Part 1: Project Description:

To evaluate the performance of cache hierarchies in multicore systems, a careful investigation is essential using simulation tools before building them. In this project, cores and caches are investigated using gem5 which is an open source tool to simulate the behavior of the specific system architecture using an event driven simulator. The project aim is to get familiar with gem5 capabilities using system call emulation for X86 ISA and be able to run known benchmarks such as GAP Benchmark Suite and generate meaningful results that can help in the design process.

Part 2: Introduction:

Gem5 is modular platform for computer system architecture research. Gem5 is considered an open source discrete event driven simulator platform that have the capability of encompassing system level architecture as well as processor level architecture. Hence, it simulates the passing of time as a series of discrete events. It can simulate an application level process detailing information down to hardware such as processor or memory. There are complex interactions and dependencies between various hardware and software elements. Elements like applications, operating system, processor, memory controllers, caches, interconnections … etc. Gem5 is presented as a solution to fully understand the behavior of these elements working together. Gem5 can cover a range of flexible, accurate and detailed system information. In addition, it can cover a design area between a high level user level view and a very detailed accurate yet non-flexible RTL tools. The tool has been provided to the public on May 2011 and since then it has been cited over 300 times. Communities that contributes to Gem5 include but not limited to MIT, ARM, AMD, HP, University of Edinburg, University of Michigan, University of Wisconsin Madison and University of Florida. The road is still paved for more contributors to join.

Part 3: Gem5 Features:

Gem5 is developed with three main goals in mind: flexibility to choose between several systems, availability for both academic and corporate researchers, and collaboration of different specialty and communities. Furthermore, Gem5 is a continuous work in progress therefore these features can be enhanced and more can be added. Gem5 essential current key features related to the project are

- Multiple interchangeable CPU models that provides handful CPU models: a simple CPU Model; a detailed in-order CPU model, and a detailed model of an out-of-order CPU.
- Event-driven memory system that features a detailed system including caches, crossbars, snoop filters, and a fast and accurate DRAM controller model, for capturing the impact of current and emerging memories. The components can be arranged flexibly to model complex multi-level non-uniform cache hierarchies with heterogeneous memories.
- A trace-based CPU model that plays back elastic traces, which are generated by a probe attached to the out-of-order CPU model. The focus of the trace CPU model is to achieve memory-system (cache-hierarchy, interconnects and main memory) performance exploration in a fast and reasonably accurate way instead of using the detailed CPU model.
- Multiple ISA support that decouples ISA semantics from its CPU models, enabling effective support of multiple ISAs. Currently, Gem5 supports x86, Alpha, ARM, SPARC, MIPS and POWER ISAs with the capability of defining a new one.

Part 4: Gem5 Modes:

Gem5 is system simulator that can run any type of application on top of an OS such as Ubuntu that runs in a specific ISA such as X86. It has two simulation models:

1. System call emulation (SE) mode which is a fast simulation, kind of plug and play, good for investigating cache hierarchy, hits, misses and associative caches to name few examples. It emulates the OS behavior but its major limitation is that it does not support thread management for binaries. In addition, applications must be statically linked to run successfully. So if you want to run threaded application such as running video compression as an approximate computing application on NoC you cannot use SE mode unless you implement thread scheduler in which you will end up with a scheduler similar to what we have in Linux OS.
2. Full system (FS) mode which runs a virtual machine that runs a full system file as a guest machine in your host. Gem5 can support multi-threaded applications and your code can be dynamically linked in FS mode. Also, you can enjoy the full thread managements that are sup- ported by Linux POSIX. However, you will need for every specific ISA:
    - Image file to store your codes or benchmarks. Your codes must be crossed compiled using a cross compiler. So, if you have x86 and you want to run codes

for ARM then you have to use an ARM cross compiler for your code to run successfully in the image.

- Linux kernel that support SMP in order to run multiple cores.

Part 5: Installing and Compiling Gem5:

This section discusses all what you need to install and run a sample exam using gem5. All the commands are tested in Ubuntu 16.04.

1. **Prerequisites:**

   Gem5 needs several prerequisites tools before you can install it. Use the following command to install them first:

   ```
   sudo apt-get install swig gcc m4 python python-dev mercurial g++
   scons build-essential zlib1g-dev protobuf-compiler libgoogle-
   perftools-dev libprotobuf-dev
   ```

2. **Install:**

   There are two ways to install gem5.

   A. Through mercurial tool using hg command as follows. This method is good for keep your version up to date and easily patched.

      ```
      hg clone http://repo.gem5.org/gem5-stable
      ```

   B. Direct download from github:

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

   Important Point:

   Gem5 is changing in a daily basis, meaning many new features are added while some are removed. To be on the same page, you need to checkout your repo to the one we have used.

   ```
   git checkout a265891e87833187e70d254a9ed09721cfce982c
   ```

3. **Compile:**

   After downloading gem5, you can build your choice of ISA the same way X86 ISA is compiles in the following lines:

   ```
   cd gem5/
   scons build/X86/gem5.opt –j10
   ```

   where –j10 represents how many cores used for compiling your ISA. Please note that this steps might take up to 20 minute based on your machine speed and how many cores did you use in -j options.

4. checking for successful installation:

To check if you compiled gen5 successfully, run the following command:

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

The result should look like the following:

```
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for
details.
...
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
0: system.remote_gdb.listener: listening for remote gdb #0 on
port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
info: Increasing stack size by one page. Hello world!
Exiting @ tick 3233000 because target called exit()
```

For help to identify more options you can try the following 2 commands:

```
%build/X86/gem5.debug -h
Usage
=====
gem5.debug [gem5 options] script.py [script options]
... Options
=======
--help, -h              show this help message and exit
--build-info, -B        Show build information
--outdir=DIR, -d DIR   Set the output directory to DIR [Default:
m5out]
--redirect-stdout, -r   Redirect stdout (& stderr, without -e)
to file
--redirect-stderr, -e   Redirect stderr to file
--stdout-file=FILE       Filename for -r redirection [Default:
simout]
--stderr-file=FILE       Filename for -e redirection [Default:
simerr]
--path=PATH[:PATH], -p PATH[:PATH]
Prepend PATH to the system path when invoking the
script

%build/ALPHA/gem5.opt configs/example/se.py -h
Usage: se.py [options] Options:
-h, --help            show this help message and exit
-c CMD, --cmd=CMD      The binary to run in syscall emulation
mode.
-o OPTIONS, --options=OPTIONS
The options to pass to the binary, use " " around the entire
string
```

```
-i INPUT, --input=INPUT
                      Read stdin from a file.
--output=OUTPUT       Redirect stdout to a file.
--errout=ERROUT       Redirect stderr to a file.
--ruby
-d, --detailed
-t, --timing
--inorder
-n NUM_CPUS, --num-cpus=NUM_CPUS
--caches
--l2cache
--clock=CLOCK
--num-dirs=NUM_DIRS
--num-l2caches=NUM_L2CACHES
--num-l3caches=NUM_L3CACHES
--l1d_size=L1D_SIZE
--l1i_size=L1I_SIZE
--l2_size=L2_SIZE
--l3_size=L3_SIZE
--l1d_assoc=L1D_ASSOC
--l1i_assoc=L1I_ASSOC
--l2_assoc=L2_ASSOC
--l3_assoc=L3_ASSOC
…
```

Part 6: GAP Benchmark Suite:

To download and compile GAP Benchmark Suite for gem5 experimentation purposes navigate the link given below:

`https://github.com/sbeamer/gapbs`

We need a single-threaded version of GAPBS. So, you need to disable the PAR_FLAG = -fopenmp in your Makefile. Disabling this parameter will generate some warnings regarding the openMP, you can simply ignore them. However, make sure your benchmarks will run successfully. For example,

./pr –u 10 –n 2 command will generate a uniform graph and do the page rank for 2 iterations. Repeat this test for all benchmarks (tc, bc,…), and include a snapshot of them in your report. Gem5 runs only static binary meaning you need to add a new flag (-static) to your Makefile.

Action items:

1. Compile GAPBS (with -static, and without -fopenmp)
2. Measure the execution times of all benchmarks for "–u 20 –n 4" input
3. Clarify the difference between –u and –g when generating a new graph

Include the answers to question above in your report.

Part 7: Project Requirement:

In this project, the cache design parameters you will modify are as follows:

- **Cache levels**: Two levels.
- **Unified caches**: Separate L1 data and L1 instruction caches, unified L2 cache. (Default)
- **Size**: 16KB, 32KB, 64KB L1 data and instruction cache, 128KB, 512KB, 1MB, 2MB unified L2 cache.
- **Associativity**: 2, 4, 8-way set-associative L1 caches, 2, 4, 8-way set-associative L2 caches.
- **Block size**: 64 bytes (applied to all caches).
- **Branch prediction**: TAGE and BiModal

While larger caches generally mean better performance, they also come at a cost. Thus, sensible design choices and trade-offs are required. To this end, in this project you will also be asked to define a cost function and to use it in order to identify the optimal configuration.

In order to specify the configuration, the following command line options are added in place of `<other options>` and should be adjusted:

`--list-cpu-types`: List available CPU types.

`--cpu-type=CPU_TYPE`: Type of CPU to run with.

`--caches`: enable cache (L1 Cache). You need this label if you want your cache specification to be utilized.

`--l2cache`: enable L2 Cache, which is similar to the above case.

`--l1d_size=L1D_SIZE`: Set size of L1 data cache.

`--l1i_size=L1I_SIZE`: Set size of L1 instruction cache.

`--l2_size=L2_SIZE`: Set size of L2 cache.

`--l1d_assoc=L1D_ASSOC`: Set associativity of L1 data cache (DCache).

`--l1i_assoc=L1I_ASSOC`: Set associativity of L1 instruction cache (ICache).

`--l2_assoc=L2_ASSOC`: Set associativity of L2 cache.

`--cacheline_size=CACHELINE_SIZE`: cache block size.

`--bp-type-list` List of all available branch prediction.

`--bp-type= BranchPrediction_Type`: Set Branch Prediction Type.

In order to execute a specific benchmark, one could use the following command:

```
$[gem5_dir]:
./build/X86/gem5.opt
                    -d ~/m5out
                    ./configs/example/se.py
                    -c ./Path_to_Benchmarks/
                    -o "-u 9 -n 5"
                    -I 100000000
                    --cpu-type= DerivO3CPU
                    --caches
                    --l2cache
                    --l1d_size=16kB --l1d_assoc=2
                    --l1i_size=16kB --l1i_assoc=2
                    --l2_size=1MB --l2_assoc=2
                    --cacheline_size=64
                    --bp-type = TAGE_SC_L_64KB
```

Execute benchmarks on "–u 9 –n 5" for 100000000 instruction in all experiments.

This defines a L1 Data cache, 2-way set-associative, with a 64-byte block, 128 sets (2*64*128=16KB). It also defines a similar Instruction cache, and a unified L2 1MB cache, 2- way set-associative, with 64-byte block, 8192 sets (2*64*8192=1MB).

The execution of this command provides the output file at "gem5/m5out/stats.txt" by default, or into your defined directory. There are many lines in the output file, each and every one of them give a specific information. For instance, we can find sim_seconds, miss rates of L1 DCache, L1 ICache and L2 Cache in the file as follows:

```
sim_seconds         0.009309           #Number of seconds simulated
system.cpu.dcache.overall_miss_rate::total 0.002566 # miss rate ...
system.cpu.icache.overall_miss_rate::total 0.000003 # miss rate ...
system.l2.overall_miss_rate::total         0.929089 # miss rate ...
```

From statics above, the sim_seconds is 0.009309, L1 DCache has a 0.2566% miss rate, L1 ICache 0.0003%, and the unified L2 Cache 92.9089%.

You can also find stats about number of cache accesses, hits and misses and many others in stats.txt.

Deliverables:

All requirement should be presented in a single PDF file, with clear tables and graphs.

Part 1:

1- Compile gem5 for X86 ISA as stated above. Show a **snapshot** of the compilation process which shows a successful compilation message. Report any issue that you might encounter.

2- For access latency, consider the following table. For the sake of simplicity we assume that both L1 and L2 caches are parallel, so associativity will not impact the access latency.

Cache Size	Access latency (cycles)
16kB, 32kB, 64kB	3
128kB	9
512kB	18
1MB	24
2MB	33

Tip 1: you can see the access latency of each cache at the following path: `Configs/common/Caches.py`

Tip 2: assume tag/response/data latency are equal.

3- Choose all 6 benchmarks of GAP Benchmark Suite and display the results in a graph for each below question.

a. For each benchmark assume L1 cache size is 16kB, L2 cache associativity is 8 ways, L1 cache associativity is 4 ways and branch prediction mode is BiModal. Explore sim_seconds for different L2 cache size (128KB, 512KB, 1MB, 2MB).

b. For each benchmark assume L2 cache size is 512kB, L2 cache associativity is 8 ways, L1 cache associativity is 4 ways and branch predictor is BiModal. Explore sim_seconds for different L1 cache size (16kB, 32KB, 64KB).

c. For each benchmark assume L2 cache size is 128kB, L2 cache associativity is 2 ways, L1 cache size is 16kB and branch predictor is BiModal. Explore sim_second for different L1 cache associativity (2-way, 4-way, 8-way).

d. For each benchmark assume L2 cache size is 128kB, L2 cache associativity is 2, L1 cache associativity is 4 ways, L1 cache size is 16kB and branch predictoris BiModal. Explore sim_seconds for different CPU type (DerivO3CPU, TimingSimpleCPU). Explain why the results are different.
e. For each benchmark assume L2 cache size is 128KB, L2 cache associativity is 4 ways, L1 cache size is 16kB and L1 cache associativity is 2 ways. Explore sim_seconds for different branch prediction modes (BiModal, TAGE).

Part 2:

Define a cost function for the caches in terms of area overhead and performance. The cost function should generally reflect the price of each design choices. For example, L1 cache should has an obvious higher price than L2 cache, while doubling the cache sizes would double the cost. The cost function should give total price for each design configuration which can serve as a rule to evaluate each one.

1. Provide an evaluation function that takes *both* sim_seconds and cost function into consideration.
2. Identify the optimal design for each benchmark in terms of both sim_seconds and cost function.
3. Present graph(s) showing the tradeoff between sim_seconds and cost for different designs.

Part 8: Main Reference:

http://gem5.org/MainPage