CS425: Computer Systems Architecture

Simulation Assignment 3
Assignment Date: 30/12/2025
Due Date: 02/02/2026 - 23:59

Cache Simulation

The purpose of this assignment is to introduce you to architectural exploration using simulation and help you familiarize with the details of multi-level cache hierarchies. 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 cache miss rates 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/

Simulation Assignment 3 – Code Repository

For the infrastructure of this assignment, we have created personal git repositories on the Department's gitlab server for each student enrolled in the course. Your personal repository is in the form:

https://gitlab-csd.datacenter.uoc.gr/hy425_2025f/sim3_submits/sim3-csdXYZWA

Get a clone of your repository with `git clone` using your username (csdXYZWA@csd.uoc.gr). Access is only available via the University network and the VPN. *(Do not fork the repo – Work directly on the cloned git repo)*.

The repo includes the following:

- `README.md` : Quick installation and run instructions
- `benchmarks/` : contains five precompiled benchmarks for you to run
- `config/` : gem5 configuration of an out-of-order core with multi-level caches
- `logs/` : this folder will contain the output statistics generated by your simulations
- `setup.sh` : Shell script for setting environment variables and PATHs
- `runall.sh` : Shell script for quickly running a gem5 simulation with the given configuration

Study the folders and the files to get an idea of the repo and feel free to create your own scripts as you see fit.

In this assignment you will compile gem5 to target the **x86 ISA**. The build requirements can be found here: https://www.gem5.org/documentation/general_docs/building

Check the **`README.md`** file which provides a quick installation and running guide. The gem5 compilation step takes time so complete it as soon as possible!

(You can reuse your previous gem5 installation from Simulation Assignment 2 and save a lot of time).

Assignment Tasks

To run a batch of five (5) benchmarks with the simulator you have to use the following command inside the `sim3-username` directory:

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

This command will run all the benchmarks 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` to figure out the different arguments that you will be 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 `--Insts` *[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 20 million instructions. If you set a very large number of instructions the simulation time will increase dramatically. 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 20-50 million instructions and this should take 2-5 minutes per benchmark; so start working on the assignment as soon as possible!

It is important to get familiar with the generated m5out directories under the `logs/benchmarkN` directories after each run. These directories include files showing the simulated configuration and their statistical information.

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 64 KBytes, the cache supports the LRU replacement policy and there is no hardware prefetcher in the L1 cache. 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 IPC. Also, note that all other parameters must remain fixed 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, **(iv)** 8-way

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

In this part of the assignment, you need to find the best performing L2 cache size, associativity, and replacement policy (only for L2) while keeping fixed the L1 cache configuration you found in Task 1 (best performing). Run experiments with varying parameters for all the given benchmarks, draw global miss-rates (combination of L1 and L2) and IPC graphs similar to those presented in class and find the L2 configuration that gives the highest IPC.

Explore the following L2 cache sizes:

(i) 256KiB, **(ii)** 512KiB

the following L2 associativity settings:

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

and the following L2 replacement policies:

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

Task 3: Measure the Impact of Hardware Prefetchers

Use the best performing L1 and L2 cache configurations that you found in Tasks 1 & 2 and find which prefetcher type and under which configuration (prefetch degree) achieves the best results when employed only on the L2 cache. Prefetchers are hardware schemes that attempt to exploit spatial locality beyond cache-block boundaries and can vary from simple to extremely sophisticated. Typically, when a cache experiences a 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.

Explore the following prefetcher types:

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

With a prefetch degree in the range of **1 up-to 4 cache-lines.**

Does hardware prefetching offer performance benefits, i.e. IPC improvements? Quantify the their performance impact.

An important implication of 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 prefetcher configuration that best balances between IPC and memory bandwidth.

Hint: Calculate the metric Memory Bandwidth/IPC.

Assignment Delivery

Report your findings and explanations (for all Tasks 1-3) in the `answers.md` file and use tables and/or graphs.

Commit and push the following files on your designated gitlab repo:

- `answers.md` with your report and comments
- All the `logs` (`benchmark1, benchmark2`, …) folders/subfolders generated by your simulations
- Any other files or scripts you consider useful

Indicative git commands:

- git status (see the changes)
- git config user.email csdXYZW@csd.uoc.gr
- git config user.name "Name Surname"
- git add answers.md logs* (add all related folders)
- git commit -m “sim3 delivery” (commit and message)
- git push (use your username and password)

Make sure that the push was successfully done and that your files have been uploaded on the gitlab repo.