CACHE SIMULATOR ASSIGNMENT 3

Sakshi Gupta – 2021CS10567

Mrunal Kadhane – 2021CS10109

- ## Design Decisions

  - We have designed an n-way set associative cache in C++ consisting of L1, L2 and DRAM memory elements where L1 is a subset of L2 which is a subset of DRAM.
  - As an input, we are accepting a trace file which includes addresses and what operations (read or write) to be performed.
  - The simulator first makes a request to the L1 cache where there are different possibilities.
    1) **Read hit -** When data is found in the L1 cache for the read case, it gives us a hit and we break.
    2) **Read Miss –** This accounts for two cases depending on whether the set is completely occupied or not.

       i)Not completely occupied – We search for the block from L2. If found we bring the block into L1. If not, we bring the block from Memory - > L2 -> L1. There might be a case that bringing a block into L2 forces us to evict another block if that set is already fully occupied. Hence, we also have to remove this block from L1 as well to maintain Inclusivity.

       ii)Completely occupied – We evict the least recently used block into L2 and then continue searching from L2 or from memory (if not found in L2) and bring the block into the place of that evicted block.

    3) **Write hit -** When data is found in the L1 cache for the write case, it gives us a hit and we break.

    4) **Write miss -** This accounts for two cases depending on whether the set is completely occupied or not.
       The procedure to be followed is same as that for a read miss with the difference being that we are now updating the values in the L1 cache and hence we have to be careful with the dirty bits.

Design for L1 cache

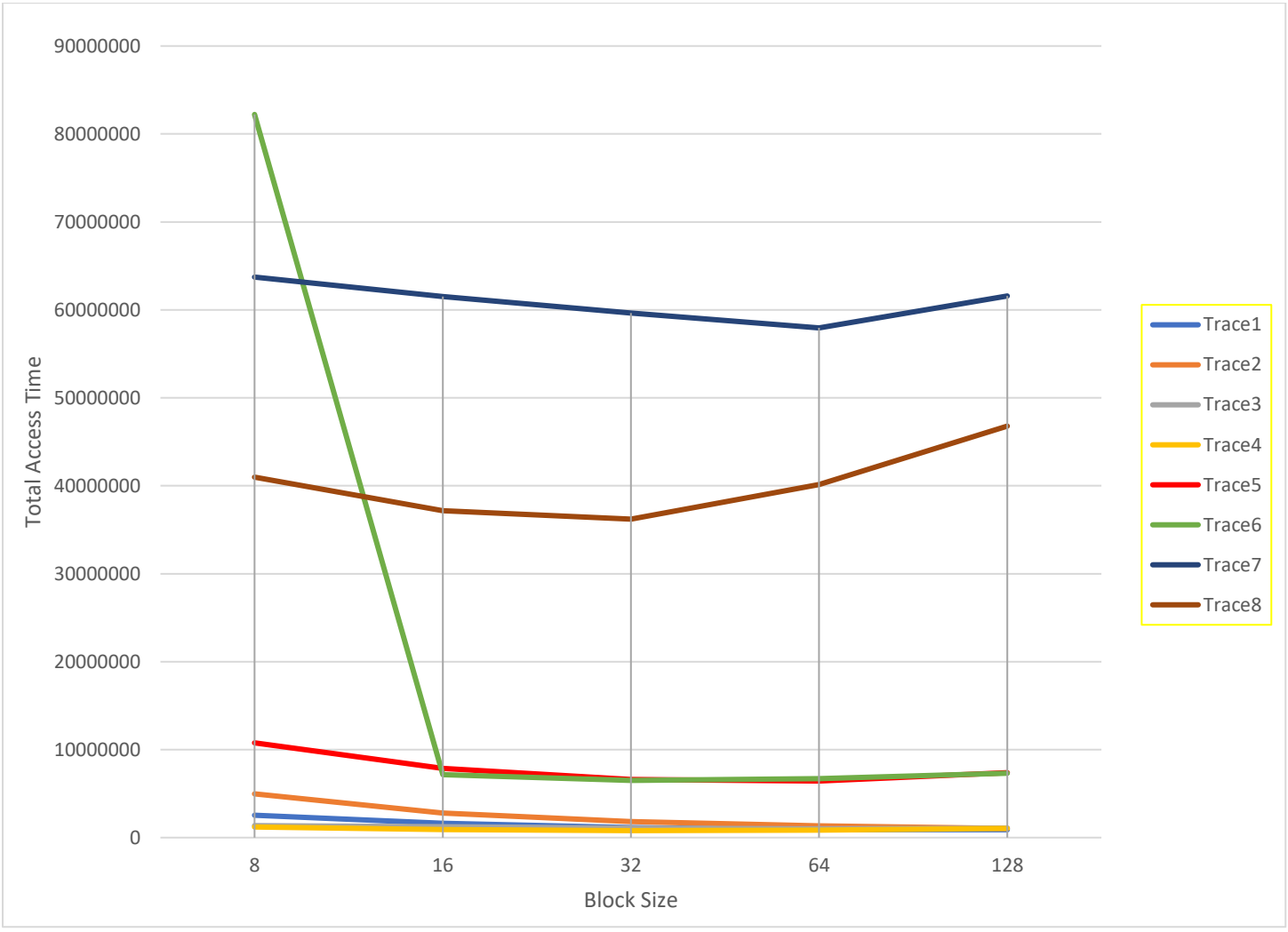
Read Miss

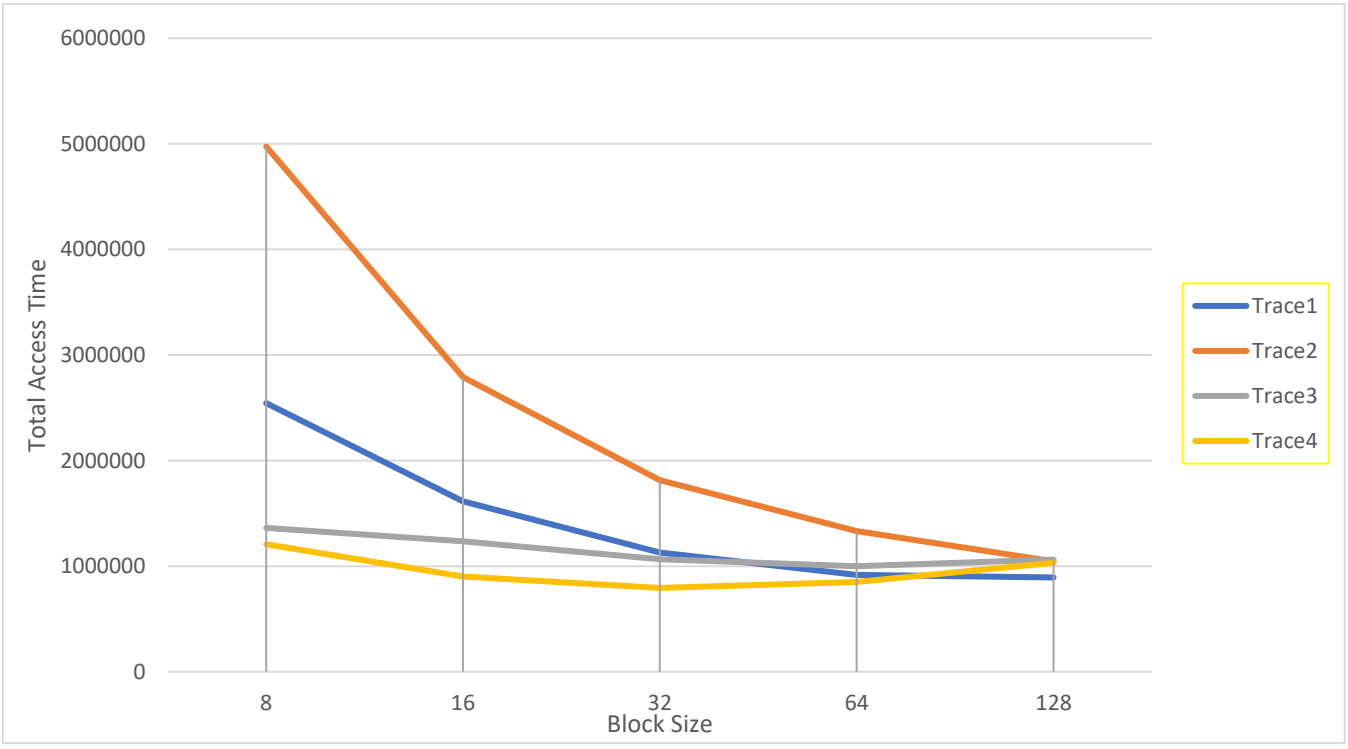
- compulsory miss (for case when we invalid_bit = true) → No data is evicted and is taken from L2
- read miss with dirty_bit == true (whose recent call time is least) → In this case first there is writeback to L2 and then data is evicted based on LRU. (making L2, dirty bit == true)
- read miss with dirty_bit == false (whose recent call time is least) → directly data is evicted on the basis of LRU

Read Miss for L2.

- Compulsory Miss. (SAME as L1 but with condition that data needs to be transferred to L2.)
- Read miss (dirty_bit of LRU == true) → first write back to Memory DRAM and the read & evict. And providing L1 with data.
- Read miss (dirty_bit == false) → In this case data in L1 is also looked for since it would be present in L1 and to make it subset that needs to be evicted. And providing L1 with data.

- **PLOTS**





Varying the **Block size** on the X – axis keeping the other parameters same

Total access time is on the Y axis

L1 size – 1024

L1 associativity – 2

L2 size – 65536

L2 associativity -8

Observation:

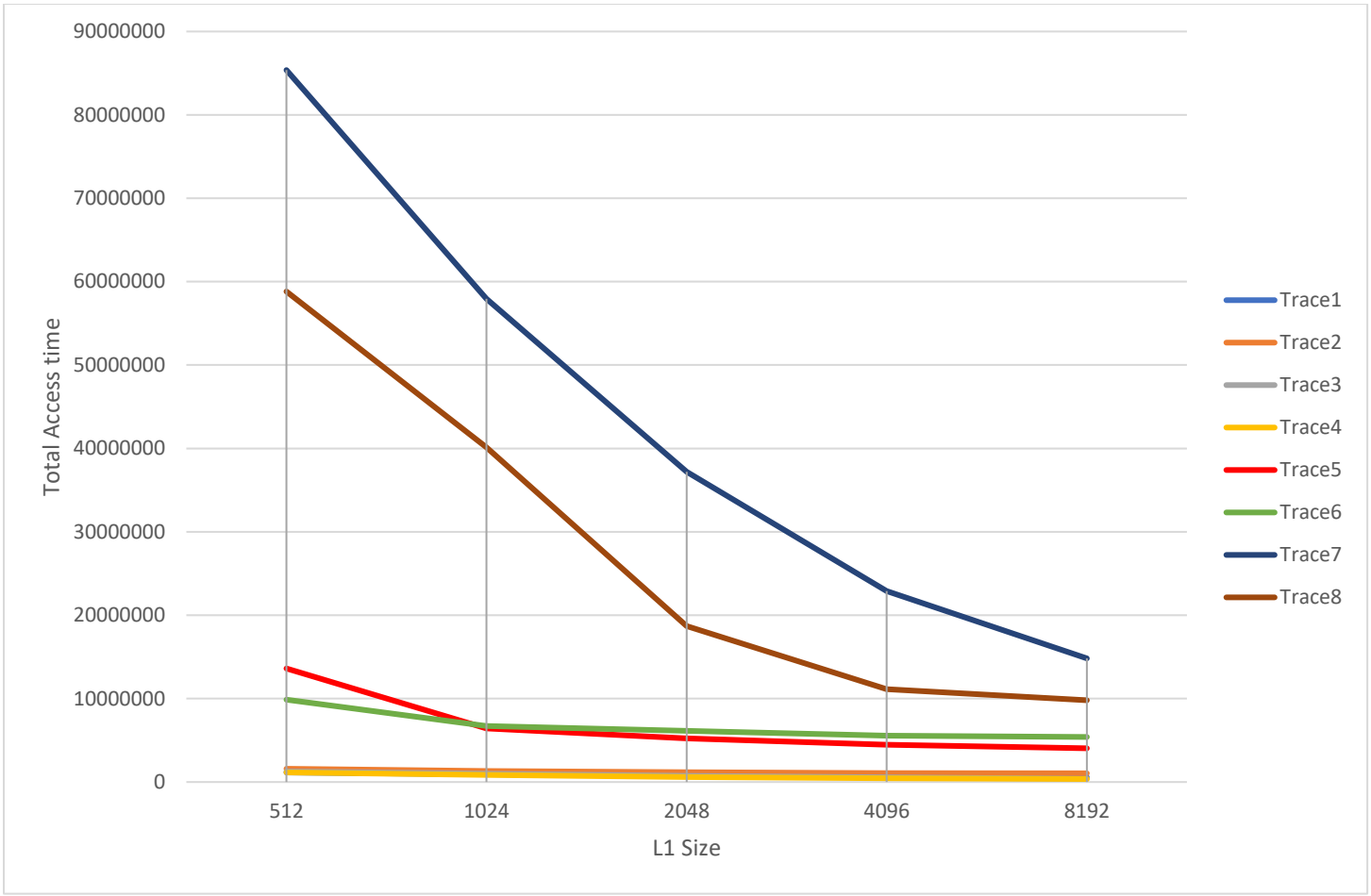
1) With increase in block size at initial stages, we get a decrease in the total access time since the number of hits increases as more neighbouring data gets fetched when we bring a block into L1 cache. Since the caches are empty initially, bringing a huge chunk of data favours the cache by leading to a hit.

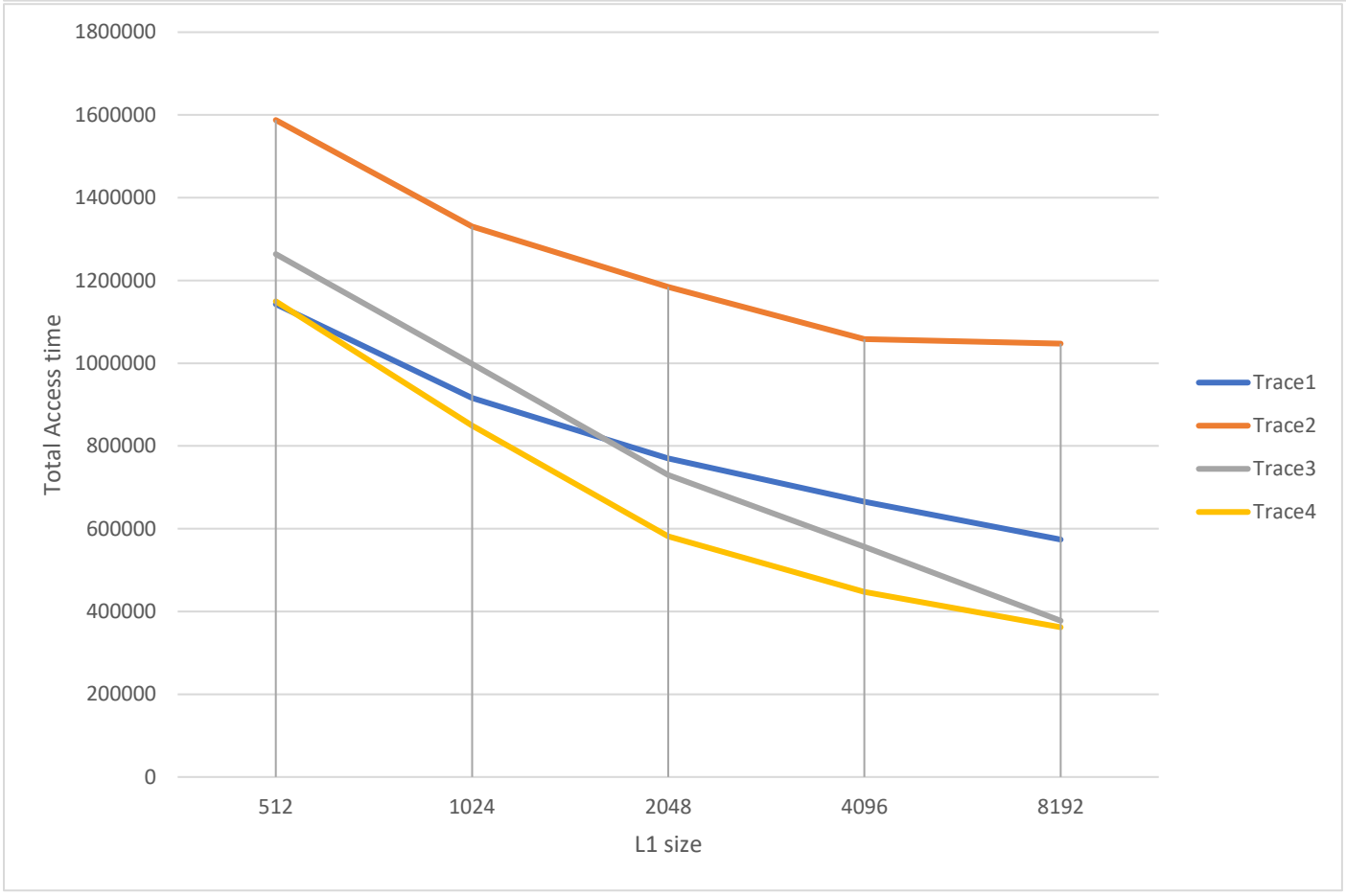
2) At later stages, we see an increase in the total access time. As the caches gets occupied, we feel the need to evict least recently used block to make room for the block with the required address to come into the L1 cache. In this process, we are also evicting huge chunk of data which results in a miss for several addresses. These misses contribute to the access time. The amount of data stored leads to a data pollution since most of the data is not required.

3) Also, in actual scenario by increasing the block size, the time required for accessing the data also increases due to size of the data.

4) Increasing the block size, the hit rate by increasing the spatial locality of reference. While decreasing the block size can improve the hit rate by increasing the temporal locality of reference.

5) One more point directly could be observed is the minima in the graph which tells that increasing the block size too much could also increase access time due to increase in miss rates because of pollution of data and less block size also results in miss rate due to unavailability.





Varying the **L1 size** on the X – axis keeping the other parameters same

Total access time is on the Y axis

Block size – 64

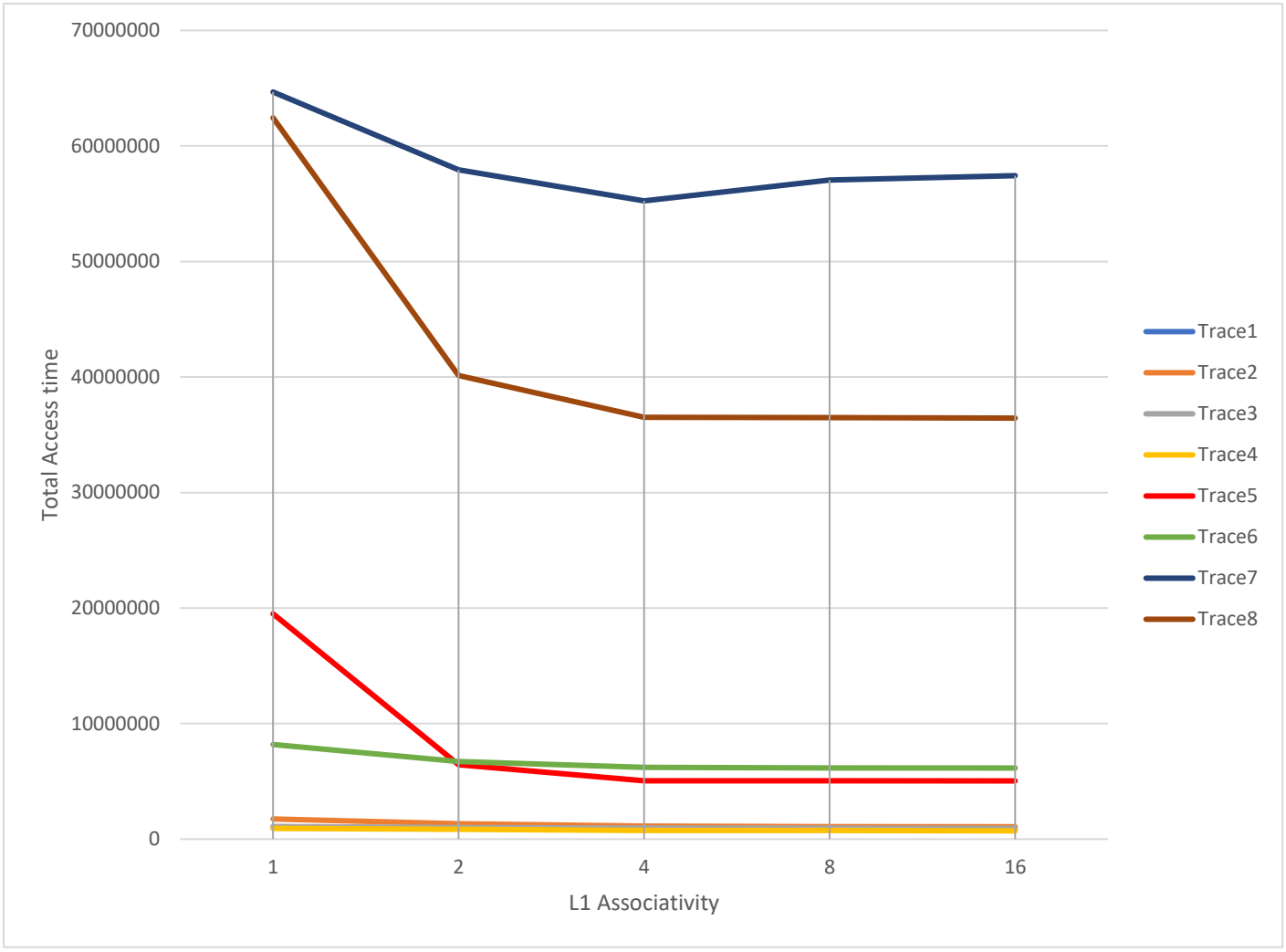
L1 associativity – 2

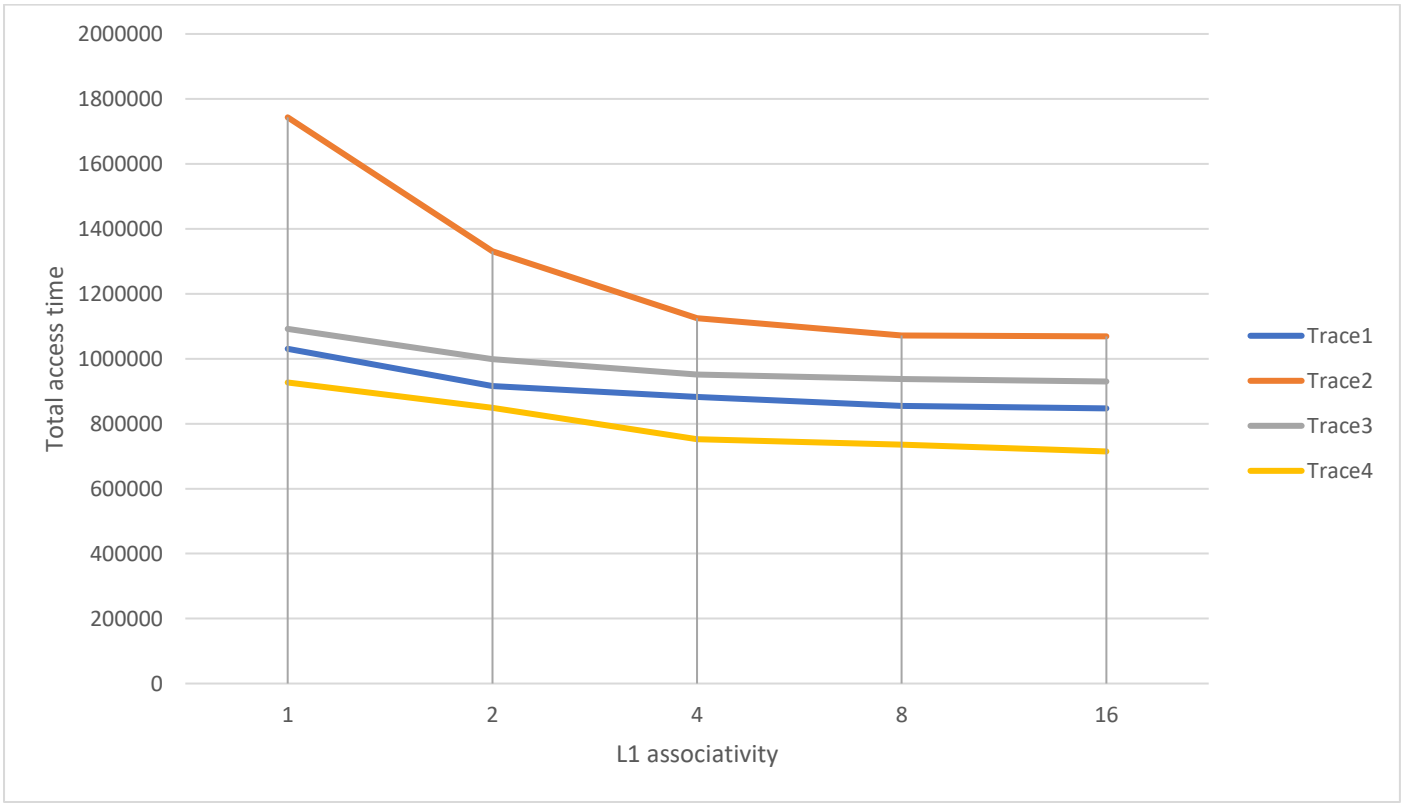
L2 size – 65536

L2 associativity -8

Observation:

1) Increasing the L1 size means we can store more data in the L1 cache which saves us the time of accessing the memory for the read and write address. This is observed with a decrease in the total access time across the trace files.
2) The cache hit rate increases due to increasing L1 size since its stores more data, decreasing the misses
3) There could be a disadvantage also since if we keep increasing space the search space for cache would increase thus increasing access latency. Though this is not evident in the graph due to its low scale.





Varying the **L1 associativity** on the X – axis keeping the other parameters same

Total access time is on the Y axis

Block size – 64

L1 size – 1024

L2 size – 65536

L2 associativity - 8

Observation:

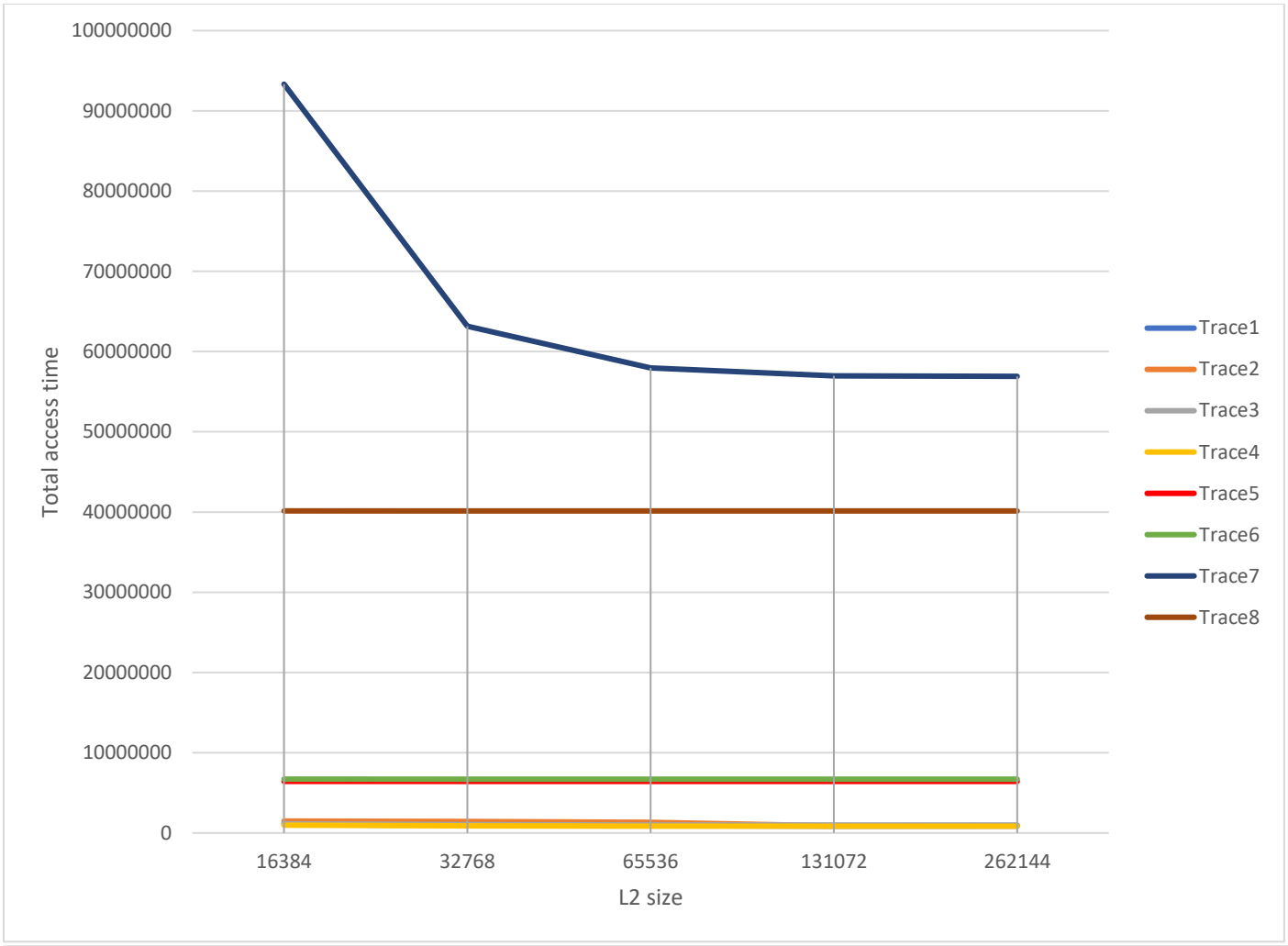
1) Increasing the associativity decreases the chances of eviction since there is more place for blocks to accommodate in the same index. Hence there are more read hits which decreases the total access time and increasing the hit rates for L1, initially we could see a steep slope in the graph but is not evenly distributed for all traces depending on the data the traces hold.

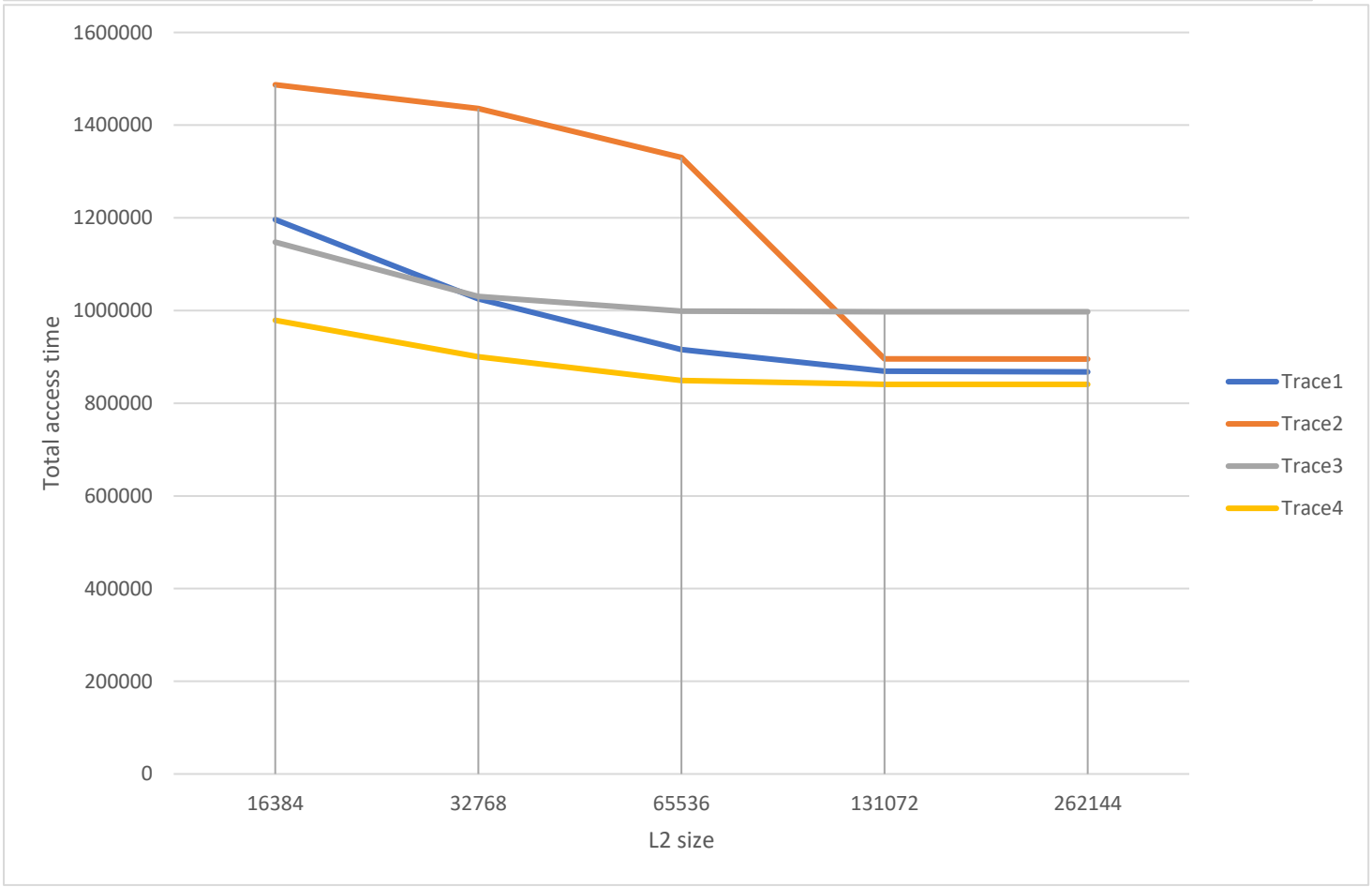
2) Initially we are decreasing the access time as for the data which occupy same place in cache, is no longer needed to be evicted and could be accessed.

3) We see an increase in access time for some test cases. Since the associativity has increased, the loop which runs along a set to search for the required tag has to iterate over a larger number of values. The cache is inclining on transforming into a fully associative cache (it is not in the given test cases) and thus faces an increase in time as more data gets fed into the cache.

   This is because the search space has increased by associativity now, we need to look and match up the tag, to find the data.

4) By increasing the associativity, we are decreasing the number of sets it could store, and would increase the miss rate.
   This is clearly visible in the graph that till a certain point slope is steep but after that it remains constant and then decreases.





Varying the **L2 size** on the X – axis keeping the other parameters same

Total access time is on the Y axis

Block size – 64

L1 size – 1024

L1 associativity – 2

L2 associativity - 8

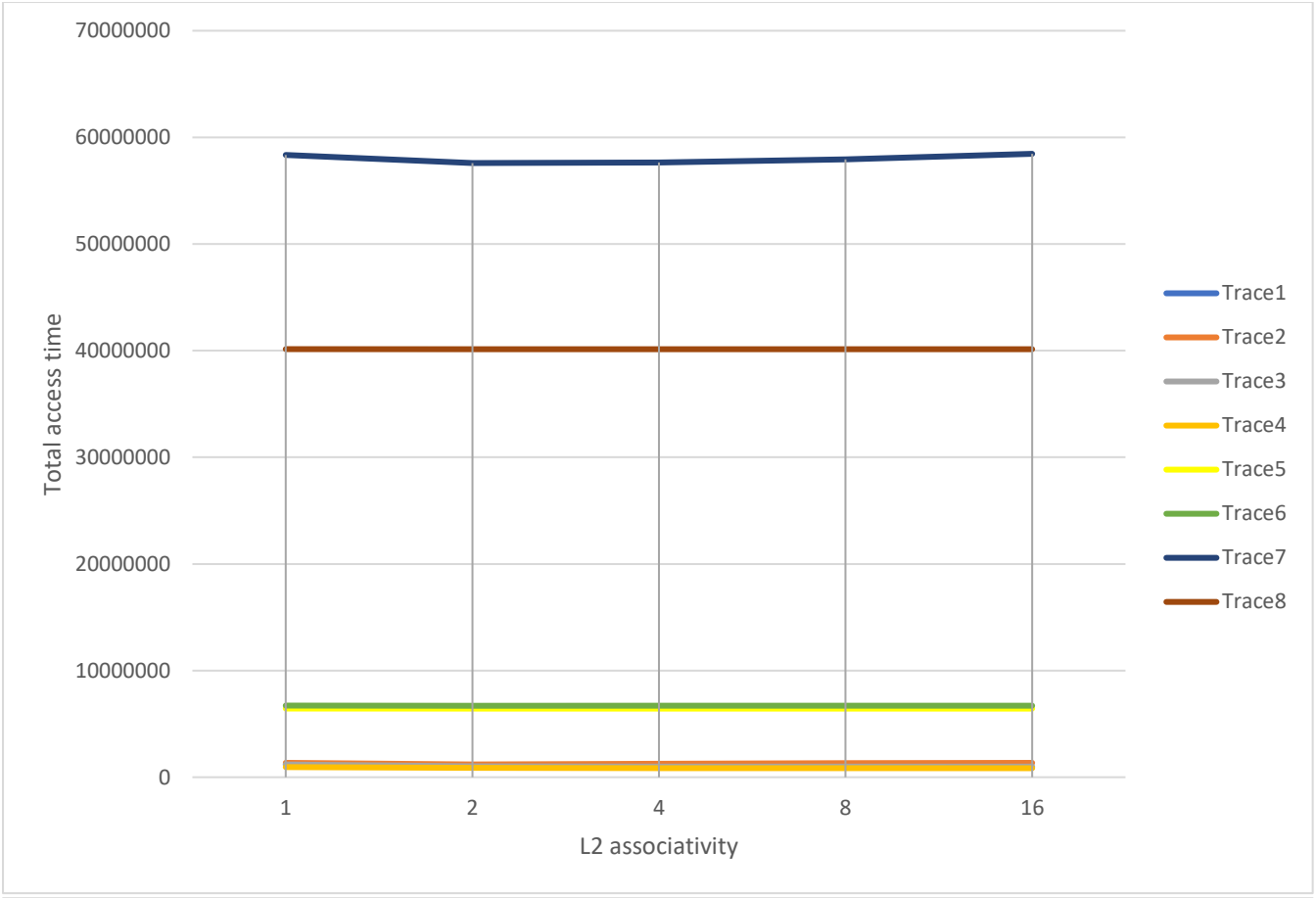
Observation:

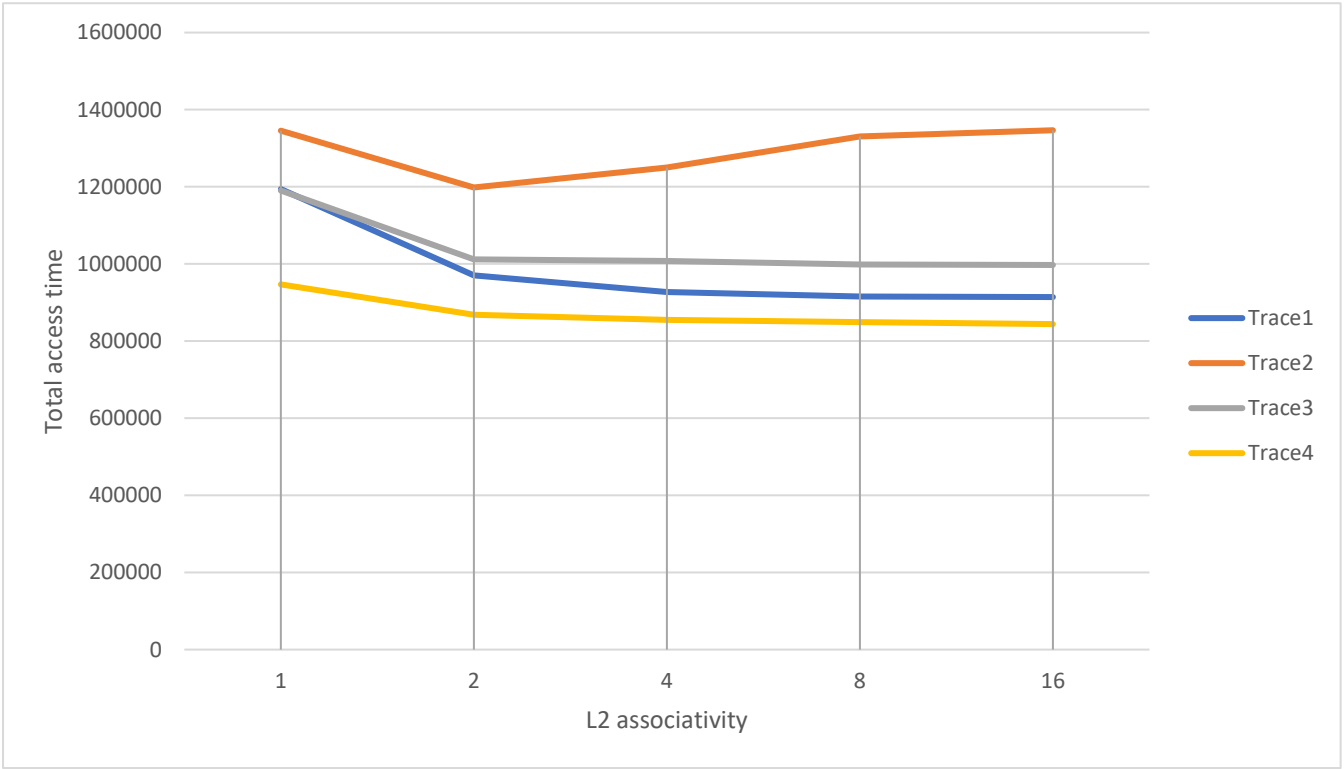
1) We see that for most of the test cases, there isn't much difference in the total access time even with an increase in the L2 cache size. This is because we are directly accessing L1 cache and hence the read and write hits hold more importance for L1 cache. L2 cache is our next immediate point of contact and hence holds some importance to determine access time. With increase in L2 cache size, the access time will decrease since more data will be there in L2 to be accessed by L1 and this will save us the time of going to the memory.

2) The reason for time becoming constant could be the data we required is mostly available in L2 since its size is increased so we have our required data in L2 and time for memory is no longer making difference due to availability of data in L2 and hence the graph is coming constant for larger caches.

3) Since the size of cache is increased the access latency increases.

4) With a grater cache size, more data needs to be transferred between caches and hence the memory bandwidth also increases. Hence the processor might not keep up with this increased demand.





Varying the **L2 associativity** on the X – axis keeping the other parameters same

Total access time is on the Y axis

Block size – 64

L1 size – 1024

L1 associativity – 2

L2 size - 65536

Observation:

1) We see that for most of the test cases, there isn't much difference in the total access time even with an increase in the L2 associativity. This is because we are directly accessing L1 cache and hence the read and write hits hold more importance for L1 cache. The advantage we get from greater associativity i.e. of less chances of eviction of a block does not hold a 'huge' impact here since eviction from L2 is only during writebacks.

2) Even with that fewer advantage, we see a narrow decrease in the total access time initially since the cache does not hold that much information. As more data is fed, we need more time to search for a certain block to bring into L1 because the L2 cache is inclining towards becoming a fully associative cache. Hence there is a slight increase in total access time in the later stages for some test cases.

3) As could be seen in trace 2 after the increasing the associativity the time increases, this is because the number of sets decreases and the required value get evicted.

4) A case we could also observe where the total access time remains same for previous stage and this stage, it does not have any impact on changing the associativity and size, the reason might be it is not much communicating with L2, either the required data is available in L1 or the size of L2 is vast enough that after a point it does not matters how big size the spatial and temporal locality are taking care of.