COL216 Assignment-3 Report

Aanya Khurana 2021CS10084
Amaiya Singhal 2021CS50598

General Implementation

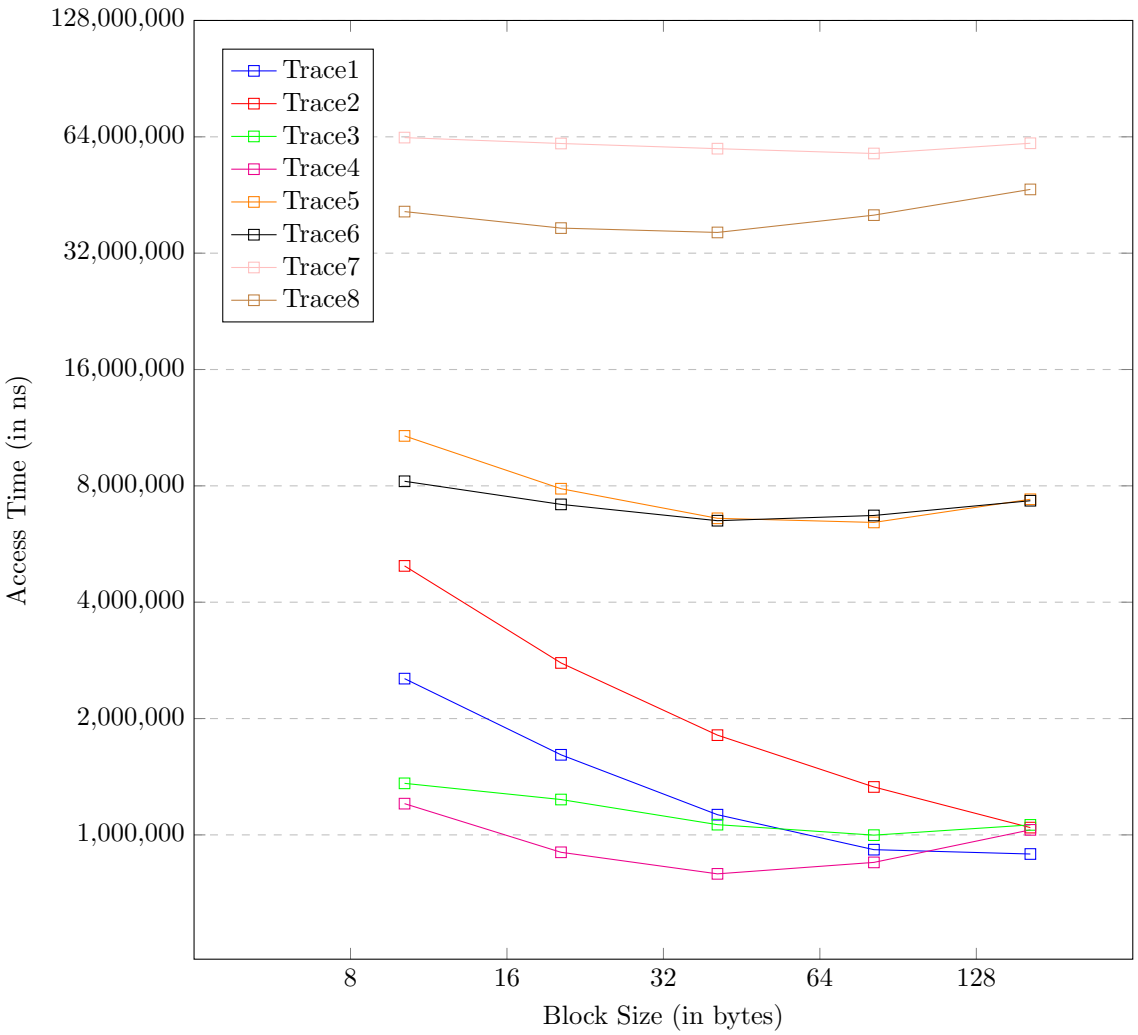
- We have implemented the two caches (**L1** and **L2**) as structs.
- The data structure used is a vector of vectors containing data of type `long long` (for the tag) and **bool** (for the dirty bit). `long long` was used because some of the addresses were of size 32 bit as well.
- Each struct has its own functions for `write, read, write_back` etc. Helper functions have also been made to manage the LRU and eviction of blocks from the 2 caches.
- The main cache.cpp file takes inputs from the trace files and runs the instructions using the functions of the caches.
- All types of scenarios have been taken care of through various helper functions and if-else conditions.
- The Write-back Write-allocate policy has been implemented. We ensure this by maintaining the dirty bits for every block that is present in the cache.
- Whenever a block is to be evicted from either of the caches then the dirty bit is checked, if it is equal to 1 then a write back request is sent to the next level.
- The LRU has been stored as a rank which specifies which of the ways has been least recently used and this is then used during the time of eviction.
- The following pages contain the observations and plots for the 5 different sets of parameters mentioned in the assignment PDF.
- For each of the parts, the first graph is a logarithmic (on the y-axis) plot and the second is a linear plot. The logarithmic plot has been made for better visualisation because the values between some of the traces differ significantly because of which the plots for some traces are not very visible.
- The code is in 2 parts, the `cache_structs.hpp` file contains the 2 caches and its helper functions that have been used in the `cache.cpp` file to execute all the instructions and calculate the results. The makefile can be used to run the final code with user specified parameters.

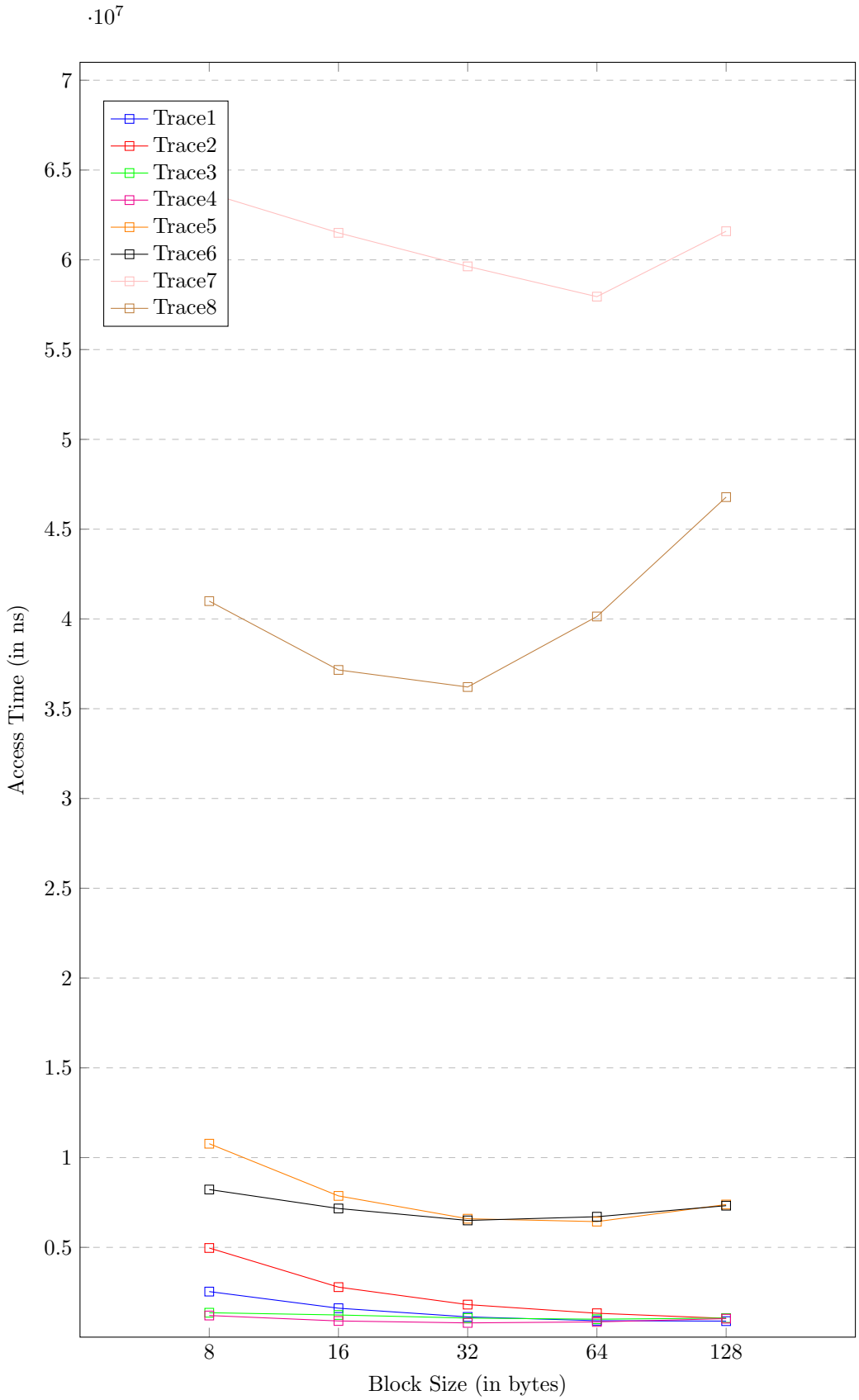
PLOT A : Variation of Access Time with respect to Block Size

Observations from the Plots

- Keeping all other parameters constant (at L1 size = 1024 bytes, L1 Associativity = 2, L2 Size = 65536 bytes, L2 Associativity = 8), we vary the block size in this case.
- Across plots, we observe that the access time reduces with increase in block size up till some limit, then it begins increasing again or becomes nearly constant.
- This is because the size of the caches are kept constant, so an increase in the size of the blocks implies a decrease in the number of sets. Even though one set can accommodate more bytes of data, as the block size becomes enormous, more and more addresses are mapped to the same set, and the conflict misses increase.
- The access times for Trace7 and Trace8 are significantly higher than the rest of the input files. For Trace7, the access time increases significantly, implying that the memory accesses involve addresses that frequently map to different sets in the cache.

Plotting from data:



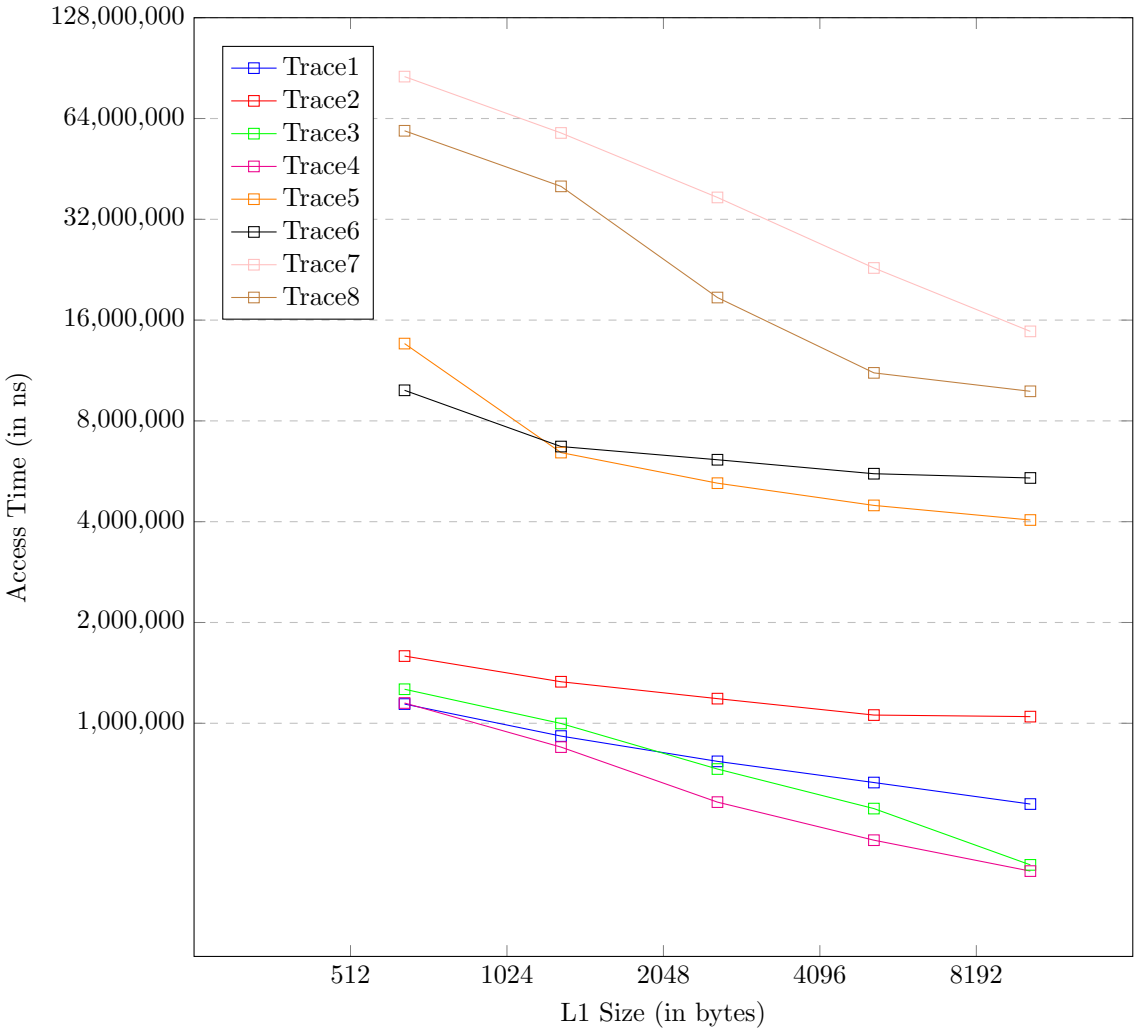

$\cdot 10^7$
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
Access Time (in ns)
Trace1
Trace2
Trace3
Trace4
Trace5
Trace6
Trace7
Trace8
8
16
32
64
128
Block Size (in bytes)

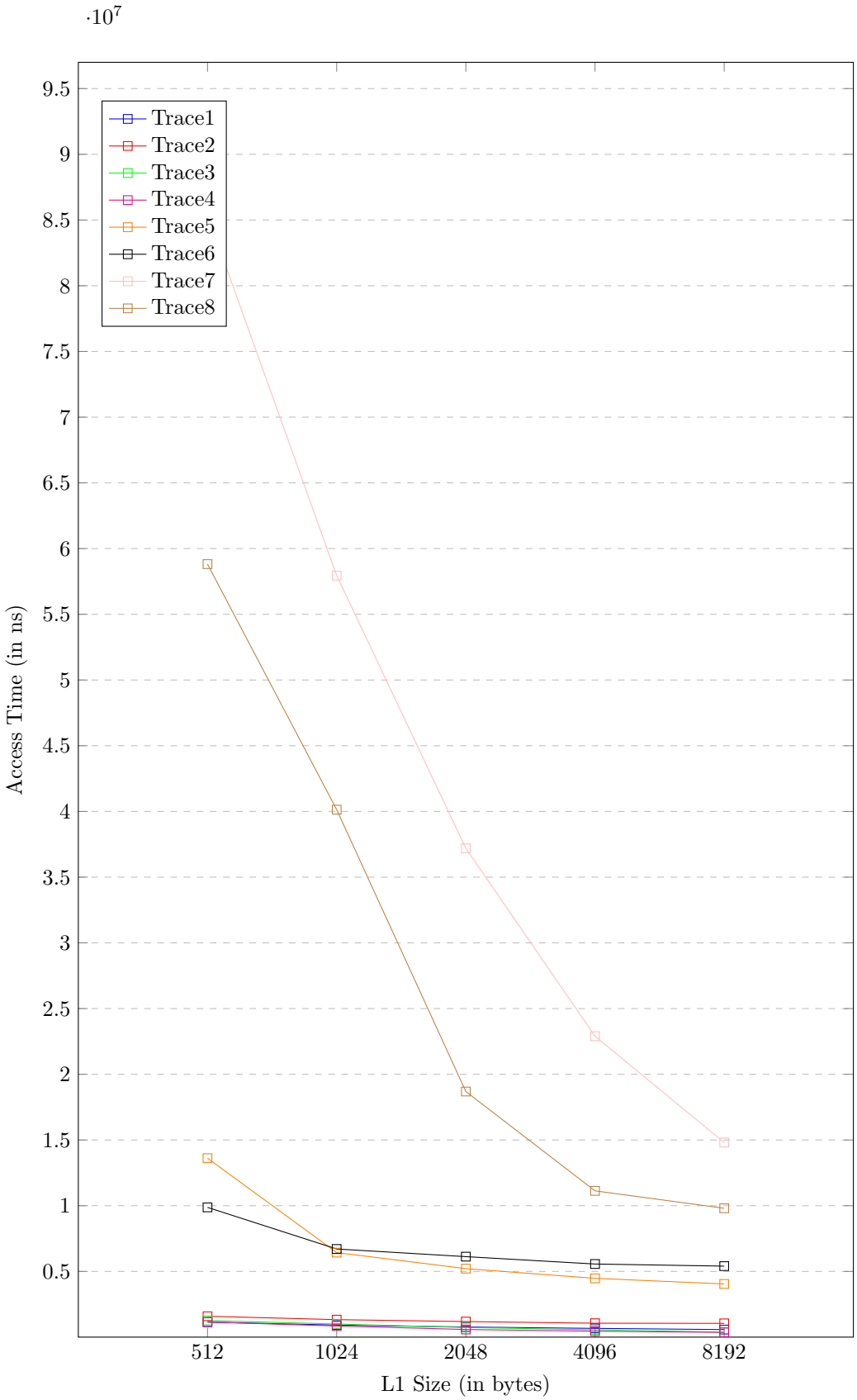
PLOT B : Variation of Access Time with respect to L1 Size

Observations from the Plots

- Keeping all other parameters constant (at Block Size = 64 bytes, L1 Associativity = 2, L2 Size = 65536 bytes, L2 Associativity = 8), we vary L1 size.
- All the plots show that access time reduces as the size of the L1 cache increases. This is expected, as we can infer that the number of sets increases with the L1 cache size (due to the fixed block size and fixed L1 associativity).
- The more the number of sets, the lesser are the capacity misses, so the latency is lesser.
- The reduction in latency is most significant for Trace7 and Trace8, indicating that in both of these traces we are accessing addresses that upon increasing the L1 size reduce the misses happening significantly, hence reducing the total access time.

Plotting from the data:



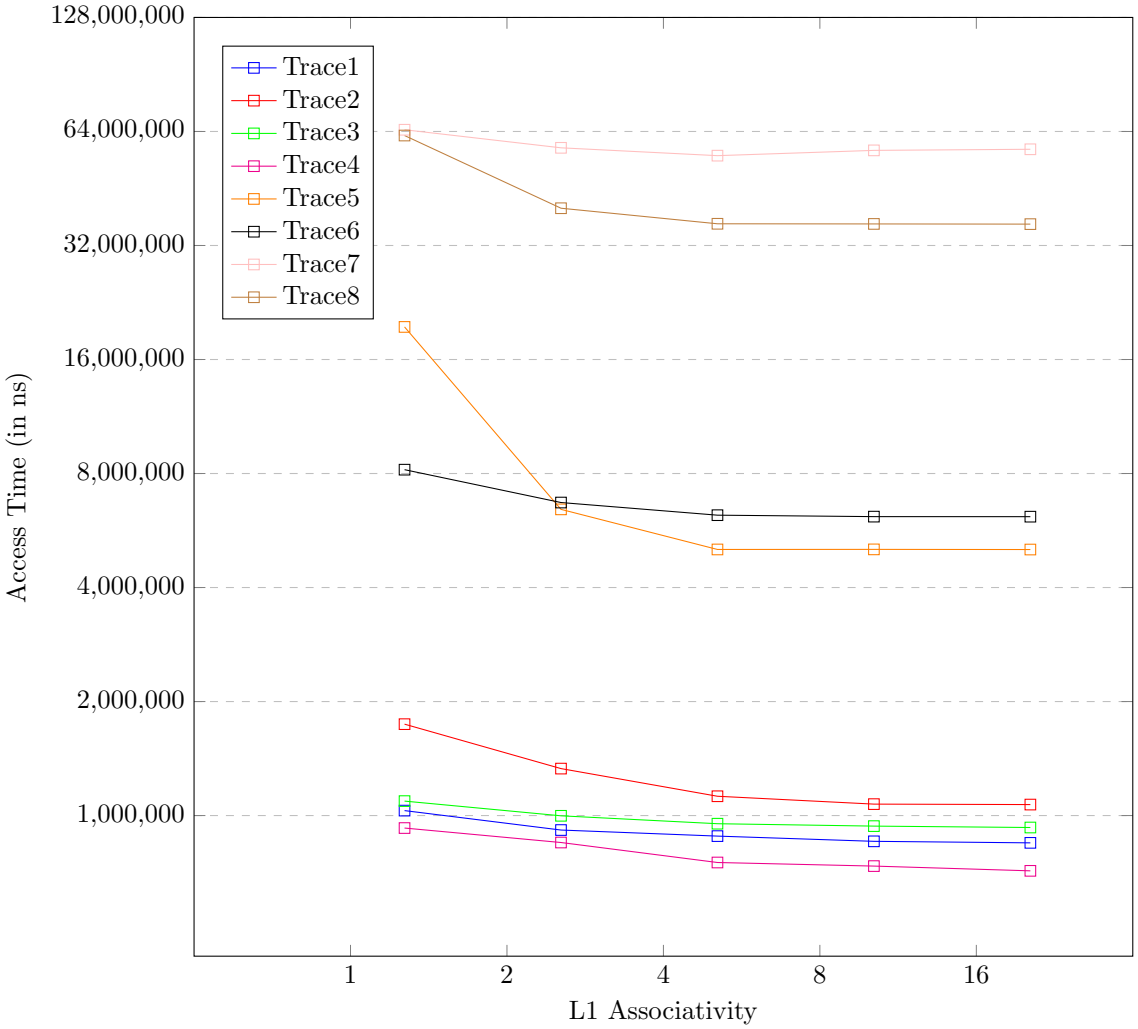

$\cdot 10^7$
9.5
9
8.5
8
7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
Access Time (in ns)
Trace1
Trace2
Trace3
Trace4
Trace5
Trace6
Trace7
Trace8
512
1024
2048
4096
8192
L1 Size (in bytes)

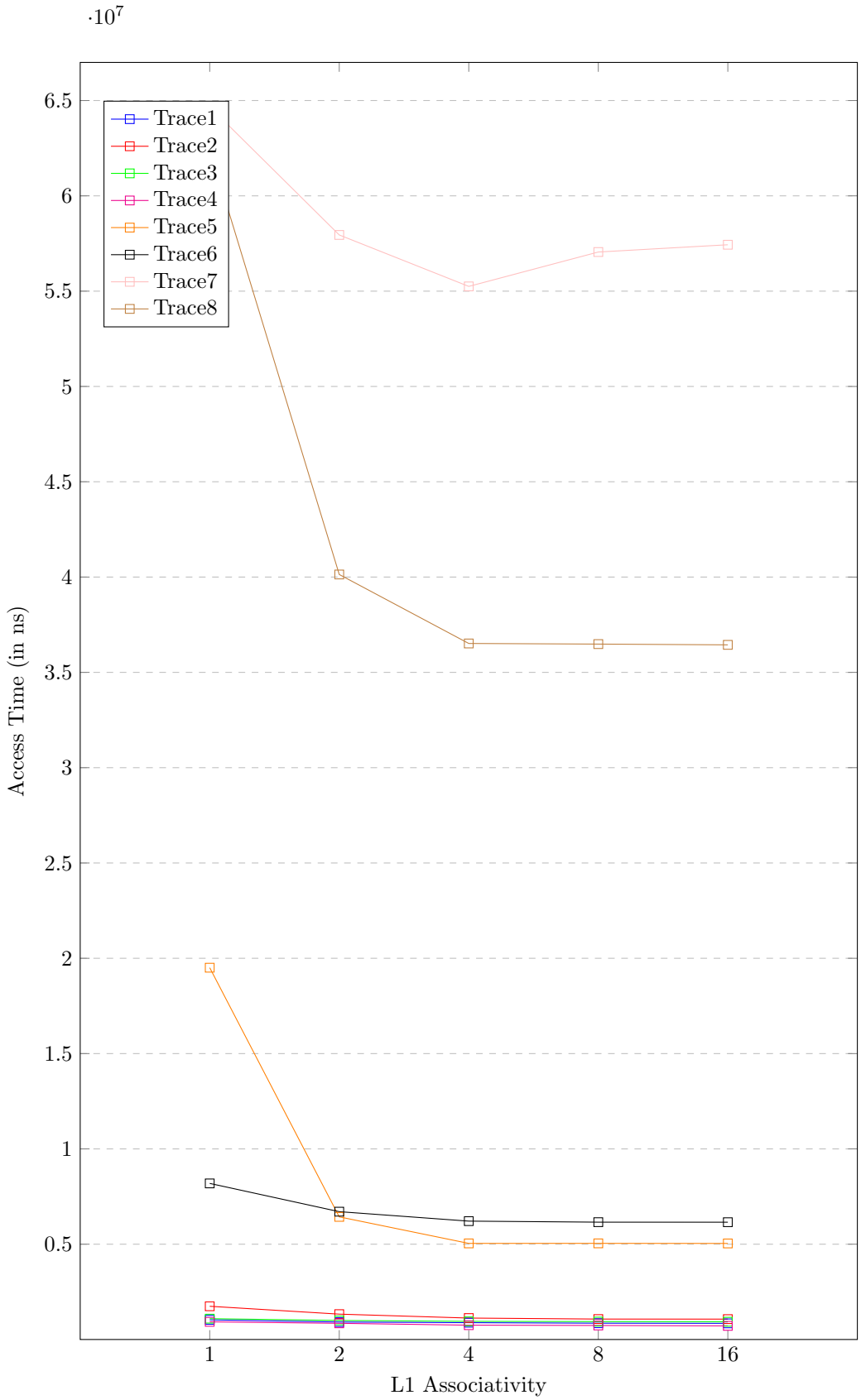
PLOT C : Variation of Access Time with respect to L1 Associativity

Observations from the Plots

- Keeping all other parameters constant (at Block Size = 64 bytes, L1 Size = 1024 bytes, L2 Size = 65536 bytes, L2 Associativity = 8), we vary L1 associativity.
- Initially, the access time reduces as the associativity increases, but as the associativity nears 8 or 16, the latency becomes constant or starts increasing.
- Given a fixed cache size and block size for the L1 cache, an increase in the associativity means a reduction in the number of sets. Initially, due to higher associativity, conflict misses are reduced as data stored in addresses mapping to the same set can occupy different ways. However, as the associativity becomes large, the effect of a less number of sets becomes prominent, and the increasing capacity misses outweigh the decreasing conflict misses.

Plotting from data:



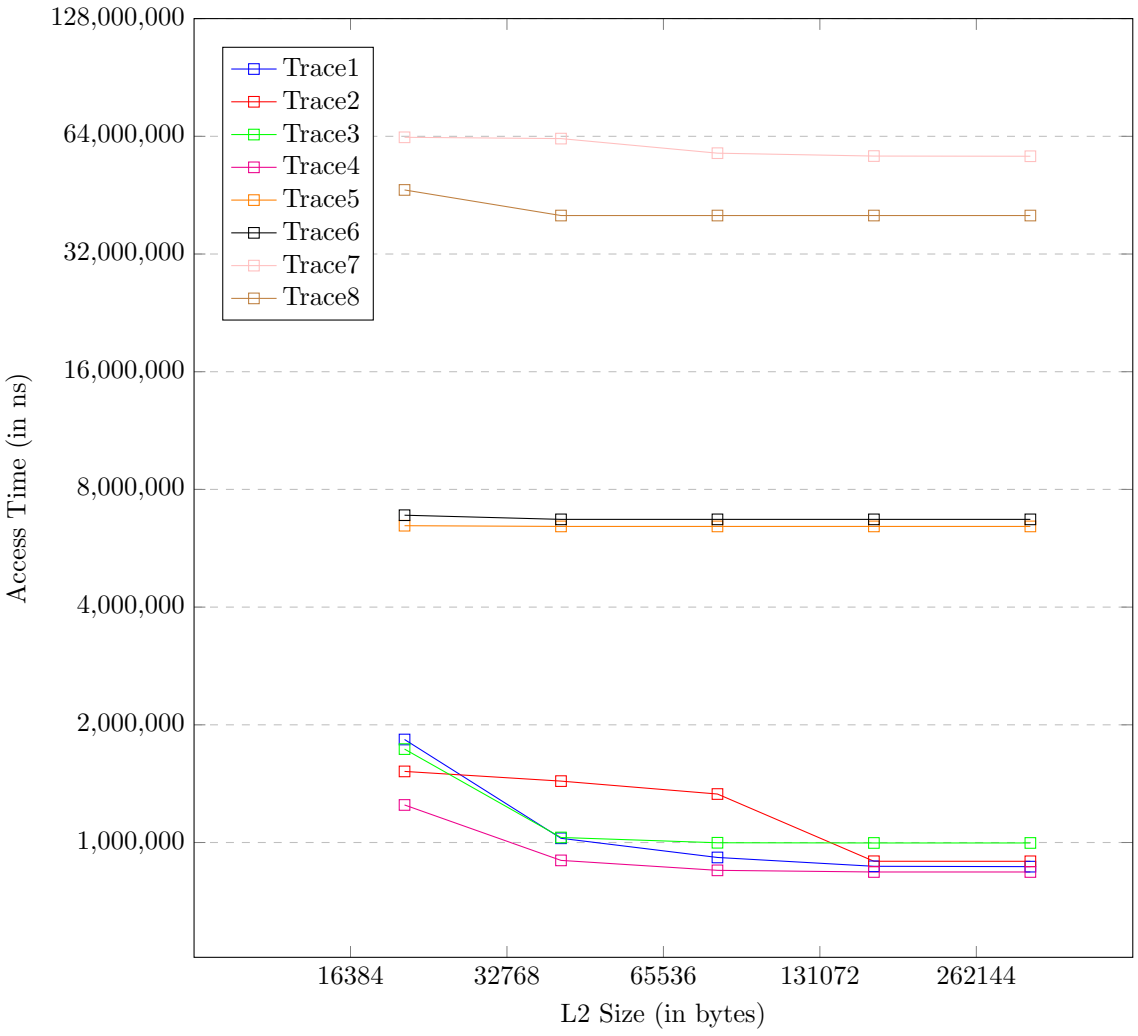

$\cdot 10^7$
Trace1
Trace2
Trace3
Trace4
Trace5
Trace6
Trace7
Trace8
Access Time (in ns)
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
1
2
4
8
16
L1 Associativity

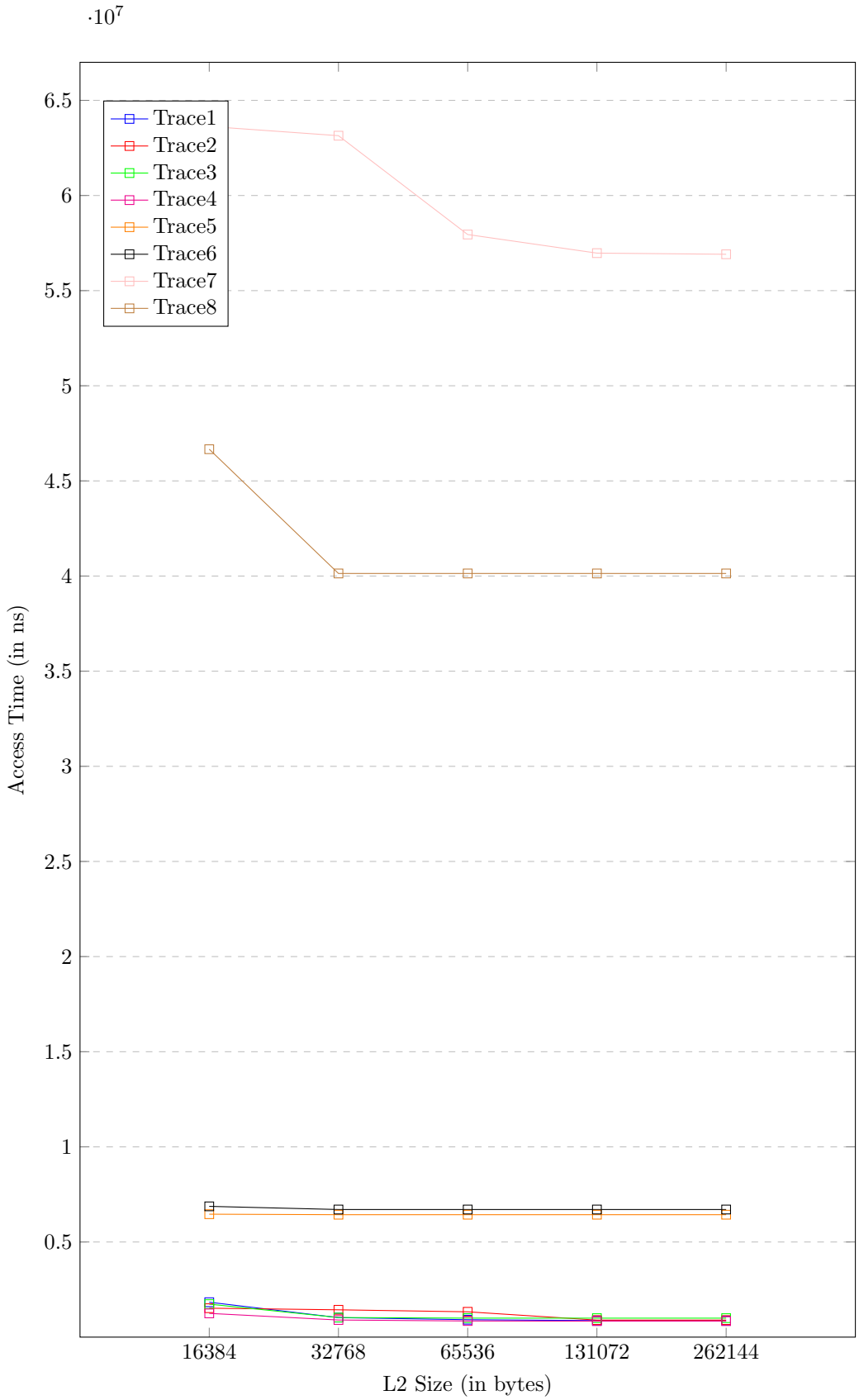
PLOT D : Variation of Access Time with respect to L2 Size

Observations from the Plots

- Keeping all other parameters constant (at block size = 64 bytes, L1 Size = 1024 bytes, L1 Associativity = 2, L2 Associativity = 8), we vary L2 size.
- The latency decreases with an increase in the size of L2 initially and as the cache size of L2 becomes huge (greater than 128 KB), the latency becomes almost constant.
- As the size of the L2 cache increases initially, the number of capacity misses of L2 decreases, and the L2 hits increase, so there is a reduction of accesses from the main memory. Hence, the access time reduces.
- However, as L2 becomes extremely large in size, as the number of L2 accesses is limited by the L1 cache, the effect of increasing size becomes constant and does not create much of a marginal difference. The reduction in memory accesses has mostly been achieved till then.
- We observe that the effect of increasing block size is way more significant in the case of the L1 cache than the L2 cache: this is because every memory access involves the L1 cache but not the L2 cache.

Plotting from data:



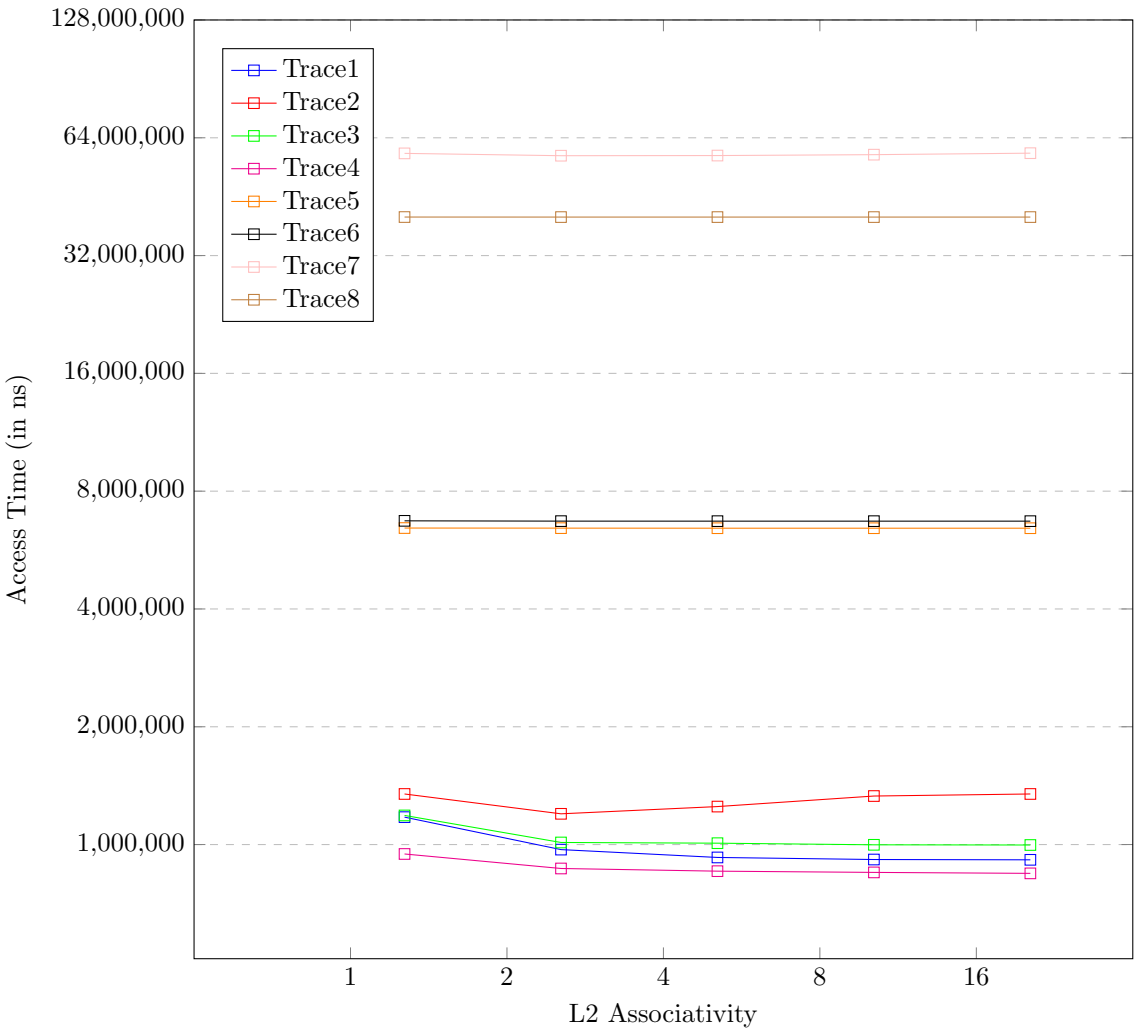

$\cdot 10^7$
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
Access Time (in ns)
Trace1
Trace2
Trace3
Trace4
Trace5
Trace6
Trace7
Trace8
16384
32768
65536
131072
262144
L2 Size (in bytes)

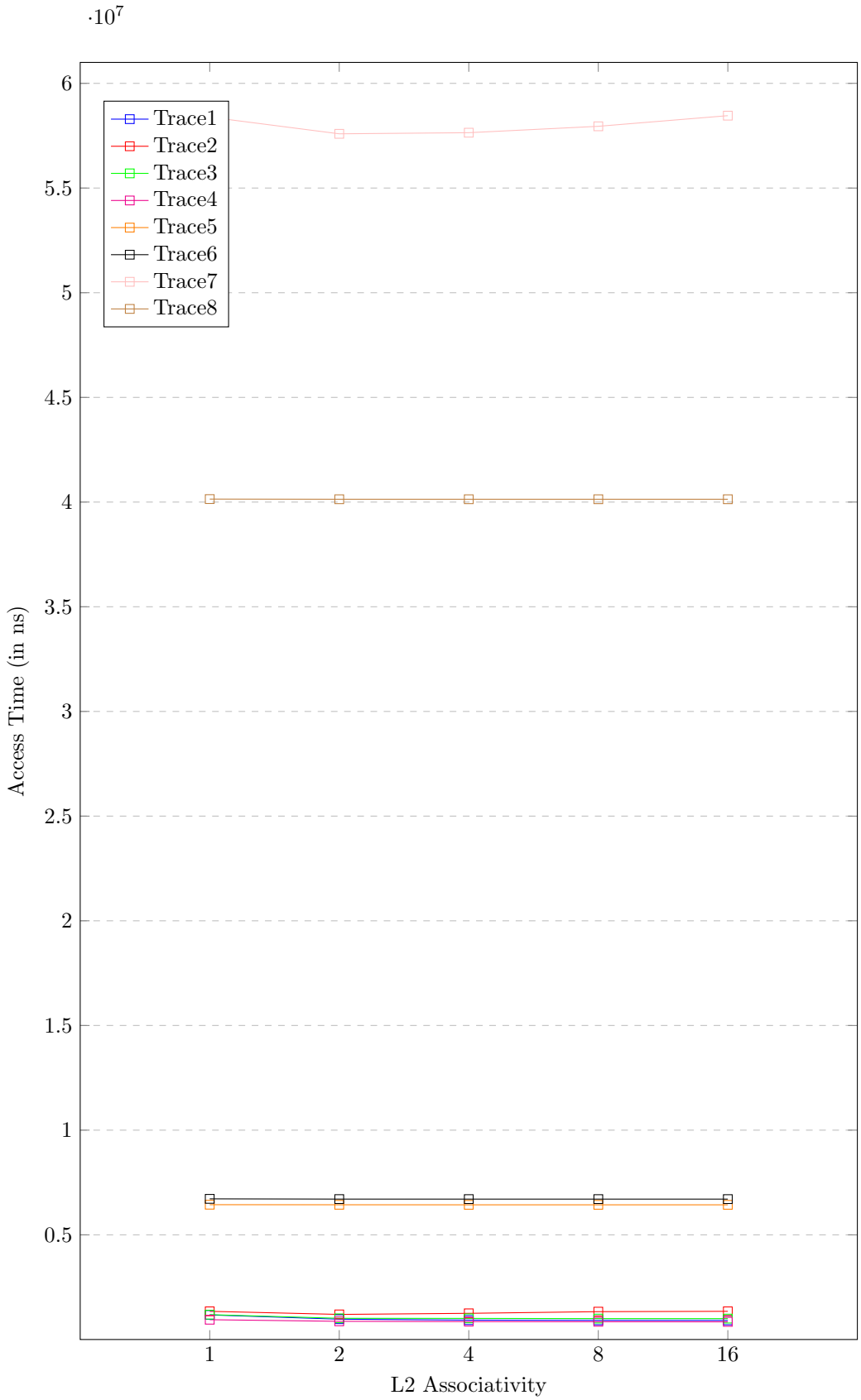
PLOT E : Variation of Access Time with respect to L2 Associativity

Observations from the Plots

- Keeping all other parameters constant (at Block Size = 64 bytes, L1 size = 1024 bytes, L1 Associativity = 2, L2 Size = 65536 bytes, L2 Associativity = 8), we vary L2 associativity.
- We observe that the latency remains almost constant with an increase in the associativity of the L2 cache.
- There is only slight variation in the case of Trace8: this is because it is the file involving most memory accesses and read/write operations, so the L2 cache is more active.
- Again, in this case, the effect of increasing associativity is as expected: a decrease is observed initially due to less conflict misses but latency rises gradually due to increased capacity misses.
- For the rest of the 7 trace files, effects are negligible.
- We observe that the effect of increasing block size is way more significant in the case of the L1 cache than the L2 cache: this is because every memory access involves the L1 cache but not the L2 cache.

Plotting from data:




$\cdot 10^7$
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
Access Time (in ns)
Trace1
Trace2
Trace3
Trace4
Trace5
Trace6
Trace7
Trace8
1
2
4
8
16
L2 Associativity