run will contain multicores, the communication between cores will be the bus-based, the traditional one.

If however, you will want for the communication to be done using a network-on-chip (NoC), then we would create the target GEM5 using the additional option RUBY=true. But, we'll not do that. We'll focus only on Bus based multicore processors.

NOTES:
1-- We can do "> scons build/ALPHA/gem5.opt --help" to see what Global build variables for ALPHA are available.
2-- We need to specify the cache-coherence protocol when building (otherwise you may not get the l2_cntrl...)

3-- If you want to change a "build" value after your GEM5 build target and configuration directory is already created, or if you want to override a value as it's created, you can specify the new values on the command line. The syntax is similar to setting environment variables at a shell prompt, but these go after the scons command.
For example, to build **MOESI_hammer** protocol for an existing ALPHA build, you could use the following command:
> scons PROTOCOL=MOESI_hammer build/ALPHA/gem5.opt
4-- We must use configuration variable FULL_SYSTEM=1 in order to build a FS target:
http://www.m5sim.org/Build_System#Per_Configuration

Running:

Example 1: Now, let's just run the **fs.py** configuration file in the gem5/configs/examples directory and utilize the kernel and disks that we downloaded earlier.

```
> build/ALPHA/gem5.opt configs/example/fs.py \
   --kernel=/home/cristinel/hw4/gem5/full_system_images_ALPHA/binaries/vmlinux \
   --disk-image=/home/cristinel/hw4/gem5/full_system_images_ALPHA/disks/linux-latest.img
```

By default, the **fs.py** script boots Linux and starts a shell on the system console.
To keep console traffic separate from simulator input and output, THIS SIMULATED CONSOLE IS ASSOCIATED WITH A TCP PORT. To interact with the console, you must connect to the port using a program such as Telnet. For example, after starting the above simulation, in a different terminal type:
> telnet localhost 3456
At the terminal prompt you got with telnet, go into / of root filesystem and type:
> ls
What do you see? You should see stuff being printed that shows the progress of Linux booting-up.

Telnet's echo behavior doesn't work well with GEM5, so if you are using the simulated console regularly, you probably want to use **m5term** instead of telnet; at least this is what GEM5 people say. So, the second way of interacting with the simulated console is to use the m5term program instead of telnet: m5term program allows the user to connect to the simulated console interface that full-system gem5 provides. To do that, go to **gem5/util/term** directory and build m5term:
> cd gem5/util/term
> make
> sudo make install
Now m5term can be used to interactively work with the simulator, though users must often set various terminal settings to get things to work.
A slightly shortened example of m5term in action:
> m5term localhost 3456
and then do as you did earlier for instance:
> ls
What do you see? You should see stuff being printed that shows the progress of Linux booting-up.

Automate benchmark simulations with .rcS file scripts:

In FS simulations, in addition to loading a Linux kernel, gem5 mounts one or more disk images for its filesystems. At least one disk image MUST BE MOUNTED AS THE ROOT FILESYSTEM. ANY APPLICATION/BENCHMARK EXECTABLE THAT YOU WANT TO RUN INSIDE GEM5 MUST BE PRESENT ON THESE DISK IMAGES!

Basically, any benchmark you will run must first be cross-compiled and made part of some disk image!

To run benchmarks without requiring an interactive shell session (as described above), which is tedious, gem5 can load **.rcS** "script" files that replace the normal Linux boot scripts to directly execute from, after booting the OS. These .rcS files can be used to configure ethernet interfaces, execute special gem5 instructions, or begin executing a binary executable that resides on the disk image.

Here is an example or Snippet of an .rcS file:

```
echo -n "setting up network..."
/sbin/ifconfig eth0 192.168.0.10 txqueuelen 1000
/sbin/ifconfig lo 127.0.0.1
echo -n "running surge client..."
/bin/bash -c "cd /benchmarks/surge && ./Surge 2 100 1 192.168.0.1 5.
echo -n "halting machine"
m5 exit
```

You can read example scripts from: **gem5/configs/boot/*.rcS**
to learn how to write your own scripts that you wish to run.

For example, this is how we would run a script, which would be executed immediately upon the Linux boot within the simulator:
> build/ALPHA/gem5.opt configs/example/fs.py --script=configs/boot/ls.rcS

NOTES: Sometimes, we can find online benchmarks already pre-compiled for certain architectures and made available for public access. These benchmarks are "prepared" to be used inside GEM5 simulations and this is done by very nice people. In such cases, we can simply download and use them as pre-compiled benchmarks inside our own GEM5 simulations. However, if you want to change the source code of these benchmarks (e.g., you make some change in some C source file that is part of the benchmark) and then run them again inside GEM5, you would need a cross-compiler to be able to re-compile the benchmark and place it on the disk-image that you use in your simulations in order to take into account changes that you make to the benchmark itself. **One example of such available benchmarks is described in the next section.**

==========

1.3 Pre-compiled PARSEC 2.1 BENCHMARKS: FS mode for ALPHA architecture

==========

NOTES:
1-- This section is written using the information available at:
http://www.cs.utexas.edu/~parsec_m5/
2--This section is the most important in the context of this homework assignment.

1) Download from the files provided for this assignment or D2L the file: **vmlinux_2.6.27-gcc_4.3.4**
Just FYI, I have it downloaded it from the above site (http://www.cs.utexas.edu/~parsec_m5/vmlinux_2.6.27-gcc_4.3.4), and it contains the "pre-built Linux 2.6.27 Kernel"
The PARSEC benchmarks available at the above site have been built to run in GEM5 full-system simulation mode.
Now place them into gem5/full_system_images_ALPHA/binaries.
>cd /home/cristinel/hw4/gem5/full_system_images_ALPHA/binaries

Download also the "disk image" and place it inside gem5/full_system_images_ALPHA/disks
Download directly from D2L the file: **linux-parsec-2-1-m5-with-test-inputs.img.bz2**
> cd /home/cristinel/hw4/gem5/full_system_images_ALPHA/disks
Here, place the "disk image" (this image contains the benchmarks themselves) downloaded from here:
http://www.cs.utexas.edu/~parsec_m5/linux-parsec-2-1-m5-with-test-inputs.img.bz2
Expand the archive:
> bunzip2 linux-parsec-2-1-m5-with-test-inputs.img.bz2
You should have now the disk image: **linux-parsec-2-1-m5-with-test-inputs.img**

2) To specify where GEM5 should look for the disk image and binaries/kernel, update the **M5_PATH** variable in your **.bashrc** by adding this line to it:
export M5_PATH=/home/cristinel/hw4/gem5/full_system_images_ALPHA
NOTE: As discussed in an earlier section you could do that by also making changes inside **configs/common/SysPaths.py**

Then, change the name of the disk image inside thr file: **configs/common/Benchmarks.py**
>cd gem5
>emacs configs/common/Benchmarks.py
Specify the path to your disk image by replacing the line:
    return env.get('LINUX_IMAGE', disk('linux-latest.img'))
with this line:
    return env.get('LINUX_IMAGE', disk('linux-parsec-2-1-m5-with-test-inputs.img'))

NOTE: If you do not want to change **Benchmarks.py**, you will have to make sure you'll run GEM5 simulator every time with the additional command line argument:
--disk-image=/home/cristinel/multicore2/gem5/full_system_images_ALPHA/disks/linux-parsec-2-1-m5-with-test-inputs.img

3) Generate .rcS running scripts **using the Perl script** provided at the same site:
http://www.cs.utexas.edu/~parsec_m5
and described in their technical report:
http://www.cs.utexas.edu/~parsec_m5/TR-09-32.pdf <--- please take a moment to read it
for each of the PARSEC benchmarks that you would like to run FS simulations.
Let's create two new folders, **benchmarks/parsec_files**, where we'll work with the above Perl script:
> cd /home/cristinel/hw4/gem5
> mkdir benchmarks
> cd benchmarks
> mkdir parsec_files
> cd parsec_files
Save here the Perl script, the input sets, and the Checkpoint script from the links below.
Actually, you can download them directly from the files included for this assignment or D2L.
http://www.cs.utexas.edu/~parsec_m5/writescripts.pl **<--- you should read this!**
http://www.cs.utexas.edu/~parsec_m5/inputsets.txt
http://www.cs.utexas.edu/~parsec_m5/hack_back_ckpt.rcS

The Perl script that we just downloaded must be run like this:
> perl writescripts.pl <benchmark> <nthreads>
So, let's plan for example to run GEM5 simulations for the benchmark called **blacksholes** on an architecture that will have **four** cores – hence I will run this script by specifying 4 for the <nthreads> option. To generate the .rcS file scripts that I will use when running GEM5, I run the Perl script for example like this:
> perl writescripts.pl blackscholes 4 <--- FOUR threads I want when I'll run this benchmark!!
This generates several running scripts, which I copy to a new folder, say **run_scripts/**:
> cd /home/cristinel/hw4/gem5/
> mkdir run_scripts
> cd run_scripts
> cp /home/cristinel/hw4/gem5/benchmarks/parsec_files/blackscholes* .
I copy them to this new folder, **run_scripts**, in order to keep things organized and to use **parsec_files/** only for running the Perl script that generates the .rcS scripts for the benchmark of interest. I will keep all .rcS file scripts for all benchmarks I am interested in inside **run_scripts/** folder.

4) Now, let's build a GEM5 target we want to simulate; in case you have not done so in the previous section. For example, let's do:

Example 1:
Let's build a GEM5 simulator instance which we'll use for simulations of multicore processor architectures that uses a **classic Bus** (not a Garnet mesh network-on-chip) as the communication between cores:
> cd gem5
> scons FULL_SYSTEM=1 build/ALPHA/gem5.opt <---you already have built that earlier! do it only if you have not!

Let's run the **blackscholes** benchmark with 4 threads on a 4 core processor.

NOTES:
1-- In the next call of GEM5 simulator I use --cpu-type=timing because the script to convert gem5 to mcpat filters out atomiccpu. Do not worry about this now.
2-- Before running .rcS script-files generated as above, you **should first comment-out the line "/sbin/m5 switchcpu" inside the blackscholes_4c_simsmall.rcS script, otherwise GEM5 will exit early.** These issues are learned as one starts using GEM5 and must use things developed by different people at different times when for example GEM5 source code and philosophy was different…

```
> build/ALPHA/gem5.opt configs/example/fs.py --cpu-type=timing \
   --script=run_scripts/blackscholes_4c_simsmall.rcS \
   --num-cpus=4 \
   --caches --l2cache --l2_size=512kB \
   --l1d_size=32kB --l1i_size=32kB --l1d_assoc=2 --l1i_assoc=2 \
   --kernel=/home/cristinel/hw4/gem5/full_system_images_ALPHA/binaries/vmlinux_2.6.27-gcc_4.3.4 \
   --disk-image=/home/cristinel/hw4/gem5/full_system_images_ALPHA/disks/linux-parsec-2-1-m5-with-test-inputs.img
```

If the above run completed successfully, you would have created the **m5out/** folder where simulation results are saved. Let's copy the result of GEM5 simulation into a new folder called **runs_results/** so that future GEM5 runs will not overwrite them:

```
> mkdir runs_results
> mv m5out runs_results/m5out_blacksholes_4c_bus
> cd runs_results/m5out_blacksholes_4c_bus
```

Open, each of the files created inside **m5out_blacksholes_4c_bus/** and study it.

Let's do one more GEM5 run of the same benchmark, but for a processor architecture with just one core (which is the default, at least at the time of this writing).

```
> build/ALPHA/gem5.opt configs/example/fs.py \
   --script=run_scripts/blackscholes_4c_simsmall.rcS \
   --kernel=/home/cristinel/hw4/gem5/full_system_images_ALPHA/binaries/vmlinux_2.6.27-gcc_4.3.4 \
   --disk-image=/home/cristinel/hw4/gem5/full_system_images_ALPHA/disks/linux-parsec-2-1-m5-with-test-inputs.img
```

By this time, you should be in good shape and have a good understanding of GEM5 simulations! Again, in this course, we'll use mostly FS simulations.

NOTES:
As discussed in the actual technical report (http://www.cs.utexas.edu/~parsec_m5/TR-09-32.pdf), generally, PARSEC benchmarks use a variety of parallelization methods including pthreads, Intel TBB, and OpenMP.
The pre-compiled PARSEC benchmarks discussed in this section, use the pthreads version of all benchmarks except for freqmine, which does not have a pthreads implementation and in that case, the OpenMP version is used.
The applications are divided into three phases:

- an initial serial phase,
- a parallel phase, and
- a final serial phase.

The parallel phase is called the **region of interest (ROI)** and is marked in the application source code by calls to the PARSEC hooks library.
Because of that, gem5 simulations will dump and reset statistics at the beginning and end of the benchmark region of interest. For example, for the simulation based on the run script above (**blackscholes_4c_simsmall.rcS** which has the line "/sbin/m5 switchcpu" **already commented out**), you should see 4 separate sets of statistics written into **m5out/stats.txt** at the end of simulation. These correspond to: 1) the start of simulation until the dump in the runscript (line "/sbin/m5 dumpstats"), 2) the beginning of the benchmark up to the beginning of the ROI in the benchmark, 3) the benchmark ROI, and 4) from the end of the ROI to when the simulation exits on "/sbin/m5 exit".
In our case we'll be interested in the third set of simulation statistics, which corresponds to the ROI portion.

2. McPAT MODELING FRAMEWORK

McPAT (Multicore Power, Area, and Timing, http://www.hpl.hp.com/research/mcpat/) is an integrated power, area, and timing modeling framework for multithreaded, multicore, and manycore architectures.
Basically, what McPAT does is to read in (micro-)architectural parameters and event statistics and to estimate the area, timing and power figures for each component of the system. It has models for different technology nodes from 90nm to 22nm. The accuracy depends on the level of details provided by the input.

Note: Until 2016 the older version McPAT 0.8 was the one that was "easier to work with" in terms of "accepting" the power.xml, created at its turn with the Python script that converts the GEM5 output to input file for McPAT. However, as of 2017, due to changes in GEM5, some issues showed up in terms of what GEM5 generates and what old McPAT 0.8 expected. Now, using latest GEM5 (stable version) and McPAT 1.0 or 1.3, together with the Python script without changes works out of the box.

Download **McPAT.tar.gz** directly from the files included for this assignment or D2L.
Install McPAT into inside gem5/ folder. Then, for simplicity, just rename the McPAT folder as mcpat.
> cd $GEM5/mcpat
> **sudo apt-get install libc6-dev-i386**
> **sudo apt-get install gcc-multilib g++-multilib**
> make
You should get created the executable "mcpat". To find out how to use McPAT, run:
> ./mcpat –help

NOTES:
Download and read documentation, technical reports, etc. that describe what McPAT is from:
http://www.hpl.hp.com/research/mcpat/McPATAlpha_TechRep.pdf

At this time we are ready to use McPAT to estimate power consumption of our architectures. For that, we need to run McPAT by providing it with the results of our GEM5 simulations. However, the files inside **m5out/** are not in a format that McPAT understands. That is why, to be able to use the output files generated by a GEM5 runs, we need to convert the GEM5's output files to the format understood by McPAT. One way to do that is to use the following script (Richard Strong's Python parser):
https://bitbucket.org/rickshin/m5-mcpat-parser
Which, I placed on D2L after some modifications to make it work.
So, download from D2L the script: **m5-mcpat-parse-se-ALPHA-ARM-Bus.py**
and place inside the gem5/gem5_2_mcpat/ folder created below.

This script is used to generate the input file needed by McPAT simulations.
The script uses as input the output files created by the GEM5 simulation, files which are saved by default under m5out/.
Let's create a new folder where we'll be using this script:
> cd $GEM5
> mkdir gem5_2_mcpat
> cd gem5_2_mcpat
Copy the result of the GEM5 run (4 cores, bus based communication):
> cp -r ../runs_results/m5out_blacksholes_4c_bus/ m5out

Place a copy of the parser into **gem5_2_mcpat/**. You can find it on D2L.
NOTE: Open the script and read it! Try to understand what it does!

Let's run the script (it knows to find the config.ini and stats.txt inside m5out/)
to generate what we need to run mcpat:
>python m5-mcpat-parse-se-ALPHA-ARM-Bus.py
If everything went ok, you should see a new file created: **power.xml**

Now, let's run McPAT to which we give as input the power.xml generated with the script:
>/home/cristinel/hw4/gem5/mcpat/mcpat -infile power.xml

If everything went ok, you should see printed at the terminal various data: area, power consumption, etc.!!!

NOTE: You may get some warnings like this "Warning: icache array structure cannot satisfy latency constraint." Try to answer what it is and why.

3. DELIVERABLES:

A single-file PDF report which shall include:
--Title, name.
--Summary of the report, listing the main things you did.
--Details of the issues you encountered during installation and while running the tools as well as how you solved them.
--Include examples of running the gem5 and McPAT tools in two Appendices.

4. REFERENCES

[1] Study on your own the GEM5 website to see how to build GEM5 for full system (FS) simulations and to have it utilize the Garnet mesh NoC. There is a lot to read; be selective and fast. Here is a list of some of the GEM5 webpages for faster study:
http://www.m5sim.org/General_Memory_System
http://www.m5sim.org/Ruby
http://www.m5sim.org/Interconnection_Network
http://gem5.org/Ruby_Network_Test

[2] Read the following webpage for more information on the simulation script (**config.py**). For example, to change the cache configuration, we can modify the settings in **config.py**.
http://www.m5sim.org/Simulation_Scripts_Explained

[3] I have put together a short PPT presentation with a “birds-eye-view” of GEM5. It contains figures with block diagrams and additional references and pointers to relevant reading materials. You can find it here:
http://dejazzer.com/coen4730/doc/lecture09_supplemental_gem5.pdf