EE/CE/CS 6304: Computer Architecture Project #1
Dr. Kanad Basu
Due Date: March 20th, 2023

Project description: Cache design choices affect the performance of a microprocessor. In this project, you are asked to fine-tune the cache hierarchy on X86 architecture based on the gem5 simulator. The cache design parameters you can modify are as follows:

- **CPU Types**: Different CPU models.
- **Cache levels**: No cache at all, one level or two levels.
- **Size**: Cache size, one of the most important choices.
- **Associativity**: Selection of cache associativity (e.g. direct mapped, 2-way set associative, etc.).
- **Block size:** Block size of the cache, usually 64 or 32 bytes.

While larger caches generally mean better performance, they also come at a greater 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.

Part 1: Preparation

Step 0: Find a (one) teammate
Projects will be carried out in teams of two. You are asked to identify your teammate and send an email to the TA and Professor with the names and UTD NetIDs of the team members by. If you cannot find a teammate, email the TA before the above date and one will be assigned to you (cc me).

Step 1: Set up gem5 or use gem5 on server
Gem5 is the simulator we will use in this project. The gem5 simulator is a modular platform for computer architecture related research. Gem5 Wiki is a good source if you want to know more.

Gem5 Wiki: http://gem5.org/Main_Page

For this project, you have two choices for using gem5: you may choose to setup the gem5 simulator on your own PC, or you can access pre-installed gem5 simulator on our class server.

Setup Gem5 on Your PC
Steps to install gem5 are generally broken down into two steps: first, install all dependencies; second, install gem5 itself. Detailed steps can be found in following links.

Install dependencies: http://gem5.org/Dependencies
Install gem5 itself: http://gem5.org/Build_System
Particularly, if you choose to install gem5 on Ubuntu system, you have an easier way to install dependencies (sudo for access administrator access):

```
     sudo apt-get install python-dev scons m4 build-essential
g++ swig zlib-dev
                                  # ('sudo' for gaining
administrator access)
```

Access Gem5 on Server

A virtual server for the class has been set up for the purpose of carrying out the projects. The server is **ce6304.utdallas.edu**, which is accessible from on-campus machines or via UTD VPN from off-campus machines. You will need your UTD NetID and password to access it. To access the server, you can use your UTD netid and password as follows:

Log on server from Windows:

1. Open PuTTY, and find PuTTY configuration window;
2. Set host name to be "ce6304.utdallas.edu";
3. Select connection type to be "SSH";
4. Click on open;
5. Fills in your NetID and password in coming windows.

Log on server from Mac/Linux:

1. Open Terminal;
2. Type in command "`ssh <your-net-id>@ce6304.utdallas.edu`";
3. Type in your password.

The gem5 simulator is found in "**/usr/local/gem5**". You can find and use all related folders and scripts in this directory. However, you cannot write or modify files within the directory, as it requires administrator access. This means that if you want to output files from the gem5 tool, or modify an existing script, you will need to define an output directory in your own network drive, and copy scripts from the gem5 directory to your local directory.

Step 2: Benchmarks

We will use a set of CPU benchmarks to simulate your architectural design. You can download the benchmarks at the following link:

https://github.com/timberjack/Project1_SPEC.git

Or you could download the zip file from the following link:

https://github.com/timberjack/Project1_SPEC

There are a total of five benchmarks which we will simulate with:

401.bzip2
429.mcf
456.hmmer
458.sjeng
470.lbm

Within each benchmark directory, you can find a "**src**" directory that contains the source code and an executable binary, and a "**data**" directory that contains any necessary input files/arguments. Within the "**src**" directory, the executable file named "**benchmark**" is the program that you need to simulate with. You can also open the script "**run.sh**" to understand how to run the benchmark.

Within "**spec**" folder, you can also find the script "**runGem5.sh**". It is a sample script that runs gem5 on benchmark **429.mcf.** You can use this as an example how to specify the command line to run a benchmark program on gem5.

As some of the benchmark programs can take considerably long time (e.g. bzip2 can take 20 hours, and sjeng can take 10 hours), you can define a max number of instructions to be executed, e.g. a maximum of $5*10^8$ (500000000) instructions. The way to specify this constraint is discussed in the following section.

Please feel free to run your own programs or programs from the following suites to test the simulator as well:

http://euler.slu.edu/~fritts/mediabench/
http://groups.csail.mit.edu/cag/streamit/shtml/benchmarks.shtml
http://axbench.org/
https://asc.llnl.gov/CORAL-benchmarks/
http://math.nist.gov/scimark2/

Part 2: Find CPI

In this part, we will calculate the Cycles Per Instruction (CPI) for a set of benchmarks. Our baseline X86 configuration is setup as follows:

- **CPU Models**: containing TimingSimpleCPU (timing), DerivO3CPU (detailed), and MinorCPU (minor).
- **Cache levels**: Two levels.
- **Unified caches**: Separate L1 data and L1 instruction caches, unified L2 cache. (default)
- **Size**: 128KB L1 data cache, 128KB L1 instruction cache, 1MB unified L2 cache.
- **Associativity**: Two-way set-associative L1 caches, Direct-mapped L2 cache.
- **Block size**: 64 bytes (applied to all caches).
- **Block replacement policy**: Least Recent Used policy (LRU). (default)

In order to simulate the configuration, the appropriate parameters need to be provided to gem5.

There are two available simulation scripts for gem5 simulation: System Call Emulation Mode (SE) and Full System Mode (FS). We will use the SE mode in this project. For this mode, binary

file must be statically compiled. With the defined options and the SE script, a microprocessor architecture configuration is established. To execute a benchmark program using the SE mode through gem5, use the following format at the command line:

```
./build/X86/gem5.opt
     -d <output directory>    # output directory
     ./configs/example/se.py  # define system script
     -I 100000000             # max instructions
     -c <program>             # define benchmark
     -o <argument>            # arguments of benchmark
     <other options>          # settings for cache
```

Tips:
If you choose to use gem5 on server, you must define "-d" option to set your output directory to your local directory, as you have no permission to write file to server directory. If you choose to use gem5 in your own PC, "-d" option is not mandatory to you. Default output directory is at "./m5out".

In order to specify the configuration we want, we will also use the following command line options:

`--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. This setting affects all caches.

Due to a restriction of gem5, only one unified mode, which contains separated L1 cache and unified L2 cache is supported. For the cache replacement policy, LRU and Pseudo LRU policy are available. The cache replacement policy can be modified through the configuration script. For this project however, you do not need to consider changes to the replacement policy.

In order to execute the benchmarks using our configuration, we can use the following command:

```
mkdir ~/m5out
cd /usr/local/gem5
```

```
./build/X86/gem5.opt -d ~/m5out ./configs/example/se.py
-I 500000000
-c ./tests/test-progs/hello/bin/x86/linux/hello
--cpu-type=timing --caches --l2cache --l1d_size=128kB
--l1i_size=128kB --l2_size=1MB --l1d_assoc=2 --l1i_assoc=2
--l2_assoc=1 --cacheline_size=64
```

This defines a L1 Data cache, 2-way set-associative, with a 64-byte block, 1024 sets (2*64*1024=128KB). It also defines a similar Instruction cache, and a unified L2 1MB cache directed mapped, with 64-byte block, `8192 sets (2*64*8192=1MB)`. For this project, you can ignore the presence of TLBs.

The execution of this command provides the output file "**stats.txt**" under folder "**m5out**" (default, or into your defined directory). We can find miss rates of L1 DCache, L1 ICache and L2 Cache in the file.

```
system.cpu.dcache.overall_miss_rate::total     0.002790        # miss rate
for overall accesses
system.cpu.icache.overall_miss_rate::total     0.000004         # miss rate
for overall accesses
system.l2.overall_miss_rate::total           0.946058          # miss rate
for overall accesses
```

From statics above, the L1 DCache has a 0.2790% miss rate, L1 ICache 0.0004%, and the unified L2 Cache 94.6058%.

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

Deliverables: Given an L1 miss penalty of 6 cycles, L2 miss penalty of 50 cycles, and one cycle cache hit/instruction execution, use configuration parameter as above and calculate the CPI for each benchmark by following equation.

$$CPI = 1 + \frac{(IL1.miss\_num + DL1.miss\_num) \times 6 + L2.miss\_num \times 50}{Total\_Inst\_num}$$

Part 3: Optimize CPI for each benchmark

If we repeat the previous experiment, with a different configuration, we will find that many factors can influence the performance of program and cache hierarchy. By exploring the design space and trying different configurations, we will try to find an optimal configuration of cache which provides the best performance (i.e. lowest CPI). Given a base configuration of the cache as follows:

1. Separated L1 cache with size of at most 256KB in total (smaller sizes allowed)
2. Unified L2 cache, with size of 1MB (smaller sizes allowed)

You should explore the tradeoffs between different factors, **which include associativity, block size, and size allocation for L1 instruction cache and L1 data cache (they may have unequal size).**

Deliverables:
Given a two-level cache hierarchy, 256KB available for L1 cache (for L1 DCache and ICache together) and 1MB available for L2 cache：

1. Identify the optimal configuration to achieve lowest CPI for each benchmark. You should decide between and choose an associativity, block size, and size allocation for L1 instruction cache and L1 data cache.
2. Explain the reasoning about your tradeoff for each optimal configuration.
3. Present graphs showing the trade-offs between the design choices.
   # You can write scripts (python, shell) to automate the process to simulate performance of different configuration.

Part 4: Define cost function

In this part, you need to use your intuition in order to define a cost function for the caches, in terms of area overhead and performance. Obviously, larger caches are more expensive, so size should be a key parameter of the cost function. Similarly, associativity increases the cost of the cache (by adding extra hardware). L2 caches are cheaper than L1 caches because they are much slower (a larger L1 cache may be a good idea, but no designer uses it because of cost). The CPU model may also be considered.

Deliverables:
Define a cost function, in arbitrary cost units, using any parameters that you see fit.

1. 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.
2. The cost function should give total price for each design configuration which can serve as a rule to evaluate each one.

Part 5: Optimize caches for performance/cost

Given the cost function you defined in the previous part, you can now accurately select an optimal cache configuration for each benchmark, as well as all benchmarks combined.

Deliverables:
1. Provide an evaluation function that takes both CPI and cost function into consideration.
2. Identify the optimal design for each benchmark.
3. Present graphs showing the trade-off between CPI and cost for different designs → Basically plot in a graph Cost (y-axis) vs. CPI (x-axis)

Import information:

- Projects should be done in groups of 2. In exceptional cases single person or more than 2 students groups can be formed, with the explicit consent of the subject lecturer
- Allocate the marks that each group member should get on the front page of your report (e.g. 100/100 = every group member receives the same grade, or 90/110= group member two gets 10% more marks)

Deliverable Format:
- Software package with your solutions
- Power point slides, as if you were presenting to your boss having 10 minutes
- Marking:
  - Technical contents (deliverables met) = 80%
  - Clarity of results presented= 20%

UTD Honor Code
UTD values academic integrity. Therefore, all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures

Group assignments must be completed solely by the members of the group. Cross-group work is not allowed. Moreover, similar assignments have been offered before at UTD and other universities. Any use of information from previous assignments is prohibited. The tutorials are to be taken individually. Failure to respect this rule constitutes dishonesty and is a direct violation of the University Honor Code.