

Before you start ...

In this document, we will introduce perf, gem5, and the benchmark package. You will need these packages for the first couple assignments.

Since perf and gem5 could only be used or compiled on a Linux system, having some basic knowledge of Linux scripting and Linux commands is highly recommended. Another thing you must figure out is how to edit text on Linux. **Vim** is a great and small tool that all Linux users should be able to use it, though it has a steep learning curve if you are coming from Windows. Recently another open source, cross-platform editor called **visual studio code** developed by Microsoft arises quickly. It can be a perfect alternative for those of you who are familiar more with Notepad++ on Windows (Notepad on Linux).

If you don't have a Linux machine, you might want to go to the computer lab in 2nd floor, BYENG217. We also encourage you to install a virtual machine in your own system. If you decide to use a virtual machine, make sure you install **VMware player** instead of VirtualBox. This is because it seems that perf can only be functionally worked in VMware player once the virtualization of PMU is enabled. Finally, you could choose Ubuntu 16.04 or 18.04 as the Linux OS in the virtual machine if you don't know how to select a Linux distribution.

I found this software website provided by ASU Fulton Schools of Engineering.

<https://onthehub.com/search>

Search for Arizona State University, and choose "Fulton Schools of Engineering". Login your ASU account, you can download a free VMWare Fusion/Workstation.

Perf

Modern CPUs usually contain limited Machine Status Register (MSR) for software developers to better understand the performance impact to the CPU microarchitecture caused by developed program so that they can further optimize their program against this specific CPU.

On Linux, perf is the most common tool to retrieve MSR's reading in high level language. Since this tool reads register value directly from CPU, it is restricted to what kind of MSR the CPU has been implemented. For example, with Intel i7-4770, perf is able to report the total number of memory write instruction, while in AMD A8-7410 this data is unavailable. But, don't worry. In the first assignment, we only need the most generic information which every CPU should've implemented.

Here is a simple tutorial about perf command where you will need for the assignment:

1. Install "linux-tools-generic" package. On Ubuntu, the command is
`sudo apt install linux-tools-common linux-tools-generic`
`linux-tools-`uname -r``
2. List available perf events under current CPU.
`sudo perf list`
you will see the output similar to the image below.

```
List of pre-defined events (to be used in -e):

branch-instructions OR branches      [Hardware event]
branch-misses                        [Hardware event]
bus-cycles                           [Hardware event]
cache-misses                         [Hardware event]
cache-references                     [Hardware event]
cpu-cycles OR cycles                 [Hardware event]
instructions                         [Hardware event]
ref-cycles                           [Hardware event]
```

3. Profile your program with perf

Let's say we want to profile our test program with cache-misses, cache-references, and cpu-cycles events. The perf command will be

```
perf stat -e cache-misses,cache-references,cpu-cycles /path/to/your/program
```

```
Performance counter stats for './test':

      2,217      cache-misses          #    5.484 % of all cache refs
     40,430     cache-references
    135,358,826   cpu-cycles

    0.035086799 seconds time elapsed
```

You must be aware that the number of MSR on a CPU is usually small, meaning that the number of event can be profiled at once is limited. Some events even need more than 1 MSR to calculate. When required number of MSR is larger than how many CPU has, perf will use multiplex mode, which use MSRs in a

time-sharing manner. When this occurs, you will see a percentage number behind a reported event like the image below.

```
Performance counter stats for 'bin/MST_opt inputs/rand-weighted-small.graph':

    1,807,141      L1-dcache-loads           (20.29%)
    133,075       L1-dcache-load-misses      #    7.36% of all L1-dcache hits
     53,082       cache-references       (81.86%)
     17,610       cache-misses          #   33.175 % of all cache refs (81.86%)

0.005290864 seconds time elapsed
```

In such mode, the perf might not be able to report accurate number. To prevent this from happening, you might want to profile target program with only few events, says 2, but in multiple times with different events combination.

In some cases, you might see a warning message says you do not have the permission for collecting stats. If you have this problem, add **sudo** before perf command to promote you with the super privilege. Be attention! If you got this problem, you also have to apply sudo to “perf list” to get a full list.

Trouble shoot on perf

Error:

You may not have permission to collect stats.

Consider tweaking /proc/sys/kernel/perf_event_paranoid:

- 1 - Not paranoid at all
- 0 - Disallow raw tracepoint access for unpriv
- 1 - Disallow cpu events for unpriv
- 2 - Disallow kernel profiling for unpriv

Solution:

```
sudo vi /etc/sysctl.conf
```

add a line “kernel.perf_event_paranoid = -1” in the file.

gem5 simulator

gem5 simulator is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture. In this instruction, you will learn how to setup and run your first test program on gem5 simulator.

For compiling GEM5, 4GB RAM/32 GB HDD is recommended, with this configuration, you can use "-j4" parameter and compile GEM5 in 30 minutes. (Tested on the VM in my MacBookPro 2012).

Required packages

We will use git to download the up-to-date gem5 source code and use gcc amd scons to build it. Following commands are just a reference for those of you who are not familiar with package management system in a Linux system.

```
sudo apt install git gcc g++ scons
```

Download gem5 source code from github using git. It will ask for your name and email if this is your first time using git.

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

This will create a folder called gem5 with all the gem5 source code in it.

On Ubuntu, you can install all of the required dependencies with the following command <http://learning.gem5.org/book/part1/building.html>.

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

Additional gem5 dependencies can be found here: <http://gem5.org/Dependencies>

Build the gem5 simulator for X86 simulation.

The building command for gem5 is

```
scons -j4 build/X86/gem5.opt
```

The string "build/X86/gem5.opt" has a special meaning other than just a filename or a pathname. Basically, it means the gem5 will be built using gcc optimization for X86 ISA simulation.

-j4 means scons will use 4 CPU cores to build. You could freely adjust this number based on how many CPU cores you have in your machine.

To get the fastest simulation speed, we should always use optimized build which has "opt" as the suffix. However, if you modified the gem5's source code(which we will do for assignment 2 and 3) and want to trace into the source code of gem5 using debugger like gdb, you need a debugging build instead of optimized build. In such case, just replace the suffix "opt" with "debug" like following command does.

```
scons -j4 build/X86/gem5.debug
```

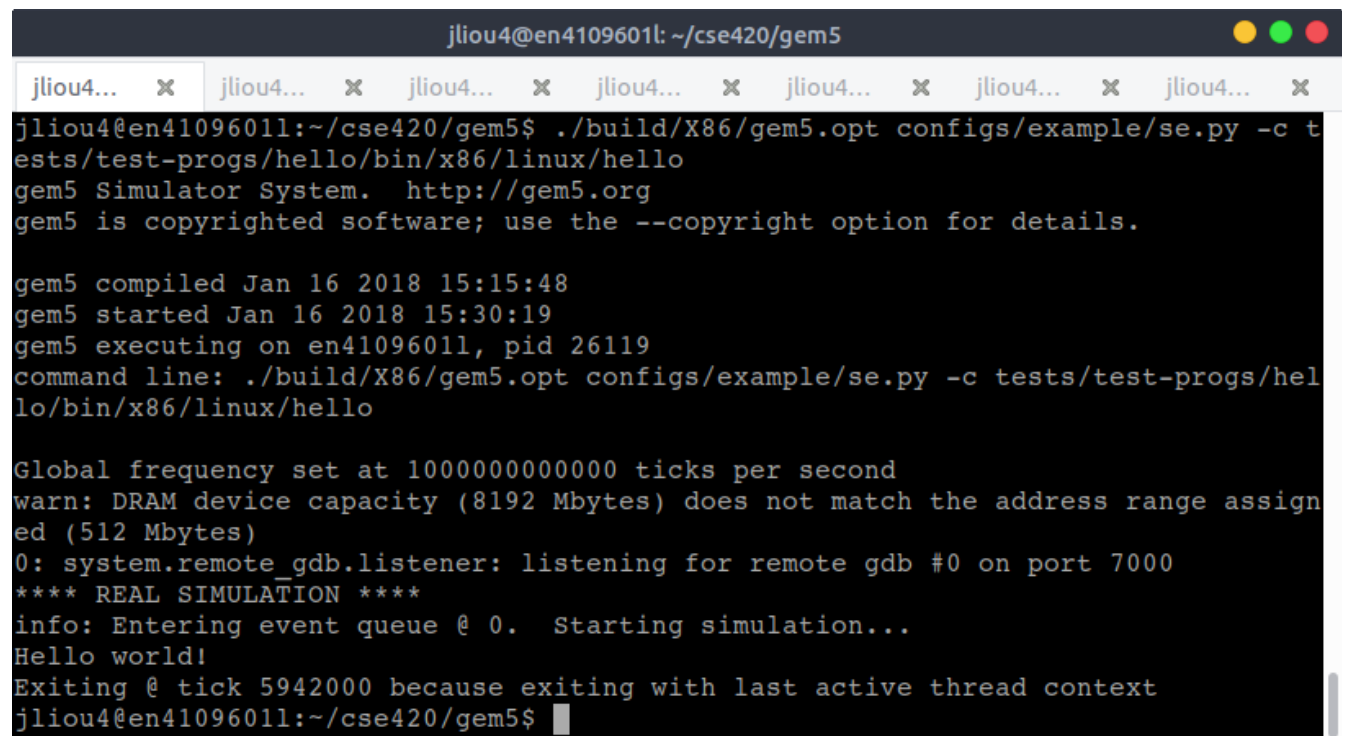
Depending on your machine performance, first time building gem5 might take 10-30 minutes. Upon finished, you should check whether the file `gem5.opt` or `gem5.debug` can be found in folder `build/X86`.

Run your program in gem5 simulator

gem5 provides two simulation environments, one called syscall emulation and another one called full system emulation. The latter one requires you to provide an OS image and boot the whole OS inside gem5 before running any benchmark on the top of that OS. This is the most detailed system simulation method but also introduces a much longer simulation time.

In this course, we only use syscall emulation so that the simulation time can be largely reduced. The config file for syscall emulation is provide in `config/example/se.py`. With using flag `-c` to specify desired program running in syscall emulation, you will get following command as your first gem5 simulation command. The test program is directly coming from gem5 under test folder

```
./build/X86/gem5.opt config/example/se.py -c  
tests/test-progs/hello/bin/x86/linux/hello
```



```
jliou4@en4109601l: ~/cse420/gem5  
jliou4... x jliou4... x jliou4... x jliou4... x jliou4... x jliou4... x jliou4... x  
jliou4@en4109601l:~/cse420/gem5$ ./build/X86/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/x86/linux/hello  
gem5 Simulator System. http://gem5.org  
gem5 is copyrighted software; use the --copyright option for details.  
  
gem5 compiled Jan 16 2018 15:15:48  
gem5 started Jan 16 2018 15:30:19  
gem5 executing on en4109601l, pid 26119  
command line: ./build/X86/gem5.opt configs/example/se.py -c tests/test-progs/hello/bin/x86/linux/hello  
  
Global frequency set at 1000000000000 ticks per second  
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)  
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000  
**** REAL SIMULATION ****  
info: Entering event queue @ 0. Starting simulation...  
Hello world!  
Exiting @ tick 5942000 because exiting with last active thread context  
jliou4@en4109601l:~/cse420/gem5$
```

Output file

Upon successfully running the test program in gem5, the following files will be created

```
m5out/config.ini  
m5out/config.json  
m5out/state.txt
```

Both config.ini and config.json state the system configuration while the stats.txt tells the simulation performance such as simulated cycles count, instruction breakdown, cache hit rate, branch prediction accuracy, and so on. For system configuration, you could simply look into config.ini if you are not familiar with json file format.

By default, gem5 uses simple, in-order CPU without any cache hierarchy. You could verify such setting in config.ini. Also, you would not find anything regarding cache and branch prediction in stats.txt.

Option for gem5 and its system configuration

In gem5 command, there are two setting groups we can play with as following

```
./build/X86/gem5.opt -help  
./build/X86/gem5.opt configs/example/se.py -help
```

The first one shows the general setting for gem5 itself such as the location for output files, while the second one is settings for the simulated system configuration inside the syscall emulation such as the CPU type and the cache size. Here is an example using both setting in one single command.

```
./build/X86/gem5.opt --outdir=tests configs/example/se.py  
--cpu-type=DerivO3CPU --caches --l1i_size=32kB --l1d_size=64kB  
--l2cache --l2_size=1MB -c tests/test-progs/hello/bin/x86/linux/hello
```

The above command indicates the system has an out-of-order CPU with both L1 and L2 caches enabled. The size for L1 instruction and data cache are 32 kb and 64 kb respectively. L2 cache is 1 MB. Besides, the gem5 will store the output file in “tests” folder instead of “m5out”.

You must take addition attention to the abbreviation of the unit of cache size. To use megabyte, both ‘M’ and ‘B’ must be upper case. But for kilobyte, ‘k’ has to be **lower** case while ‘B’ is upper case.

You should put this long command into a shell script to be easily executed and modified.

An example to expose cache replacement policy for L2 cache configuration.

For the first assignment, you are requested to change cache replacement policy in order to make the comparison between gem5 simulator and real machine. Although gem5 has already defined three replacement policies, they are not exposed through the command interface where you configure your simulation system. This section will walk you through how to expose the settings so that you can roughly understand part of gem5 infrastructure.

gem5 use hybrid language mixed with C++ and python in its infrastructure. Basically what it does is to use C++ to implement all individual architecture module but use python to expose them for a modular reason.

The generic cache structure are put under `src/mem/cache`, and the cache replacement policy are implemented under `src/mem/cache/tags`. Through `src/mem/cache/tags/Tags.py`, we

can find three cache replacement policies, LRU, RandomRepl, and FALRU(Fully associated LRU). They are BaseSetAssoc class and instantiated as a member called tags in BaseCache class. You can further look into `src/mem/cache/Cache.py` to know that Cache class inherit from BaseCache class.

Now go to `configs/common` folder and open both `Option.py` and `CacheConfig.py`, search for the arguments how you set your L2 cache size, namely `l2_size`. You will understand it changes the size of `l2_cache_class`, while `l2_cache_class` is essentially `L2Cache` class and inherited from `Cache` class. Recall `Cache` class inherit `BaseCache` class in `src/mem/cache/Cache.py`. Now you have a picture how all these classes are connected to each other.

The steps to expose the these cache replacement policies are described as below.

1. Define a new flags called `l2_tags` in `configs/common/Option.py` following the style as what `l2_size` get defined at line 106 by adding following line
2. Pass `l2_tags` from option to the `l2_cache_class` in `configs/common/CacheConfig.py` at line 87 by modifying `l2_cache_class` from

```
system.l2 =  
l2_cache_class(clk_domain=system.cpu_clk_domain,  
               size=options.l2_size,  
               assoc=options.l2_assoc)  
  
to  
  
system.l2 =  
l2_cache_class(clk_domain=system.cpu_clk_domain,  
               size=options.l2_size,  
               assoc=options.l2_assoc,  
               tags=eval(options.l2_tags))
```

We use eval function here to convert string into real python class since tags is a class, not a string. This is also the reason in previous step why we use LRU() as the default value instead of just LRU.

Now you are able to change L2 cache replacement policy by appending one of two flags below in your `gem5` command.

```
--l2_tags="LRU() "  
--l2_tags="RandomRepl() "
```

`gem5` will automatically apply FALRU if you set your cache set association equal to number of cache blocks. Directly applying FALRU to `l2_tags` with set association not equal to number of cache block will result in segmentation fault.

Trouble shoot on GEM5

Patch the emulated kernel number in `gem5`

```
cd gem5
```

open `src/arch/x86/linux/process.cc` and change

```
strcpy(name->release, "3.0.0");
```

into

```
strcpy(name->release, "5.0.0");
```

This is a workaround for problem “FATAL: kernel too old” when running benchmark in the gem5 simulator. The underlying reason is because the gem5 specifies the emulated linux kernel version in 3.0.0 while nowadays the linux kernel version has bumped to 4.x. This is just to bypass the kernel version check by defining the number larger than any known linux kernel version, and cause no real behavior change in the simulator.

Benchmark

The benchmark suite consist of 4 workloads: BFS, MST, queens, and sha. We suggest you unzip benchmark under gem5 root folder since you are going to run these workloads in gem5 for the first 3 assignments.

Following command is for you to unzip this file.

```
unzip benchmark.zip
```

Compile

Please use the following command to compile your benchmark

```
make all && make all_opt
```

All binaries are generated under bin folder after successfully compiled

Binaries with **opt** suffix are compiled with fully optimization(-O2) while the others are compiled without optimization(-O0).

Arguments for each workload

Following command provide the necessary arguments for each workload when running on the real machine. The commands are issued in the benchmark root folder.

```
bin/BFS inputs/RL5k.graph  
bin/MST inputs/rand-weighted-small.graph  
bin/sha inputs/example-sha-input.txt  
bin/queens -c 10
```

In gem5, you will issue your benchmark command as following. Pay attention to how workload arguments are passed to gem5 with -o flags.

```
build/X86/gem5.opt configs/example/se.py -c benchmark/bin/BFS -o  
"benchmark/inputs/RL5k.graph"  
build/X86/gem5.opt configs/example/se.py -c benchmark/bin/queens  
-o "-c 10"
```