# COL216 Assignment 3

Dhruv Gupta 2021CS50125
Niranjan Sarode 2021CS50612

## 1 Introduction

### 1.1 Problem Statement

- In this Assignment we were required to implement a cache simulator with 2 level caches L1 and L2 in C++ which reads a trace file and counts the hits,misses,reads,writes etc for both the caches and prints the stats.
- `w 400341a0` This is a write (read if start in r) operation at address 400341a0 (hexadecimal address).
- Cache eviction policy: Simulator uses least replacement policy (LRU).
- Cache write policy: Simulator uses Write-back Write-allocate (WBWA).

### 1.2 Design Decisions

- We have used a struct to represent a cache line. It has the following fields:
  - Valid
  - Dirty
  - Tag
  - Value
- for LRU we used a vector of size equal to the associativity of the cache. Each element of the vector represents a cache line. The element at index i represents the ith most recently used cache line.
- For a read Instruction (identified by r)
  - We find index bits tag bits and offset bits for both L1 and L2 from the address
  - Then we check L1 cache from the L1 index bits
  - If its tag matches with cache block tag, its valid is true then its a read-L1-hit.
  - Otherwise (if its invalid or tag doesn't match) then its a read-L1-miss then it searches in L2
  - Writeback the least used element in LRU to L2 if its dirty bit is true
  - Then we check in L2 cache from the L2 index bits
  - If its tag matches with cache block tag, its valid is true then its a read-L2-hit then replaces the least recently used block in L1.
  - Otherwise (if its invalid or tag doesn't match) then its a read-L2-miss then it searches in memory and replaces the least recently used block in L2 and it replaces the least recently used block in L1.
- For a write Instruction (identified by w)
  - We find index bits tag bits and offset bits for both L1 and L2 from the address
  - Then we check L1 cache from the L1 index bits
  - If its tag matches with cache block tag, its valid is true then its a write-L1-hit then we Modify the dirty attribute to true for that block.
  - Otherwise (if its invalid or tag doesn't match) then its a write-L1-miss.

  - Writeback the least used element in LRU to L2 if its dirty bit is true
  - Then we check in L2 cache from the L2 index bits
  - If its tag matches with cache block tag, its valid is true then its a read-L2-hit.Modify the dirty attribute to true for that block and replace the least recently used block in L1.
  - Otherwise (if its invalid or tag doesn't match) then its a read-L2-miss then it searches in memory and replaces the least recently used block in L2 replaces the least recently used block in L1.

- For writeback to L2 from L1:
  - We find index bits tag bits and offset bits for L2.
  - Check if block is found in L1 if yes then write in L2 and make dirty bit true.
  - Otherwise (if its invalid or tag doesn't match) then its a write-L2-miss and value is read from memory and written in L2.

# 2 Results and Analysis

## 2.1 Graphs

### 2.1.1 Variation in Block Size

Graph obtained by varying block size on x-axis between 8, 16, 32, 64, 128, and keeping L1 size, L1 associativity, L2 size and L2 associativity fixed at 1024, 2, 65536, 8 :-

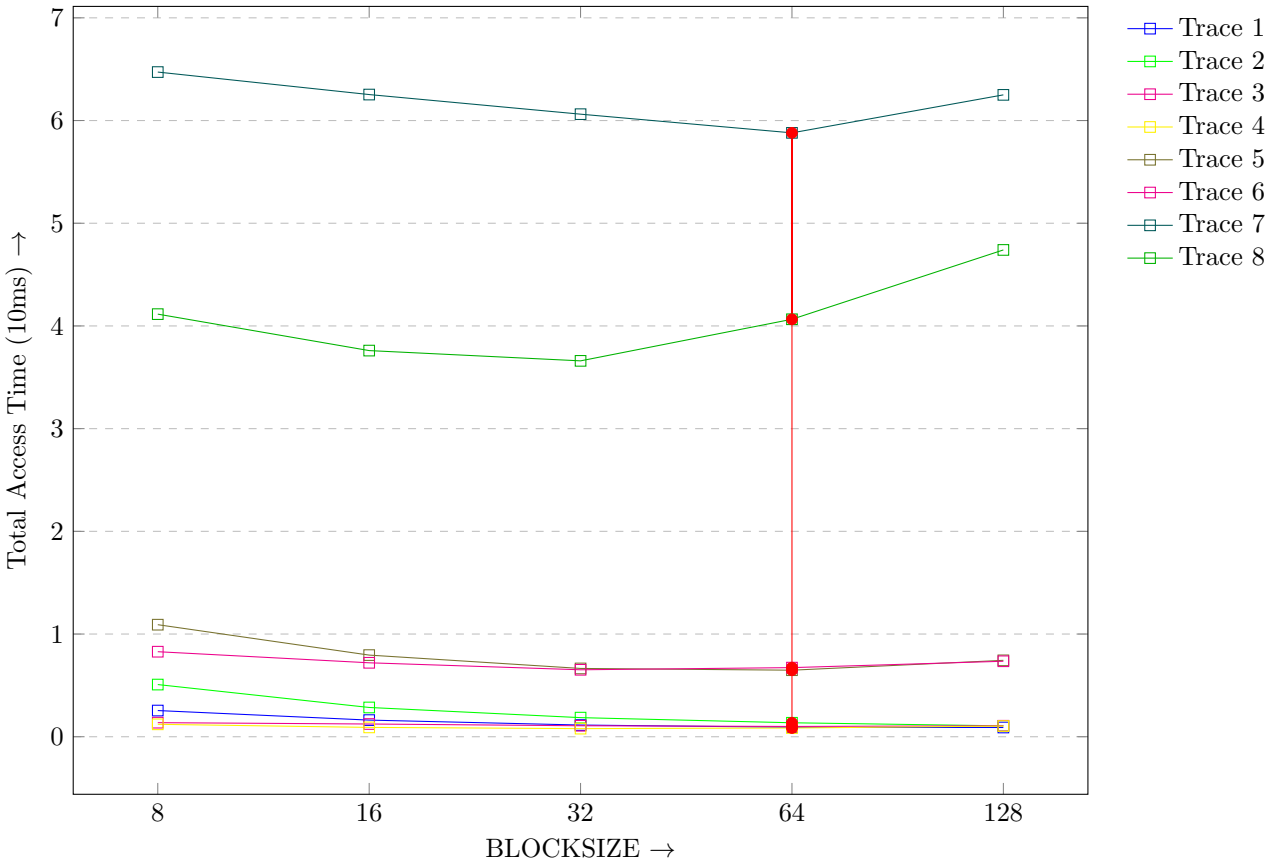


Figure 1: Variation in blocksize

- **Theoretical Expectation** : As the block size increases, the number of blocks that can fit in a cache line increases, which reduces the number of cache misses. Therefore, we expect to see a decrease in the miss rate and total access time as the block size increases. However, as the block size becomes larger, the cache line may become less efficient due to unused data. Hence, there is an optimal block size that minimizes the miss rate and access time.
- We see that total access time decreases then increases which matches with the theoretical expectation. The variation is significant for trace 7 and trace 8. The default value of 64 is high enough to give a decent performance for some of the traces.

## 2.2 Variation in L1 size

In the second graph, vary the second parameter L1 size between 512, 1024, 2048, 4096, 8192, and keep block size, L1 associativity, L2 size and L2 associativity fixed at 64, 2, 65536, 8.

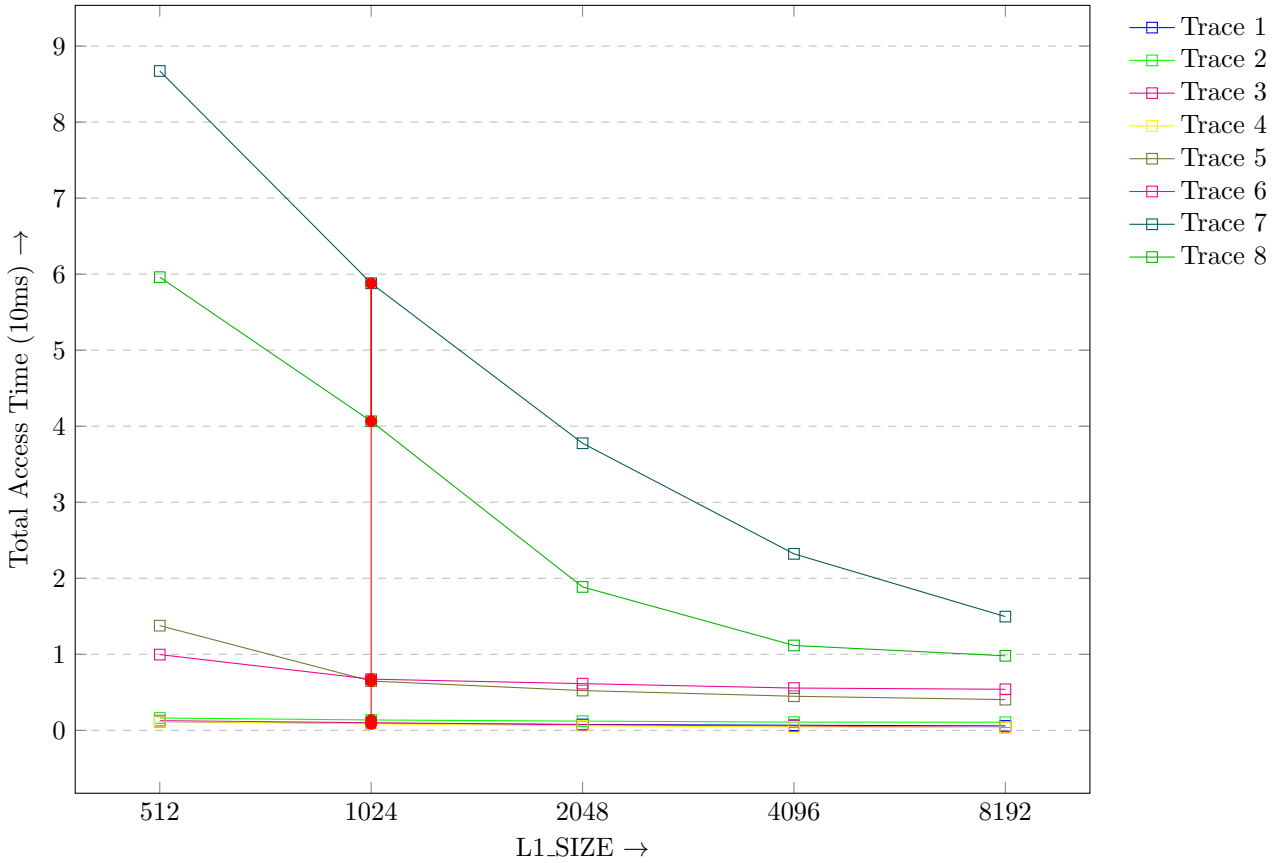


Figure 2: Variation in L1 size

- **Theoretical Expectation** : As the L1 cache size increases, the cache can store more blocks, which reduces the number of cache misses. Therefore, we expect to see a decrease in the miss rate and total access time as the L1 cache size increases. However, increasing the L1 cache size also increases the access latency, which may offset the reduction in the miss rate. Hence, there is an optimal L1 cache size that minimizes the total access time.
- We can see that for all the traces, the total access time decreases as the L1 size increases which matches with theoretical expectation. The variation is significant for trace 5, trace 7 and trace 8.

### 2.2.1 Variation in L1 Associativity

In the third graph, vary the third parameter L1 associativity between 1, 2, 4, 8, 16, and keep block size, L1 size, L2 size and L2 associativity fixed at 64, 1024, 65536, 8.

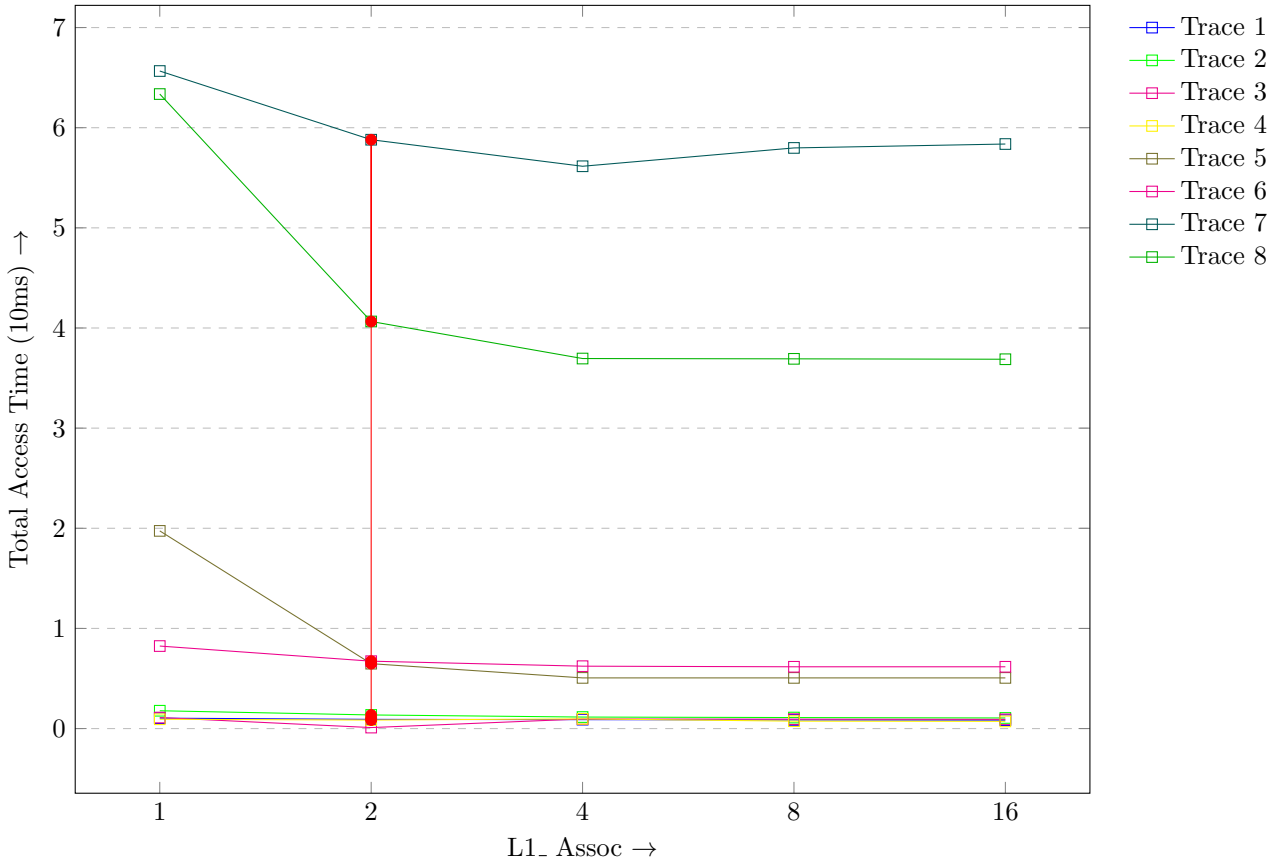


Figure 3: Variation in L1 Associativity

- **Theoretical Expectation** : As the L1 cache associativity increases, the cache can store more blocks and hence reduce the number of cache misses. Therefore, we expect to see a decrease in the miss rate and total access time as the L1 associativity increases . However, increasing the associativity also increases the access latency, which may offset the reduction in the miss rate. Hence, there is an optimal L1 associativity that minimizes the total access time.
- From the Graph we can conclude that the access time decreases as the associativity increases then increases at higher values for some traces which matches with Theoretical Expectation. The effect is significant for trace 7 and trace 8. The default value of associativity is 2 which is the optimal value for some of the traces.

### 2.2.2 Variation in L2 Size

In the fourth graph, vary the fourth parameter L2 size between 16384, 32768, 65536, 131072, 262144, and keep block size, L1 size, L1 associativity and L2 associativity fixed at 64, 1024, 2, 8.

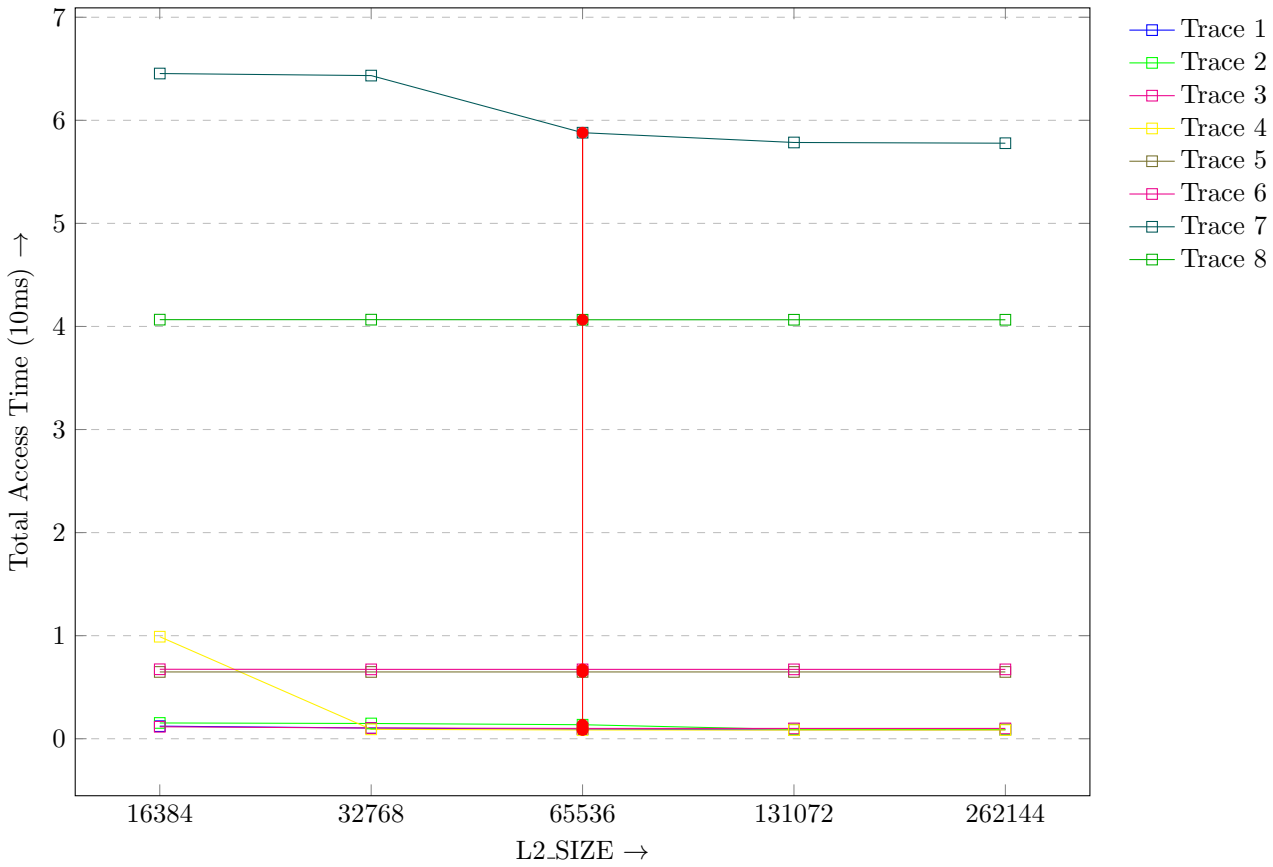


Figure 4: Variation in L2 Size

- **Theoretical Expectation** : As the L2 cache size increases, the cache can store more blocks and hence reduce the number of cache misses. Therefore,we expect to see a decrease in the miss rate and total access time as the L2 cache size increases. However, increasing the L2 cache size also increases the access latency, which may offset the reduction in the miss rate. Hence, there is an optimal L2 cache size that minimizes the total access time.
- From the Graph we can conclude that the access time decreases negligibly for most traces as the L2 size increases and significantly for trace 7 which matches theoretical expectation. This means that the L2 size is not a major factor in determining the access time and affects significatly only if L1 miss rate is high for a trace. The default value of L2 size is 65536.

### 2.2.3 Variation in L2 Associativity

In the fifth graph, vary the fifth parameter L2 associativity between 1, 2, 4, 8, 16, and keep block size, L1 size, L1 associativity and L2 size fixed at 64, 1024, 2, 65536.

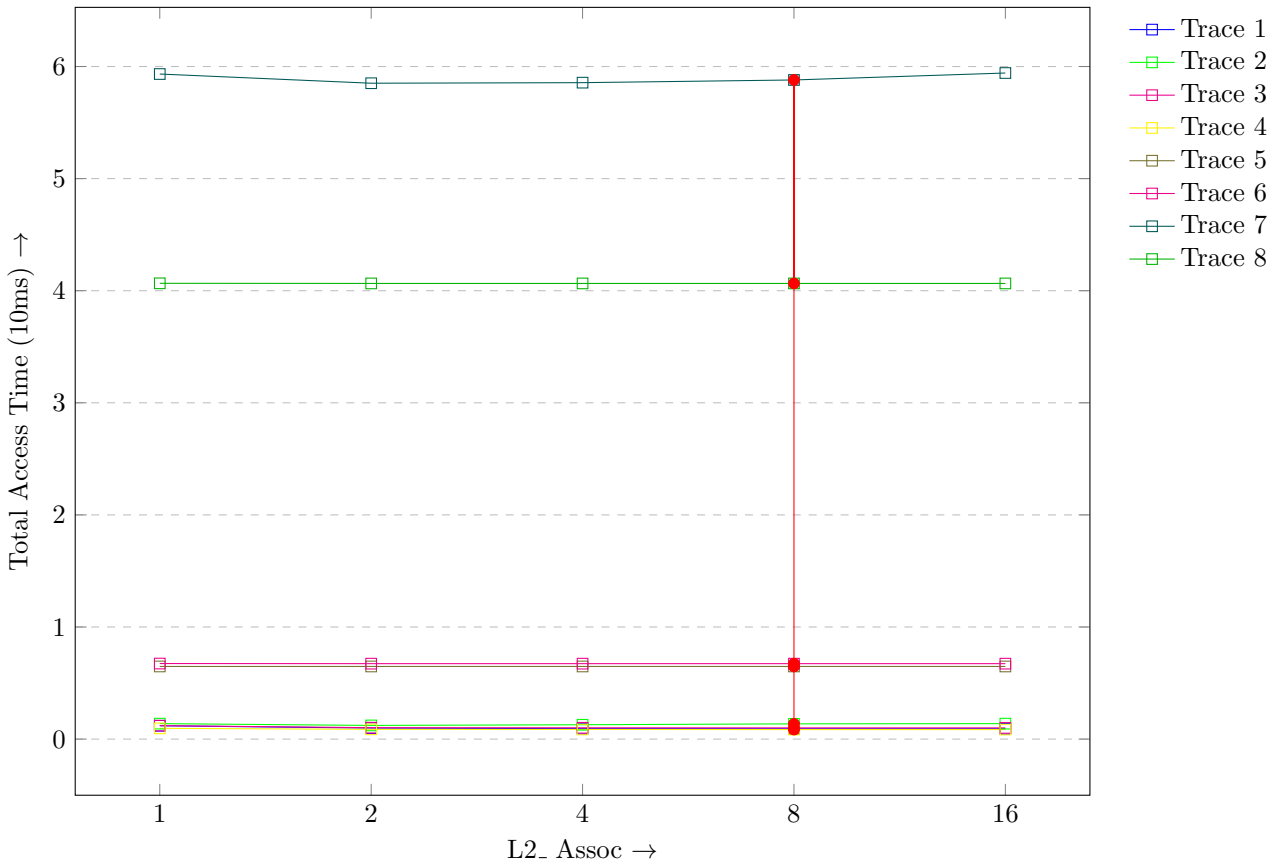


Figure 5: Variation in L2 Associativity

- **Theoretical Expectation** : As the L2 cache associativity increases, the cache can store more blocks and hence reduce the number of cache misses. Therefore, we expect to see a decrease in the miss rate and total access time as the L2 associativity increases. However, increasing the associativity also increases the access latency, which may offset the reduction in the miss rate. Hence, there is an optimal L2 associativity that minimizes the total access time.
- From the Graph we can see that the access time remains constant for most traces except trace 7 for which it decreases slightly then increases slightly which matches theoretical expectation.This means that the L2 associativity is not a major factor in determining the access time and affects significatly only if L1 miss rate is high for a trace. The default value of L2 associativity is 8.

# 3 Instructions to run the code

run the following command in the terminal

```
$ make run
```

You can adjust the parameters in the Makefile to run the code for different traces and different cache parameters.

```
@./cache_simulate 64 1024 2 65536 8 memory_trace_files/trace1.txt
```

Here the parameters are in the order of block size, L1 size, L1 associativity, L2 size, L2 associativity and the trace file name.

# 4 Work Split

| S.No. | Team member | %Work Split |
|---|---|---|
| 1 | Dhruv Gupta | 50% |
| 2 | Niranjan Sarode | 50% |

Table 1: Token Distribution