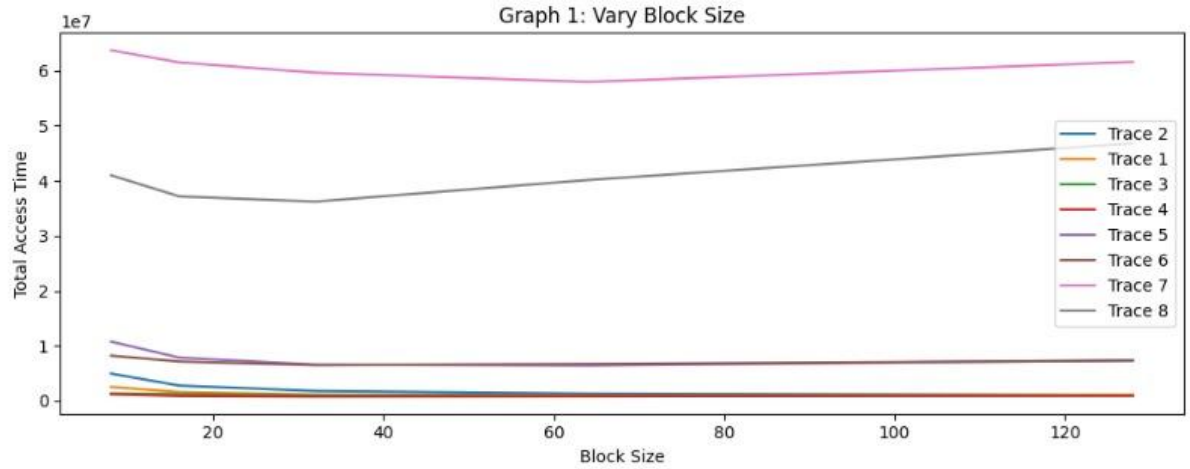
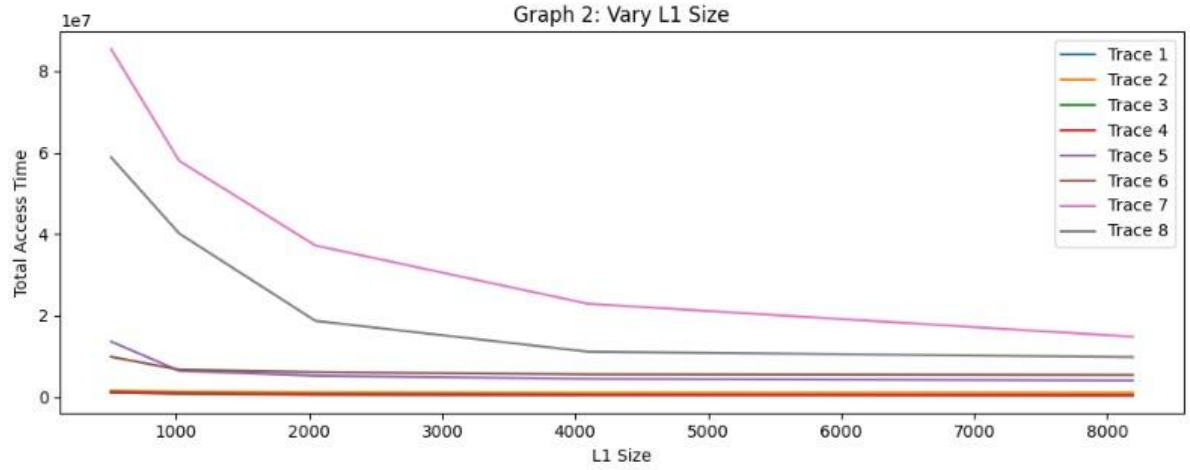
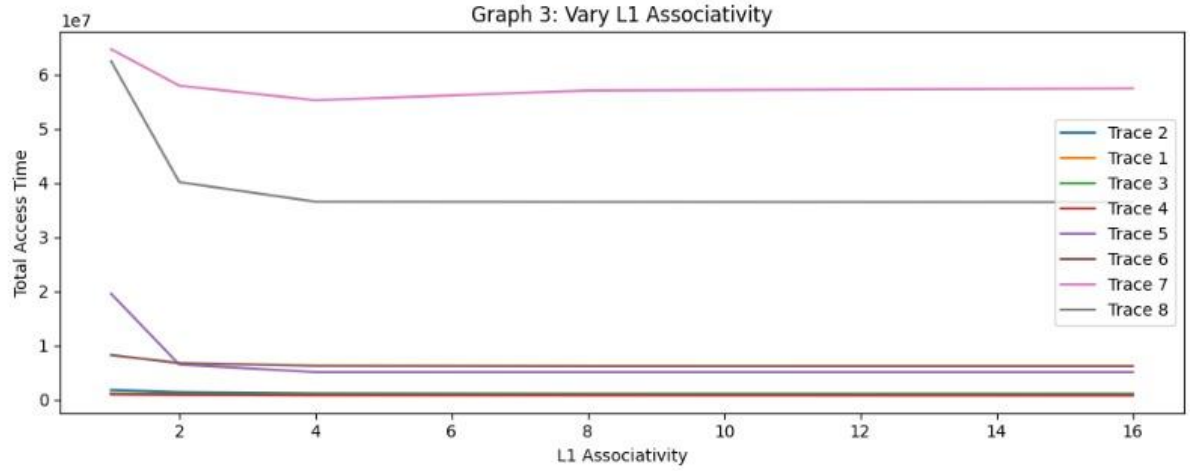
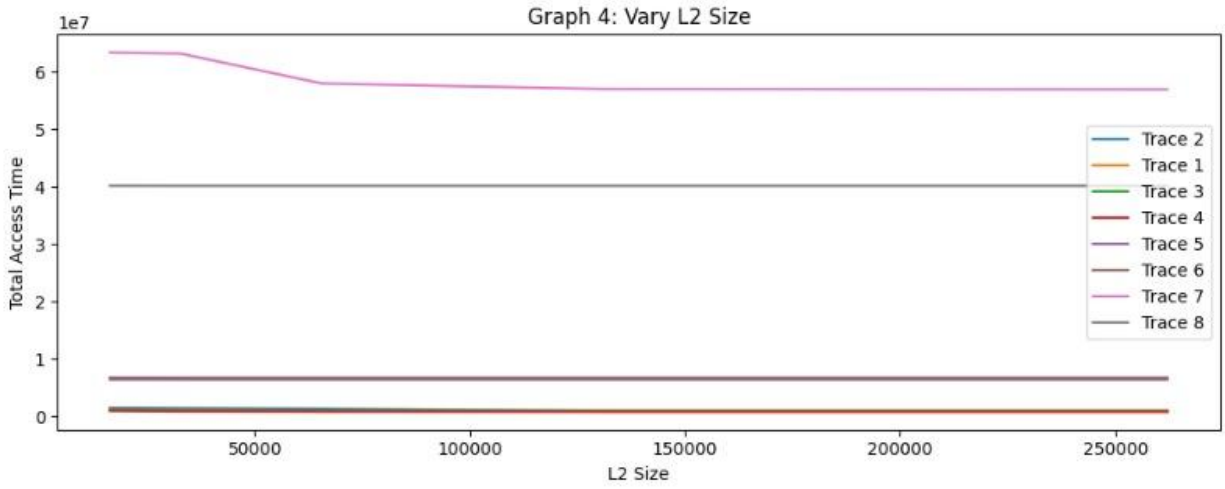
Cache Simulation Report

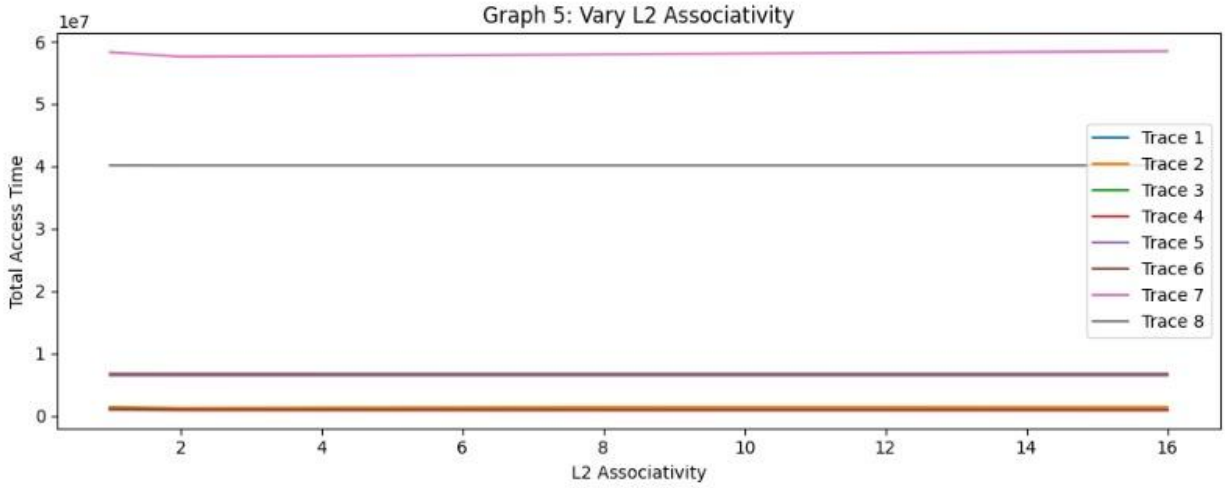
The graphs obtained by running the code on all eight trace files using different parameters for the cache are given as follows. The Y axes have Total Access time(in ns), and the X axes have the size in Bytes in the graphs where size varies.







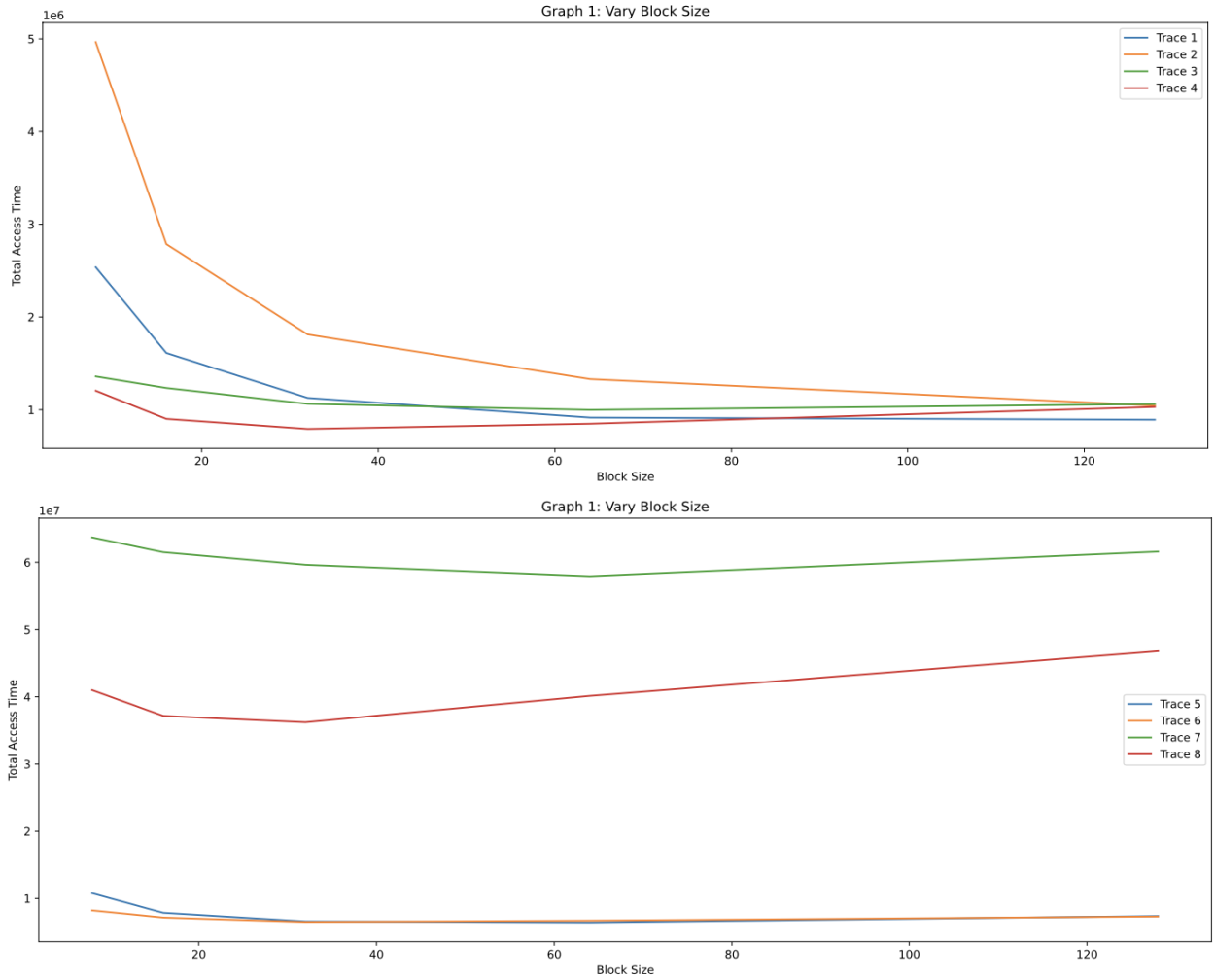




These graphs contain the plots for all eight files. However, the access times for the first few traces is lesser, and it is difficult to make any observations for those plots at this scale; they differ by a factor of 10. So we have provided separate plots for the first four and last four traces, as shown below. The observation and analysis for the following graphs also apply the corresponding graph for all files shown above.
The graphs were generated using Matplotlib; the code has also been included in the submission.

Observations:

Vary Block Size:

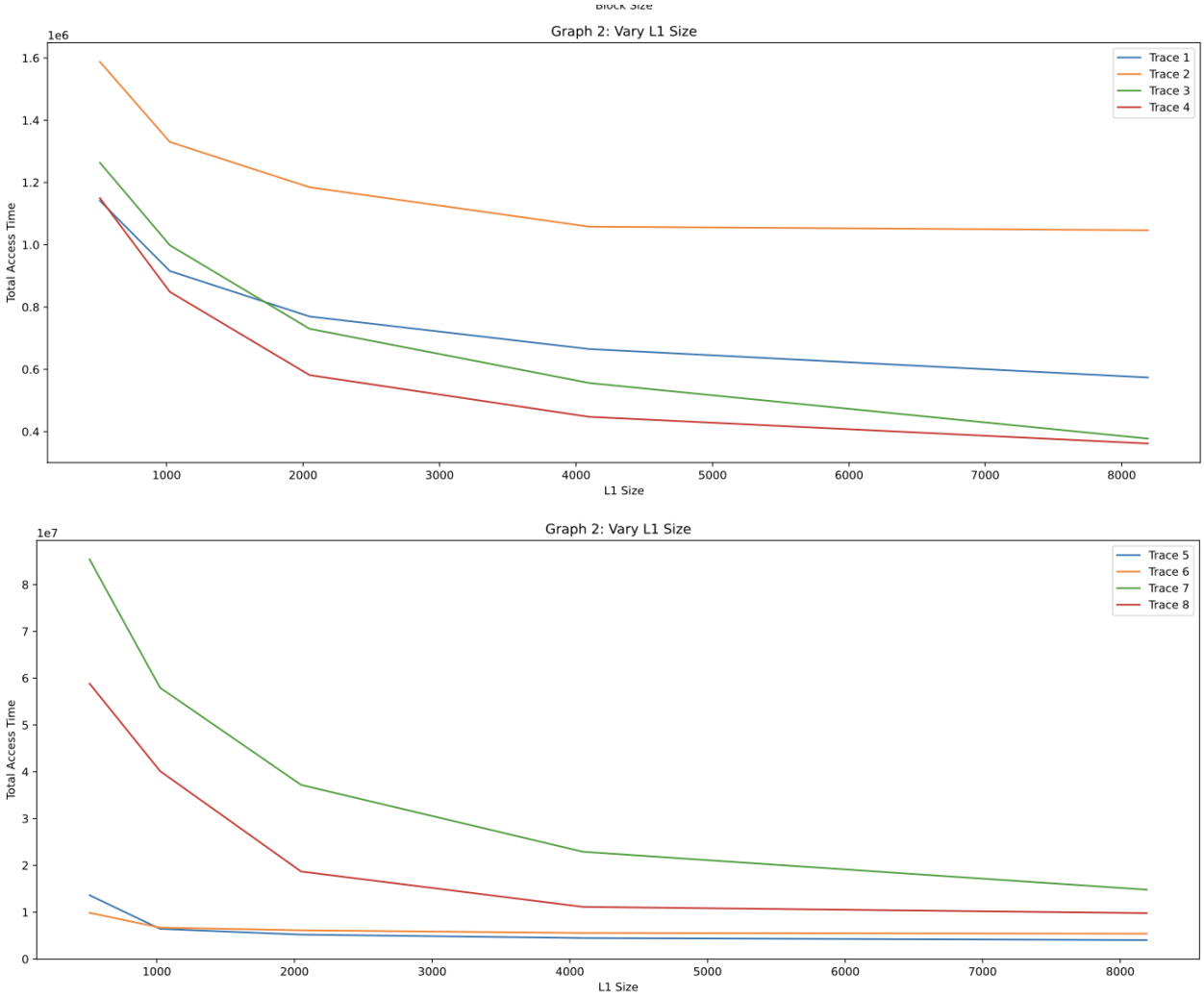


- The access time goes down at first and then increases gradually for 6 files, whereas there is a constant decrease for the other 2.

Increasing block size increases the chances of a cache hit because each block covers more area in memory. Hence, there is a better chance of finding a particular memory location. However, beyond a certain point, the size of these memory chunks will become so big that we will have to fetch a new block for each memory access, mainly since the number of sets will decrease due to the fixed size. This also depends on the trace in consideration. The best case would be that we are accessing nearby locations repeatedly, in which case larger block sizes are preferable. The worst case would be that we have only 1 or 2 (huge) blocks, and we alternate access locations in different blocks. So we will have more misses. If we had a smaller block size (keeping other parameters the same), we would have more sets, and we could get

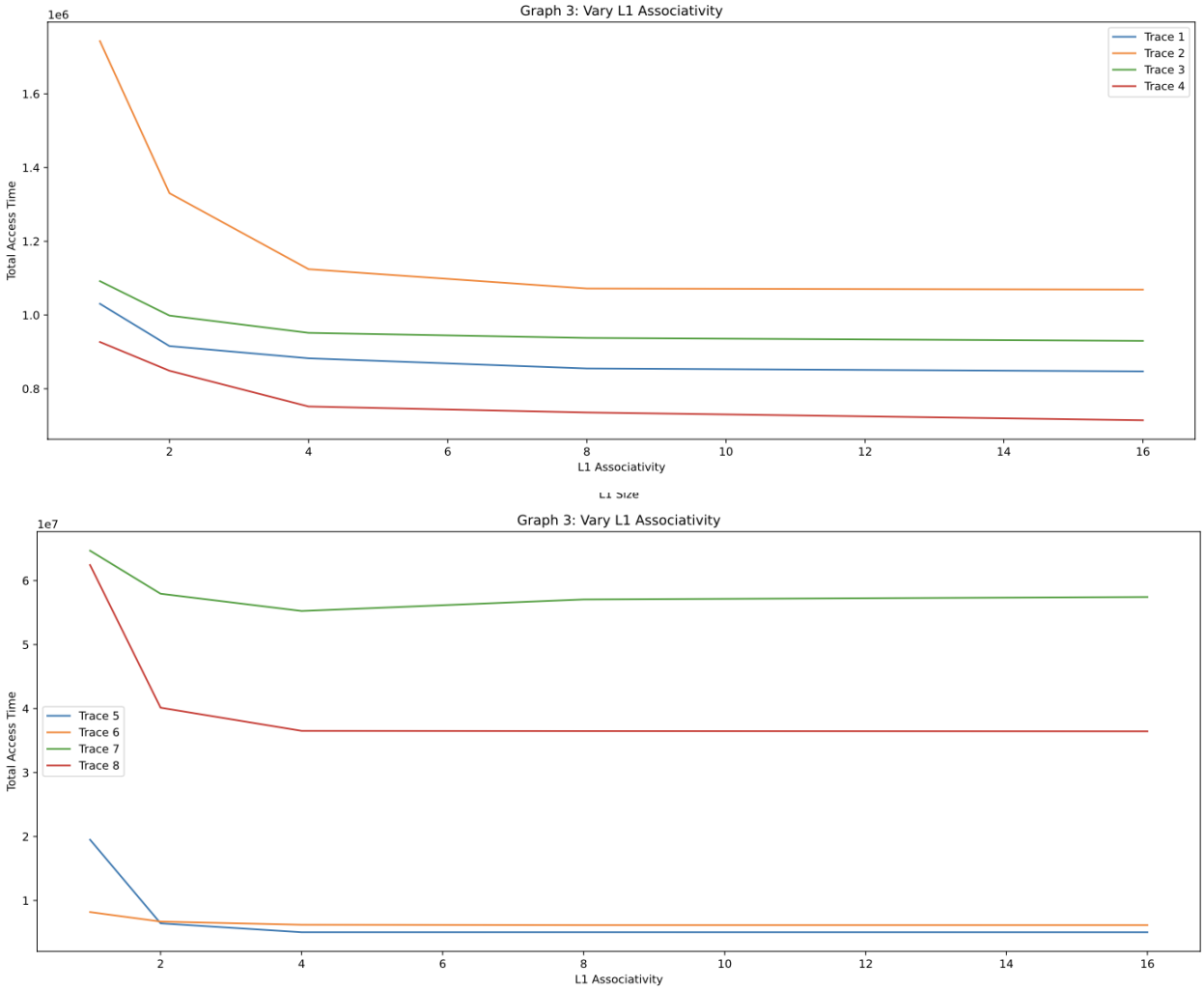
memory blocks from different regions of memory. All these factors are reflected in the graph as different trace files depict this variation in behavior.

Vary L1 Size:



- The access time decreases with increasing L1 Size. Initially, the drops are high, and then there are gradual drops. The simple explanation is that a larger cache can't perform worse than a smaller cache.

Vary L1 Associativity:

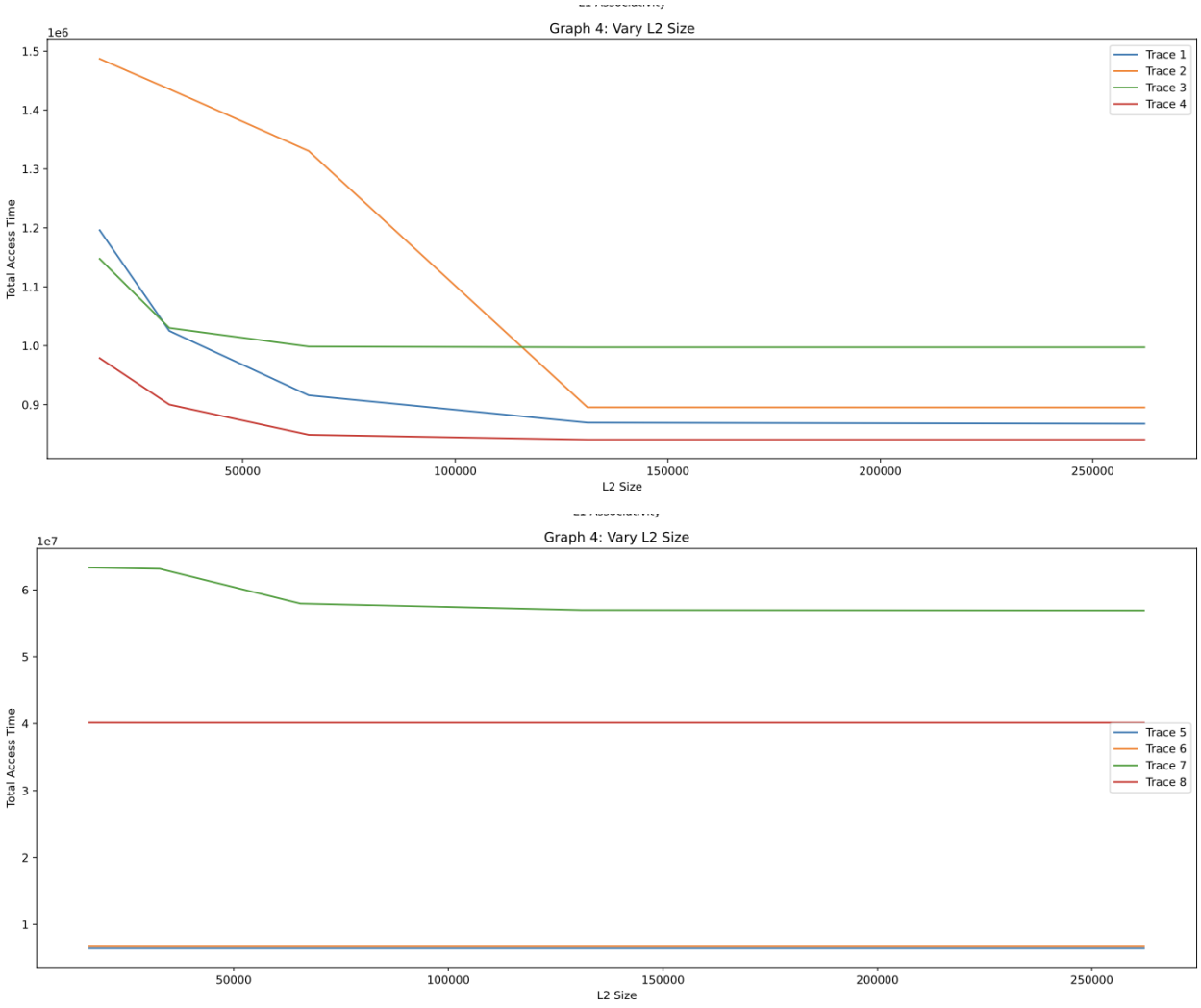


- The access time decreases sharply at first and then stays almost constant except for trace 7, which shows a nominal increase.

Very low associativity can be a performance bottleneck since an associativity of 1 will cause a conflict miss every time we access different locations with the same set id. This bottleneck is removed as soon as we move to a 2-way or 4-way associative cache. Beyond that, increasing associativity does not make much difference, meaning that most or all conflict misses have been removed, and no further improvement can be achieved without changing other parameters. The slight increase in trace 7 shows what can happen if we increase the associativity too much. A lower number of sets implies that more and more memory blocks will be mapped to the same set. There is a possibility that the set may get filled up very early on, and after that, writebacks due to eviction will increase the access time. Please note that this is

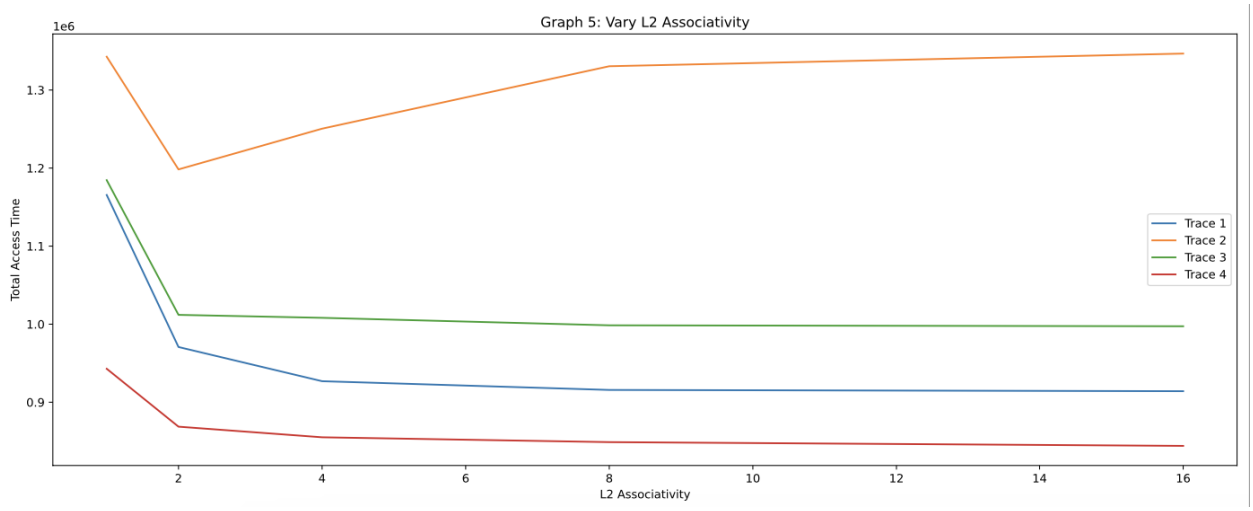
highly dependent on the trace file under consideration. These observations show why good programming practices are required to avail the full benefit of caches.

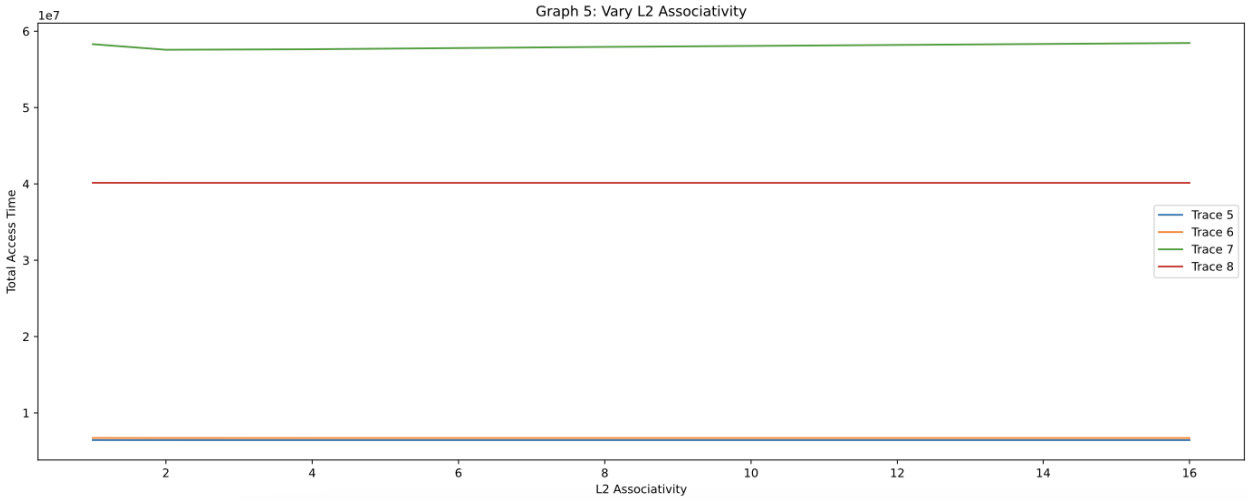
Vary L2 Size:



- The access time decreases sharply at first and then stays almost constant. The reason for the decrease is the same as the reason for the decrease in access time vis-a-vis L1 size. The constant line after a point is because there are a limited number of accesses to the L2 cache. Depending on the trace, after one point, we would have decreased the misses to only compulsory misses, and hence there would be no more performance improvement.

Vary L2 Associativity





- Access time for traces 1, 3, 4, 5, 6, and 8 decreases at first, then stays almost constant
- For traces 2 and 7, access time drops at first but then increases sharply and keeps increasing slowly?

The reason for the decrease is similar to that in the L1 cache. After one point, we managed to remove most conflict misses. The one anomaly in Trace 2 is because the total cache size is constant. As a result, as the associativity increases, the number of sets decreases(so does the number of set bits). This can cause more writebacks to the main memory(which is expensive). For example, once filled, a fully associative cache will have to evict one block for every new memory access. Depending on the trace, this can drastically increase the access time. Essentially we should expect access time to first decrease, and after one point, it should increase as we increase the associativity.

% distribution of work:

Name	Entry Number	Percentage Work Distribution
Parth Thakur	2021CS50615	50%
Amish Kansal	2021CS50622	50%