

CS-5513-002-Computer Architecture  
Fall-2022

**Cache Simulator that Simulates Set Associative  
Caches with LRU replacement policy**

**Group Member**  
Shobnom Roksana (nok057)

Date: November 30, 2022

## 1. Project Overview:

The project implements the cache simulator that simulates set-associative caches of any sizes and any associativities/ways. I have implemented the Least-Recently-Use (LRU) algorithm for cache replacement policy mainly a **true LRU** policy. For write policy, the program uses a **write-back** policy.

## 2. Brief Description of the Implementation of the Simulator:

The implementation basically contains **2 python files**:

**2.1. Configuration.py:** In this file the configuration parameters value for n-way set associative cache, size of the cache, block/cacheline size. The **configuration.py** file need to be updated and then restarted running the file for any changes and tests with various parameters. Below is a quick explanation of each variable in this Python code-  
**n:** represents the  $2^n$  way set-associative cache  
**x:** represents that cache size is  $2^x$   
**y:** represents that size of each block/cacheline is  $2^y$   
**LRU=True:** if the **LRU replacement policy is applied**; otherwise LRU=False

**Note:** From the above parameters the number of sets and the number of bits to represent the set index  $s$  has been determined by this -

$$s = x - n - y$$

Therefore, in this project, if the  $s=0$  then it will act as a **fully associative cache**. And if  $n=0$  then the cache simulator will act as a **direct-mapped cache**.

**2.2. Cache.py:** This file contains the input file processing, cache creation along with cache-set and blocks inside the cache-sets. For a 16-way set-associative cache there are 16 blocks in each of the sets and the total number of sets depends on the size of the cache (set in the configuration.py file).

**i) Processing of input files:** At first, I took the virtual addresses and W/R labels by reading the input files line by line. Then I eliminated any lines that didn't have 3 space-separated items on them. Once the virtual address has been processed, I used the **get\_block\_offset()**, **get\_set\_index()**, and **get\_tag()** functions to get the offsets, set index, and tag for that entry accordingly.

**ii) Enqueue/ Dequeue operations:** Queue module of the Python **collections.dque** has been used to implement the LRU policy. The next tag block was added to the right of the queue using the **dqueue.append()** function, and the oldest block in the queue was replaced with the new block that had just arrived using the **dqueue.popleft()** technique. Again after, I had to update the queue each time I got the cache hit with the block's new position as the rightmost position.

For the write policy, the **write-back** policy has been implemented. However, the read and write programs function equally in this program.

### 3. Experiments and Results:

In this section, a couple of sample test cases are presented with my own sample test cases and different configuration settings.

#### 3.1. Sample Test Case 1:

##### Sample Input Trace File (mytest1.txt):

0x0000000: R 0x0  
0x0000000: R 0x40  
0x0000000: R 0x80  
0x0000000: R 0xc0  
0x0000000: R 0x100  
0x0000000: R 0x140  
0x0000000: R 0x0  
0x0000000: R 0xc0  
0x0000000: R 0x40  
0x0000000: R 0x100  
0x0000000: R 0x140  
0x0000000: R 0x80

##### Configuration Setting:

N-way = **4-way** set associative cache

Cache Size= 256B

Block size = 32B

So, for this test in "**Configuration.py**" file the updated values are,

n = 2, x = 8, y = 5

Output: 91.67%

#### 3.2. Sample Test Case 2:

##### Sample Input Trace File (mytest1.txt):

0x0000000: R 0x0  
0x0000000: R 0x40  
0x0000000: R 0x80  
0x0000000: R 0xc0  
0x0000000: R 0x100  
0x0000000: R 0x140  
0x0000000: R 0x0  
0x0000000: R 0xc0  
0x0000000: R 0x40

0x0000000: R 0x100  
0x0000000: R 0x140  
0x0000000: R 0x80

**Configuration Setting:**

N-way = **8-way** set associative cache  
Cache Size= 1KB  
Block size = 32B

So, for this test in “**Configuration.py**” file the updated values are,  
n = 3, x = 10, y = 5

Output: 50.00%

**3.3. Sample Test Case 3:**

**Sample Input Trace File (mytest2.txt):**

0x0000000: R 0x12C  
0x0000000: R 0x130  
0x0000000: R 0x4C0  
0x0000000: R 0x1134  
0x0000000: R 0x1138  
0x0000000: R 0x24C8  
0x0000000: R 0x128  
0x0000000: R 0x130  
0x0000000: R 0x4C4  
0x0000000: R 0x8C8

**Configuration Setting:**

N-way = **2-way** set associative cache  
Cache Size= 2KB  
Block size = 16B

So, for this test in “**Configuration.py**” file the updated values are,  
n = 1, x = 11, y = 4

Output: 60.00%

### 3.4. Sample Test Case 4:

#### Sample Input Trace File (mytest3.txt):

```
0x00000000: W 0x12C
0x00000000: R 0x130
0x00000000: R 0x1134
0x00000000: W 0x1138
0x00000000: W 0x2130
0x00000000: R 0x2134
0x00000000: R 0x130
```

#### Configuration Setting:

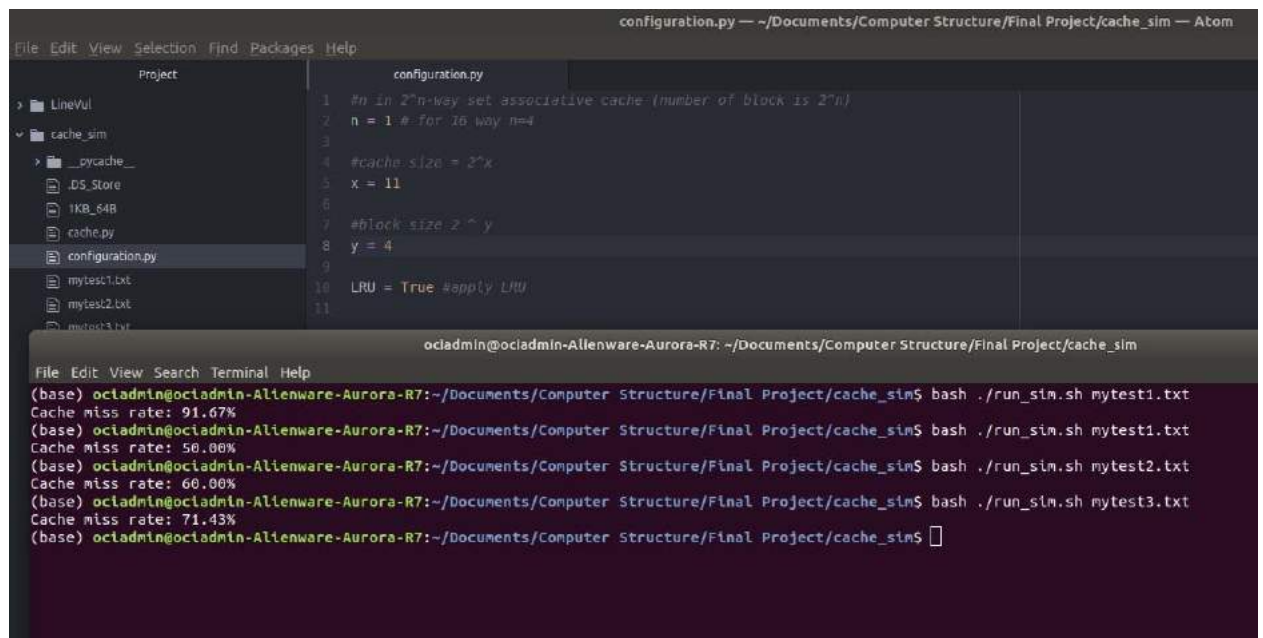
N-way = **2-way** set associative cache  
Cache Size= 2KB  
Block size = 16B

So, for this test in “**Configuration.py**” file the updated values are,

n = 1, x = 11, y = 4

Output: 71.43%

As test case 1 & 2 I have used same input trace file and different configuration so there is in total 3 test files “mytest1.txt”, “mytest2.txt” & “mytest3.txt”. Here is the screenshot of my results –



The screenshot shows an IDE window titled "configuration.py — ~/Documents/Computer Structure/Final Project/cache\_sim — Atom". The left sidebar shows a project tree with folders "LineVul" and "cache\_sim", and files ".DS\_Store", "1KB\_64B", "cache.py", "configuration.py", "mytest1.txt", "mytest2.txt", and "mytest3.txt". The "configuration.py" file is open in the editor, showing the following code:

```
1 #n in 2^n-way set associative cache (number of block is 2^n)
2 n = 1 # for 16 way n=4
3
4 #cache size = 2^x
5 x = 11
6
7 #block size 2^y
8 y = 4
9
10 LRU = True #apply LRU
11
```

Below the editor is a terminal window titled "ocladmin@ocladmin-Allenware-Aurora-R7: ~/Documents/Computer Structure/Final Project/cache\_sim". It shows the following commands and output:

```
(base) ocladmin@ocladmin-Allenware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh mytest1.txt
Cache miss rate: 91.67%
(base) ocladmin@ocladmin-Allenware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh mytest1.txt
Cache miss rate: 50.00%
(base) ocladmin@ocladmin-Allenware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh mytest2.txt
Cache miss rate: 60.00%
(base) ocladmin@ocladmin-Allenware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh mytest3.txt
Cache miss rate: 71.43%
(base) ocladmin@ocladmin-Allenware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$
```

**Table 1: Summary of results on the given benchmark memory traces for “16-way set associative cache”**

<b>Input Trace File</b>	<b>Configuration</b>	<b>Cache Miss Rate</b>
<b>1KB_64B</b>	Cache size= 1KB Block size= 64B	50%
<b>4MB_4B</b>	Cache size= 4MB Block size= 64B	2.08%
<b>32MB_4B</b>	Cache size= 4MB Block size= 64B	6.25%
<b>bw_mem</b>	Cache size= 4MB Block size= 64B	1.56%
<b>ls</b>	Cache size= 32KB Block size= 64B	2.17%
<b>gcc</b>	Cache size= 32KB Block size= 64B	1.89%
<b>native_dgemm</b>	Cache size= 256KB Block size= 64B	50.25%
<b>native_dgemm_full</b>	Cache size= 256KB Block size= 64B	49.37%
<b>openblas_dgemm</b>	Cache size= 256KB Block size= 64B	8.30%
<b>openblas_dgemm_full</b>	Cache size= 256KB Block size= 64B	7.5%

In “Table 1” the results of all of the miss rate matches with the given project miss rate references with the same configuration and using 64 byte block size. Here is the screenshot –

```

configuration.py
1 #n in 2^n-way set associative cache (number of block is 2^n)
2 n = 4 # for 16 way p=4
3
4 #cache size = 2^x
5 x = 18
6
7 #block size 2^y
8 y = 6
9
10 LRU = True #apply LRU
11

ociadmin@ociadmin-Alienware-Aurora-R7: ~/Documents/Computer Structure/Final Project/cache_sim
File Edit View Search Terminal Help
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh 1KB_64B
Cache miss rate: 50.00%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh 4MB_4
Cache miss rate: 2.08%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh 32MB_48
Cache miss rate: 6.25%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh bw_mem.traces.txt
Cache miss rate: 1.56%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh ls.trace.txt
Cache miss rate: 2.17%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh gcc.trace.txt
Cache miss rate: 1.89%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh naive_dgemm.trace.txt
Cache miss rate: 50.25%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh naive_dgemm_full.trace.txt
Cache miss rate: 49.37%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemm.trace.txt
Cache miss rate: 8.30%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh full_dgemm.traces.txt
Cache miss rate: 7.50%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ 

```

Next in Table 2, the results on the “openblas\_dgemm” trace file with different configurations like changing the block size and cache sizes to see if that impacts the cache miss rate.

**Table 2: Summary of results on the openblas\_dgemm trace file for different configuration**

Cache Size	Block Size	Miss Rate
32KB = $2^{15}$	64B	22.13%
1MB = $2^{20}$	64B	8.06%
4MB = $2^{22}$	64B	2.27%
32KB = $2^{15}$	128B	12.27%
1MB = $2^{20}$	128B	4.04%
4MB = $2^{22}$	128B	1.14%

According to table 2, when the cache size increases from 256KB to 4MB, the miss-rate considerably reduces. Additionally, although the change is not very significant, the miss rate decreases as block size increases. We can see that the miss rate is 8.06% for 1MB cache and 64B block size but drops to 4.04%

for 128B block size with the same 1MB cache. Here is the screenshot –

```
configuration.py
1: #n in 2^n-way set associative cache (number of block is 2^n)
2: n = 4 # for 16 way n=4
3:
4: #cache size = 2^x
5: x = 22
6:
7: #block size 2^y
8: y = 7
9:
10: LRU = True #apply LRU
11:

ociadmin@ociadmin-Alienware-Aurora-R7: ~/Documents/Computer Structure/Final Project/cache_sim
File Edit View Search Terminal Help
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 22.13%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 8.06%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 2.27%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 12.27%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 4.04%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$ bash ./run_sim.sh openblas_dgemv.trace.txt
Cache miss rate: 1.14%
(base) ociadmin@ociadmin-Alienware-Aurora-R7:~/Documents/Computer Structure/Final Project/cache_sim$
```

#### 4. Contribution Summary:

On the project, I worked by myself. I ran into some issues when running the code for Python 3 first, as well as when implementing the queue for LRU. However, I was able to work in my lab office afterwards. Even though I worked alone on this project, I enjoyed it and tried my best to complete.