EE524/CPTS561 Advanced Computer Architecture, Fall 2020
Programming Assignment 1
Due date: Friday, 16 October 2020, 11:59 pm (on Blackboard)

Introduction

Cache design choices affect the performance of a microprocessor. In this Assignment, 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 Assignment you will also be asked to define a cost function and to use it in order to identify the optimal configuration.

Part 1: Set up Gem5

Step 1: Set up gem5 or use gem5 on server
Gem5 is the simulator we will use in this Assignment. The gem5 simulator is a modular platform for computer architecture related research. Gem5 documentation is very useful to learn more about gem5.

Gem5 Documentation: https://www.gem5.org/documentation/

You may set up gem5 in your laptop or over the EECS server. **sig<1-4>.eecs.wsu.edu** are configured for the use of this class. The servers have been set up with gem5 that you can use.

Setup Gem5 on Your PC or Laptop
gem5 installation can be broken down into two steps: 1) Installation of dependencies and 2) Building gem5. The steps for installation of dependencies and gem5 are outlined at the following link:

https://www.gem5.org/documentation/general_docs/building

If you are using an Ubuntu system, you can follow the following steps:

1. Install the dependencies:
   ```
   sudo apt-get install python-dev scons m4 build-essential g++ swig zlib-dev
   ```
2. Build:
   ```
   scons build/X86/gem5.opt [-j<#cpus>]
   ```

Access gem5 on Server

Four servers have also been setup for the purposes of running gem5 and completing the programming assignments. The servers are:

```
sig1.eecs.wsu.edu
sig2.eecs.wsu.edu
sig3.eecs.wsu.edu
sig4.eecs.wsu.edu
```

To access the server, you can use your EECS id and password as follows:

Log on server from Windows:

1. Open PuTTY, and find PuTTY configuration window
2. Set host name to be "sig#.eecs.wsu.edu", where # is to be replaced with 1, 2, 3, or 4
3. Select connection type to be "SSH"
4. Click on open
5. Fill in your EECS ID and password

Log on server from Mac/Linux:

1. Open Terminal;
2. Type in command "ssh <your-eecs-id>@sig1.eecs.wsu.edu";
3. Type in your password.

We have pre-installed the gem5 on the server so that you do not have to build it yourself for this assignment. The pre-built gem5 is located in **"/net/fs/gbhat/pub/gem5**". The binary of gem5, source files, and all other related files are located in this location. However, you are not able to modify the files in this directory. If you need to make any changes, please copy the directory to your own home directory and work from there. To build in your home directory, use the following command line inside your gem5 directory:

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

Step 2: Benchmarks

We will use a set of CPU benchmarks to simulate the architectural designs. The benchmarks have been set up at: https://github.com/gmbhat/EE524_CPTS561

Use the following command to download the benchmarks using git:

```
$ git clone https://github.com/gmbhat/EE524_CPTS561
```

Or you could download the zip file from the following link: https://github.com/gmbhat/EE524_CPTS561

There are total of five benchmarks that we will use:

401.bzip2
429.mcf

456.hmmer
458.sjeng
470.lbm

Each directory contains a "**src**" directory that includes the source files for the benchmark. It also contains the executable binary. The "**data**" directory contains any input files that are required for running the benchmarks. The "**benchmark**" executable found in the src directory will be used to simulate our architectural designs. The benchmark directories also contain a run script named "run.sh". It provides the proper arguments to run each benchmark.

Within "**spec**" folder, you can also find the script "**runGem5.sh**". It is a script that sets up the gem5 path and other parameters required for running the benchmark. By default, it is set to run the **429.mcf** benchmark. Please use this as an example and modify it for each of the benchmarks. Note that "**runGem5.sh**" has to be copied to each benchmark folder to ensure that the outputs are not overwritten.

Some of the benchmarks can take a long time to complete executing. In order to avoid this, we will specify the number of instructions to execute before stopping the simulations. gem5 will stop simulating and output the stats after executing the specified number of instructions. The procedure to specify the number of instructions to execute is described in the next section.

In addition to SPEC, you may experiment with other benchmark suites 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: Calculate 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)
- **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 Assignment. 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 -I
500000000             # max instructions
./configs/example/se.py  # define system script -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". Make sure to choose a different output directory for each benchmark to avoid overwriting.

In order to specify the configurations 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 Assignment 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:

```
$ cd EE525_CPTS561/429.mcf
$ /net/fs/gbhat/pub/gem5/build/X86/gem5.opt -d ./m5out
/net/fs/gbhat/pub/gem5/configs/example/se.py -c ./src/benchmark -o
./data/inp.in -I 100000000 --cpu-type=TimingSimpleCPU --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. For this assignment, you can ignore the presence of TLBs. The command also specifies that gem5 should simulate for 100000000 instructions.

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     0.002789          # 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           0.941897          # miss rate for
overall accesses
```

From statics above, the L1 DCache has a 0.2789% miss rate, L1 ICache 0.0004%, and the unified L2 Cache 94.1897% miss rate.

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

Deliverables: Given an L1 miss penalty of 5 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.

$$CPI = 1 + \frac{6 * (IL1_{misses} + DL1_{misses}) + 50 * (L2_{misses})}{Total\ instructions}$$

Part 3: Optimize CPI for each benchmark

If we repeat the previous experiment, with different configurations, 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).

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, 512KB available for L1 cache (for L1 d-cache and i-cache together) and 4MB 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, L1 data cache and L2 cache. You can vary the sizes of L1 i-cache and d-cache separately as long as it is permitted by the simulator setting. Do not exceed 512KB on L1 (in total) and 4MB on L2 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 configurations.

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).

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.

Final Deliverable Format:

- You need to upload all the scripts used, all the output files in there (for example: stats.txt, m5out.txt). This is critical because the results presented in the report will be cross checked with the output when necessary.
- Software package with your solutions
- Type a report that includes your results for each part. Include explanations of any design decisions you made, the parameters you explored. Also include tables and figures that support your results. You will be graded on the report that you submit.

Grading rubric:
Part 2 Find CPI from equation: 10 points
Part 3 Find optimal configuration: 25 points
Part 4 Define cost function: 25 points
Part 5 evaluation function: 40 points

Acknowledgements: This assignment is based, in part, on the EE6304 course at UT Dallas.