CS425: Computer Systems Architecture

Programming Assignment 3
Assignment Date: 21/12/2022
Due Date: 30/01/2023 - 23:59

Instructions: Put your answers in a .pdf file and send them together with your other deliverables via email to HY425 course e-mail (hy425@csd.uoc.gr). Set the email subject: HY425 – Programming Assignment 3.

Cache Simulation

The purpose of this assignment is to introduce you to architectural exploration using simulation and help you familiarize with some of the details of cache memories. You have to simulate caches yourself and measure their quantitative properties by applying different configurations. You will also see the impact of alternative design choices in the miss rate and IPC. For the simulation you will use the gem5 simulator (www.gem5.org). Although the assignment does not require prior experience with gem5, it is recommended that you take some time to follow the gem5 tutorial:
www.gem5.org/documentation/learning_gem5/introduction/

Simulator

In this assignment you will compile gem5 to target the x86 ISA. The build requirements can be found here: www.gem5.org/documentation/learning_gem5/part1/building/
You are able to work on this assignment solely in your own environment..
You can get a snapshot of the simulator by typing the following git command:

```
git clone -b hy425_pa3 https://github.com/sotot0/gem5.git
```

This command will automatically clone from the main repository and checkout to hy425_pa3 branch of it. You will work only on this branch.
The repository contains:

1. the gem5 directory
2. and the benchmark directory (hy425)

After you have installed the indicated dependencies, compile the simulator by typing:

```
python3 `which scons` build/X86/gem5.opt -j <Avail. Cores>
```

inside the gem5 directory. The compilation step takes time so complete it as soon as possible!

To run a batch of five (5) benchmarks with the simulator you have to use the following command inside the hy425 directory:

```
./runall.sh [List of arguments]
```

This command will run all the benchmarks for each Task below (1-3), while simulating a setup with an Out-Of-Order CPU (O3CPU) and a good performing branch predictor (TournamentBP). You are encouraged to examine thoroughly the runall.sh/README files to figure out the different arguments that you will be allowed to change. The arguments that are related to the branch predictor (TournamentBP) and CPU are not allowed to change.

In this assignment you will tweak different parameters that are only related to caches and their policies (list below). Note that the -I *[Instruction Count]* argument means that for each benchmark, gem5 simulates *[Instruction Count]* instructions. You can change this parameter during early testing but for generating final numbers you should have at-least 100 million instructions. If you set a very large number of instructions the simulation time will increase dramatically. (Even with the parameter set to 100 million instructions you may experience long simulation times especially when running benchmark1/benchmark4). Ensure that your comparisons are based on the same instruction count between different benchmarks and/or settings to extract uniform and fair results. Also, ensure that this parameter is large enough to extract meaningful results. An estimated number would be between 100-150 million instructions. In any case, this is another reason to begin the assignment as soon as possible!

Task 1: Measure L1 Miss Rate and IPC

Your first task is to explore the effects of cache-block/cache-line size and associativity in an L1 data cache. Assume that your L1 data cache budget is 32 KBytes, the cache supports the LRU replacement policy and that there is no prefetching in L1. Run experiments with varying block sizes and associativity for all the given benchmarks, draw miss-rate and IPC graphs similar to those presented in class and find the configuration that gives the highest average IPC. Also, note that the rest of the parameters must remain stable during Task 1.

Explore the following block sizes:

(i) 32 bytes, **(ii)** 64 bytes, **(iii)** 128 bytes

and the following associativity settings:

(i) direct-mapped, **(ii)** 2-way, **(iii)** 4-way

Task 2: Measure L1/L2 Combined Miss Rate and Overall IPC

At this part of the assignment, you need to figure out the best performing L2 cache size, associativity, replacement policy (only for L2) and “inclusivity” while keeping stable your previously found (best performing) L1 cache configuration. Run experiments with varying parameters for all the given benchmarks, draw global miss-rates (combination of L1 and L2) and overall IPC graphs similar to those presented in class and find the L2 configuration that gives the highest average IPC.

Explore the following L2 cache sizes:

(i) 128KB, **(ii)** 256KB , **(iii)** 512KB the following L2 associativity settings:

(i) 4-way, **(ii)** 8-way, **(iii)** 16-way

the following L2 replacement policies:

(i) LRURP, **(ii)** RandomRP, **(iii)** FIFORP

and the following L2 inclusivity options:

(i) mostly_incl, **(ii)** mostly_excl

Task 3: Measure the Impact of Hardware Prefetchers

Use the best performing L1 and L2 cache configurations that you found above, and figure out which prefetcher type and under which configuration (prefetch degree) achieves the best results when attached only on the L2 cache. Prefetchers are extremely simple hardware schemes that attempt to exploit spatial locality beyond cache-block boundaries. Typically, when a cache experiences a cache miss for line A, then a simple next-K-line prefetcher fetches the next K sequential cache-blocks, if not already present in the cache; **K** is known as the prefetch degree. An important implication of cache prefetching is the need for higher memory bandwidth. Based on the memory traffic statistics of the simulator (system.mem_ctrl.dram.bwTotal::total), find the prefetching configuration that best balances between IPC and memory bandwidth.
Explore the following prefetcher types:

(i) StridePrefetcher **(ii)** TaggedPrefetcher

With a prefetch degree in the range **1-16 next lines**

Hint: *Calculate the metric Memory Bandwidth/IPC.*

Write on your report your findings and explanations using graphs/tables. It is important to get familiar with the generated m5out directories under /benchmarkN directories after each run. These directories include files showing the simulated configuration and its statistical information.

Deliverables
You have to deliver one **zip/tar.gz** with the following files:
- Your report in PDF format
- Any custom scripts that you used
- Any other files that you find useful
- m5out directories that contain useful statistics