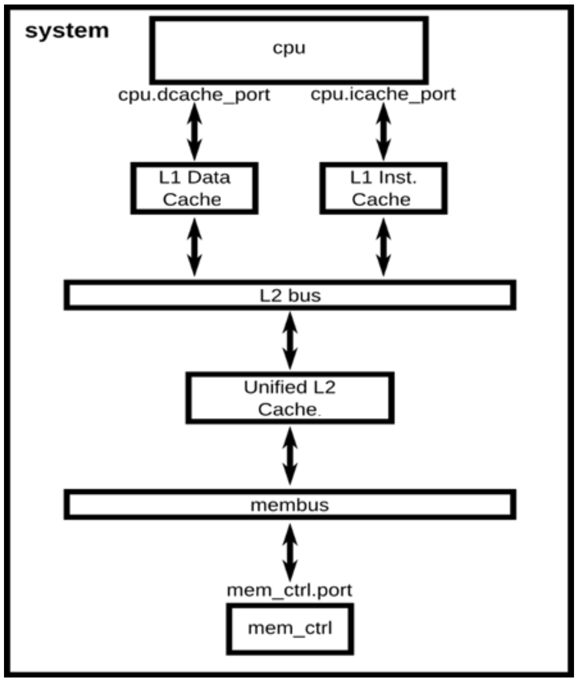
Lab 3: Adding a Cache Hierarchy to an X86 Architecture in gem5

EEL 5764 – Fall 2022



Introduction:

In this lab, you will:

Part A: Follow a gem5.com tutorial to add a cache hierarchy (L1-data, L1-instruction, and L2 caches) to the X86 architecture that you created in Lab 2 (see Figure 1). Then run "hello world" X86 binary code on it.

Part B: Create, compile, and run a matrix-multiply program (matmult.c) on the new architecture. Then make changes in the script code and observe the difference in performance .

Part C: Review "Understanding about gem5 statistics and output" tutorial again.

Figure 1. An X86 architecture with L1 and L2 caches

Part A – Adding L1 and L2 caches to the X86 architecture

In Part A, you will follow the tutorial ("Adding cache to the configuration script"") from the gem5.org website to add a cache hierarchy to the basic X86 architecture that you created in Lab 2 (Figure 1) and run the hello-world binary code (from Lab 2) on it.

https://www.gem5.org/documentation/learning_gem5/part1/cache_config/

Important notes with regards to the tutorial: (I will further explain in class)

- **You do not do the entire tutorial**
  This tutorial will teach you: Step (1) how to add a cache hierarchy to an architecture; and Step (2) to make performing experiments more flexible, how to add the capability of specifying the cache parameter values from a command line. However, we will only do Step (1) in Lab 3 and save Step (2) for Lab 4. Thus, in this lab, you will follow the tutorial to perform the following:
  - Define cache objects with specific (fixed) parameters (caches.py)
  - Modify the configuration script (simple.py from Lab 2) to add the cache components into the X86 architecture. To do so, you are instructed to copy simple.py (from Lab 2) to two_level.py; then follow the tutorial to change two_level.py to add in the cache components.

  More specifically, you will follow the tutorial and perform the sections named "Creating cache objects" and "Adding caches to the simple config file". You will stop when you finished (quote from the tutorial): "Everything else in the file stays the same! Now we have a complete configuration with a two-level cache hierarchy. If you run the current file, hello should now finish in 58513000 ticks. The full script can be found in the gem5 source at ~/gem5/configs/learning_gem5/part1/two_level.py." You will stop right before and not perform the "Adding parameters to your script" section.

Lab 3: Adding a Cache Hierarchy to an X86 Architecture in gem5

EEL 5764 – Fall 2022

- Unlike the previous tutorial ("Creating a simple configuration script"), this tutorial is not a clear "step by step" tutorial. Thus, it is easy to "get lost" and not clear on where they want you to make changes or additions to the base simple.py. As stated above, the complete two_level.py script is provided at ~/gem5/configs/learning_gem5/part1/two_level.py.

  The provided two_level.py has basically all the commands as the one you would create if you followed the instructions in the tutorial, except:

  - It has more comments and also has extra code that you do not need for this tutorial (although the extra code may not cause any problems).
  - There are "old" deprecated functions used. Note that the code will still work; but you will get several warning messages about deprecated functions when you run it.

  I suggest that you create your own two_level.py file based on the tutorial; but use the provided two_level.py as a guide to where to add the new statements.

  Part A deliverable: <u>last screen of the output</u> showing the number of ticks for the hello-world program. (Page 1 of .pdf file of the lab report).

Part B – Create, compile, and run matmult.c in the new architecture.

B(1) Given in the attached file is a template that you will use for matmult.c. Complete the code following the specification given below.

- You have to use the given defined constants (NRA, NCA, NRB, and NCB) and the matrix names (A, B, C).
- Fill in the code based on the comments given in the template.
- Initial matrix A and matrix B with synthetically generated data as follows.
  - Use a nested loop for i and j; initialize i=0 and j=0.
  - For matrix A[i][j] = i + j; For matrix B[i][j] = i - j;
  - For example, assume the dimension for A is 5X4, and B is 6X5, then after initialization the contents of A and B are as follows:

```
matrix A   0 1 2 3 4        matrix B  0 -1 -2 -3 -4 -5
           1 2 3 4 5                  1  0 -1 -2 -3 -4
           2 3 4 5 6                  2  1  0 -1 -2 -3
           3 4 5 6 7                  3  2  1  0 -1 -2
                                      4  3  2  1  0 -1
```

Compile matmult.c, modify the two-level.py configuration script, and run the matmult binary in the new X86 architecture with cache hierarchy.

B(1) deliverable: <u>last screen</u> of the output showing the number of ticks for the matmult program. (Page 2 of pdf file).

B(2) Using simple.py (i.e., without the L1 and L2 caches), run the matmult program.
Deliverable: <u>last screen</u> of the output showing the number of ticks for the matmult program. **And one sentence** giving your observations. (Page 3 of pdf file).

B(3) In the cache.py script, vary **L1Cache data_latency** from 1, 2, 4, 8 (note that "2" is the default value from Part B(1) above).
B(3) deliverable:

- <u>One line</u> of the output showing the number of ticks <u>for each</u> of the 4 cases. (total of 4 lines, <u>not 4 screens</u> for Page 4 of pdf file).

- Briefly (a couple of sentences) giving a description of the results (e.g., comparison, your observations), also in Page 4 of pdf file.

Part C – Review "Understanding about gem5 statistics and output" tutorial again:

(https://www.gem5.org/documentation/learning_gem5/part1/gem5_stats/)

- Run the original two-level.py (with L1Cache data_latency = 2) with matmul, go to the m5out directory, and open the config.ini file.
- Browse through the config.ini file, find the information on following SimObjects for L1DCache, L1ICache, and L2Cache. Copy/paste the information into the pdf file started on Page 5.
  - [… replacement_policy] (for the L1Dcache, the full name of the SimObject is [system.cpu.dcache. replacement_policy]
  - [...tags]
  - [...tags.indexing_policy]

  Deliverable: I will explain further what I want in class. The purpose of this part of Lab 3 is to make you browse through the information of the config.ini file. You will gain a better appreciation for the meaning of some of these concepts (e.g., replacement policy, tag, assoc, etc.) after the lectures on cache and cache design.

<u>**SUBMISSION INSTRUCTIONS**</u>**:** Turn in **one pdf file** and **one zip file**.

<u>**The pdf file**</u> should have the following naming convention:

LastNameFirstNameLab3.pdf ex: LamHermLab3.pdf

The **pdf file** should contain (repeating the deliverables stated above):

Page 1: Part A deliverable, showing the hello world output (last screen).

Page 2: Part B(1) deliverable (using two-level.py), showing the matmult output (last screen).

Page 3: B(2) deliverable Output Part B2 (using simple.py without the caches) (last screen).

Page 4: B(3) deliverable, vary L1Cache data_latency from 1, 2, 4, 8 (4 lines of outputs (not 4 screens), each showing the exiting @ tick number), followed by a brief description.

Starting on **Page 5**, etc.**: Part C** deliverable, copied/pasted from config.ini file.

<u>**The zip file**</u> should have the following naming convention:

LastNameFirstNameLab3.zip ex: LamHermLab3.zip

There should be 4 files in the zip file:

1. caches.py
2. two_level.py
3. matmult.c
4. matmult binary