COL216 Cache simulation report

Dhruv Ahlawat, 2021CS10556
Kartik Meena, 2021CS50618

May 2023

1 Design Decisions

1. When writeback is required due to evicting a dirty block, we first evict it and writeback to the next level cache, and then read the data from the next cache. We did this because this made the outputs match exactly with the one's given on piazza. On using a buffer for storing the written back value and reading first then writing back, we noticed that the efficiency was very slightly increased, but we did not employ this method.

2. we made a single 'Cache' class for all L1, L2 and DRAM memories, to make it modular. the class contains pointers to the next level of memory that it must take the values from, so L1 has the pointer of L2, and L2 has the pointer to DRAM.

3. LRU is implemented using a simple counter, every time a block is accessed we set its last used time to the current timer variable, and increment the timer variable. This allows us to find the LRU block in $O(n)$ time where n is associtavity. the counter is implemented as a long long so it won't overflow. Also, O(n) is the best we can do as to realize we will have to traverse through all of the n 'ways' to determine if the block placed there matches with our block, which is itself $O(n)$.

4. We kept all the Data-structures consistent for it to be actually used in data simulation (actually doing read and writes to blocks from commands), but since the assignment asks only for the simulation of the time and hits/misses, we have left out the data part (on calling a read, there is no actual data being sent back, or any actual data being written on write).

5. An important fact we have considered is that, the access time mentioned are the overall access time. For example suppose we ask for a read and it generates an L1 read miss, and then an L2 read hit which sends data back to L1 and L1 stores it and then sends the data to the processor. In this scenario, the time taken would be access time for L1 + access time for L2, even though L1 first checks for a value, and then even takes a value from L2 and stores it, and then returns it. We assume all of these steps are covered in the access times mentioned.

2 Observations

For the following observations, the corresponding graphs have been separated into 3 parts for showing more detail, as if they were on the same y-scale then the entire graph would look compressed and there would be much of white-space. The full sized graphs with all traces in them have been added towards the end of the document.

The graphs are logarithmic on the x and y axis.

2.1 BLOCK SIZE

when the Blocksize of the caches are increased, the effect of spatial locality will have more effect and generally the access time will be less. But since size is kept constant, this means that the overall number of blocks and sets are reduced.

1. Traces 4,5,6,7,8 :- in the plotted graph we can see increasing the Blocksize initially reduces the total access time but soon conflict misses dominate over the spatial locality resulting in increasing total access time because number of total sets are reduced in the cache, which causes more conflict misses.

2. Traces 1,2,3 :- These generally show a decline in the total access time, which signifies that these traces must have a lot of spatial locality. (nearby memory locations are accessed frequently)

3. on comparision between traces, it can be seen that traces 5 and 6 reach nearly the same total time at Blocksize = 128, while traces 2,3,4 reach the same total time at Blocksize = 128. trace 1 is the third highest at Blocksize = 8 but at Blocksize = 128 it becomes the one with the least time. This means that Trace 1 has a lot more spatial locality than the other traces, and hence its time decrease is the most rapid.

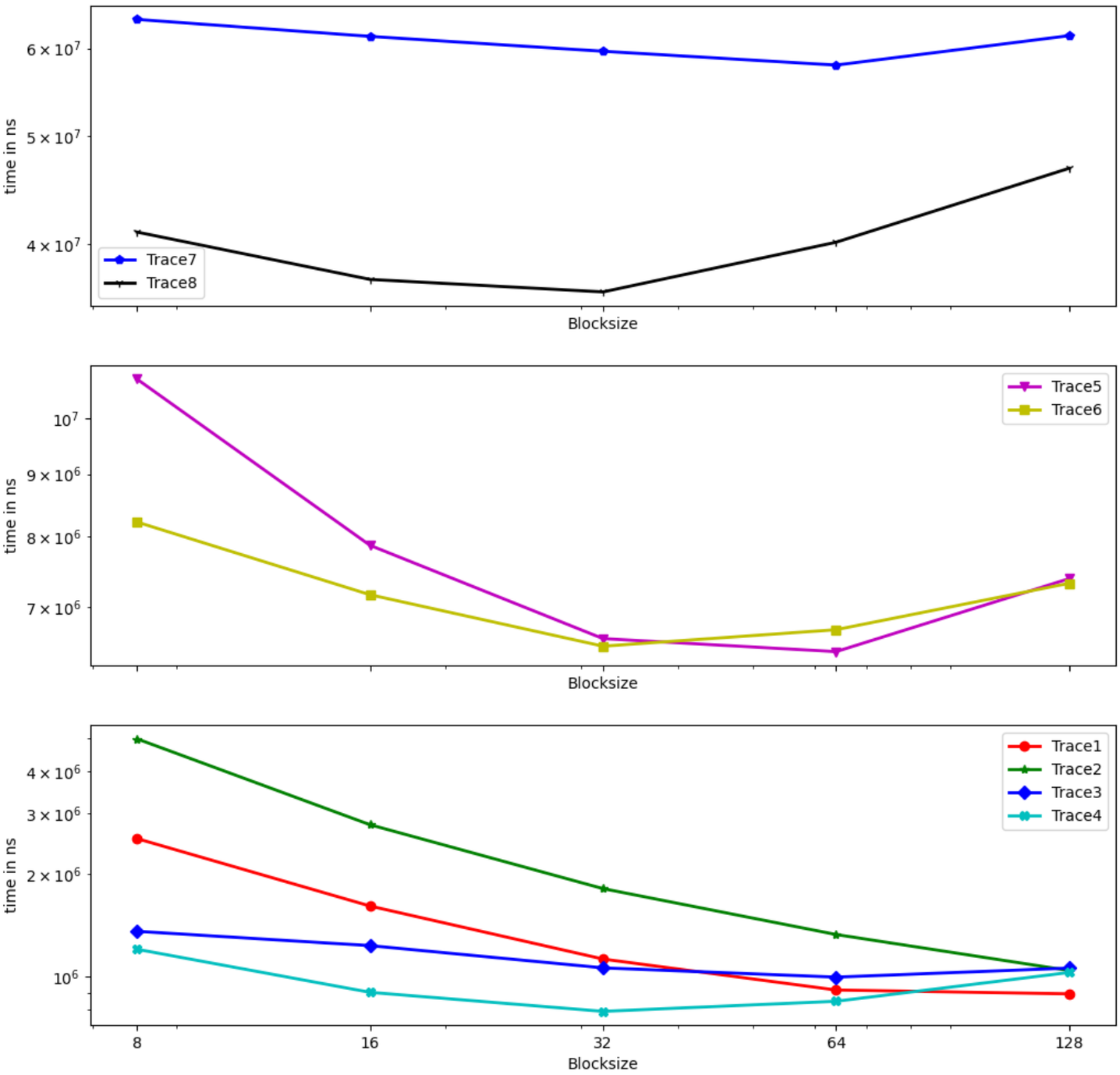

Figure 1: time v/s Block size

2.2 L1 cache Size

increasing cache size can lead to better performance in computer systems because larger caches can store more data, allowing the CPU to access frequently used data more quickly as it reduces the capacity misses.

An important point to consider in this comparison is that the graph plotted here is only theoretical in the sense that we assume the access time to be constant. In reality, on increasing cache size, we are bound to increase cache access time as well(which is why L2 has much higher access time than L1).

1. traces 1,2,3,4,6 :- these show downward decline in access times as expected. Capacity misses are reduced.

2. traces 5,7,8 :- these show more decline with increasing size (which is more visible in the graph present

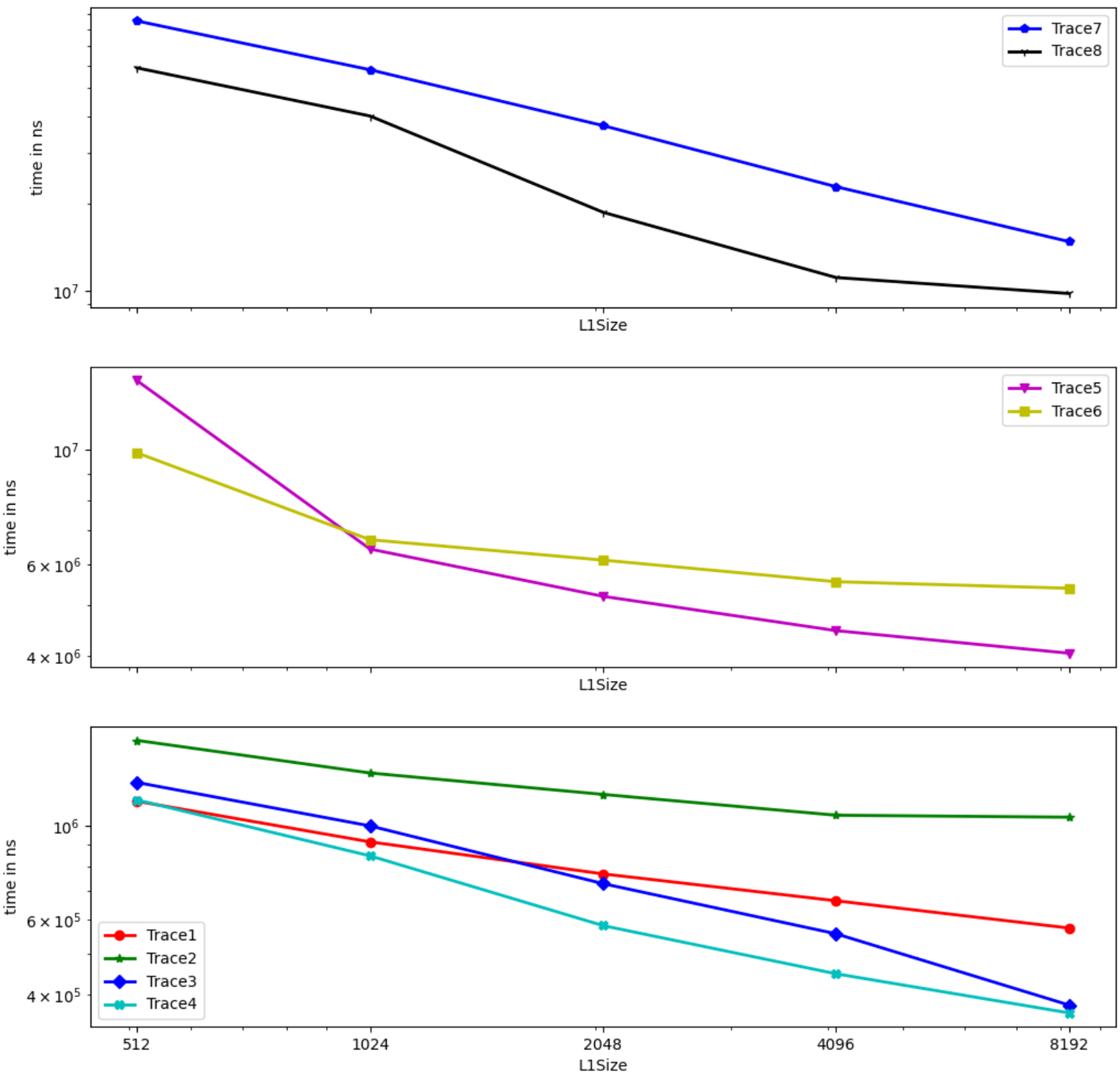

Figure 2: time v/s L1 cache size

at the end of the report). This means that in these traces, conflict misses in L1 were more of an issue in these traces compared to the others, which was reduced dramatically on increasing L1 size.

3. Between trace 3 and trace 1, its seen that on increasing L1 cache size, the total access time for trace 3 becomes lesser than that of trace 1. hence we can say that trace 3 had more capacity miss issues than trace 1 and hence it steeply declines on increasing L1 size.

4. similarly for traces 5 and 6, Trace 5's total time becomes lesser than that of trace 6 meaning that it had more capacity miss issues originally which were dramatically reduced on increasing the L1 cache size.

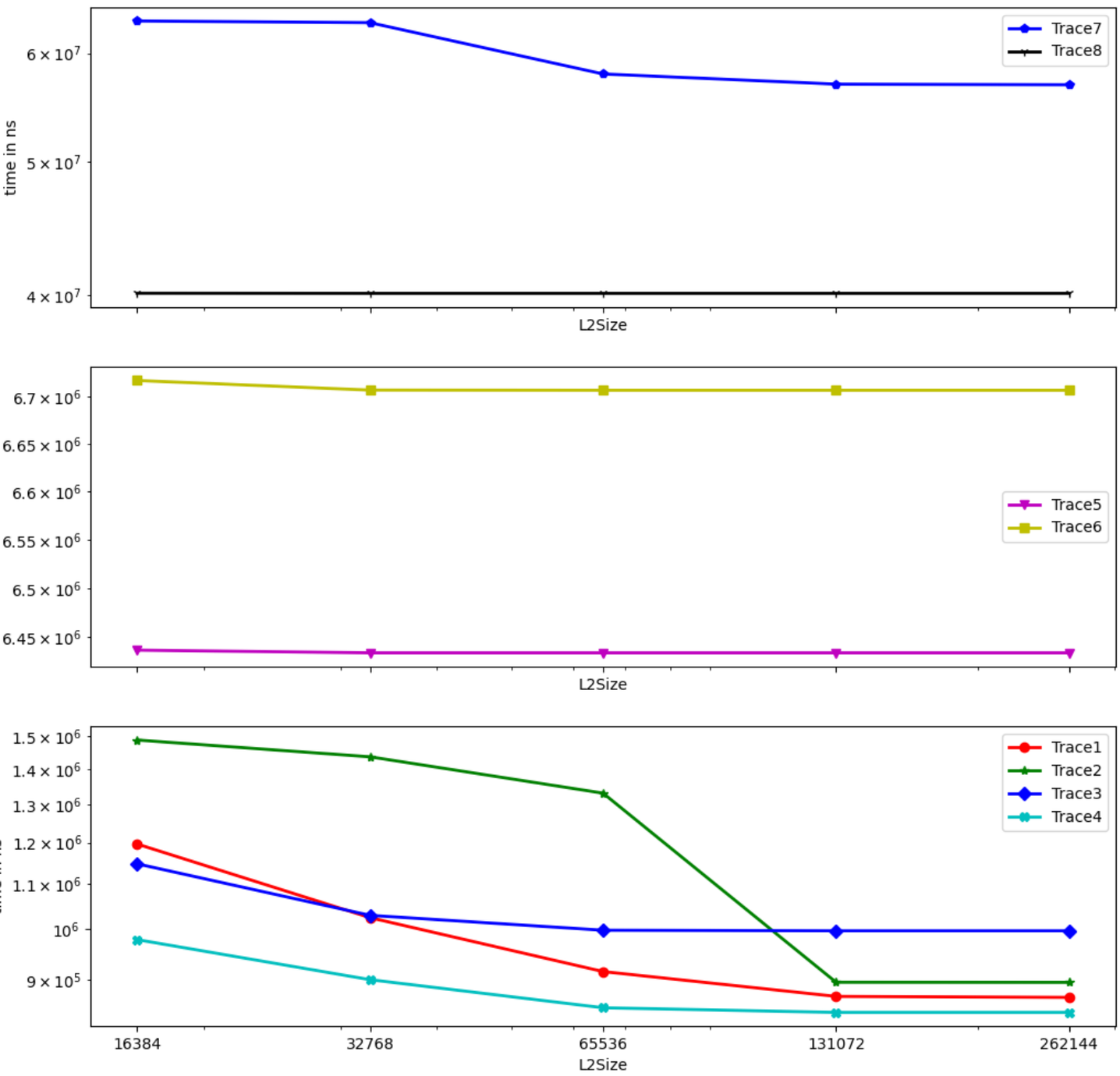

Figure 3: time v/s L2 cache size

2.3 L2 cache Size

1. traces 1,2,4,7 :- show a considerable decrease in time on increasing the L2 cache size. Among these trace 2 and 7 have a very large change at size=131072 and size=32768 respectively. This signifies that these 2 trace files were utilising L2 more, and had frequent capacity misses which disappeared on changing the size to the values 131072 and 32768, resulting in the sudden large decrease in access time.

2. traces 5,6,8 :- these have nearly constant times compared to the changes in other trace files. This means that in these traces, L2 cache wasn't being utilised as often as in the other traces or its misses were not due to capacity misses, hence there is only a slight change in the overall access time.

3. on comparison between traces, it can be seen that trace 3 which was at first the second highest time,

becomes the 4th highest at the end. This means that it either did not use L2 as much or its misses in L2 were less due to capacity misses, unlike the other traces.

2.4 L1 associativity

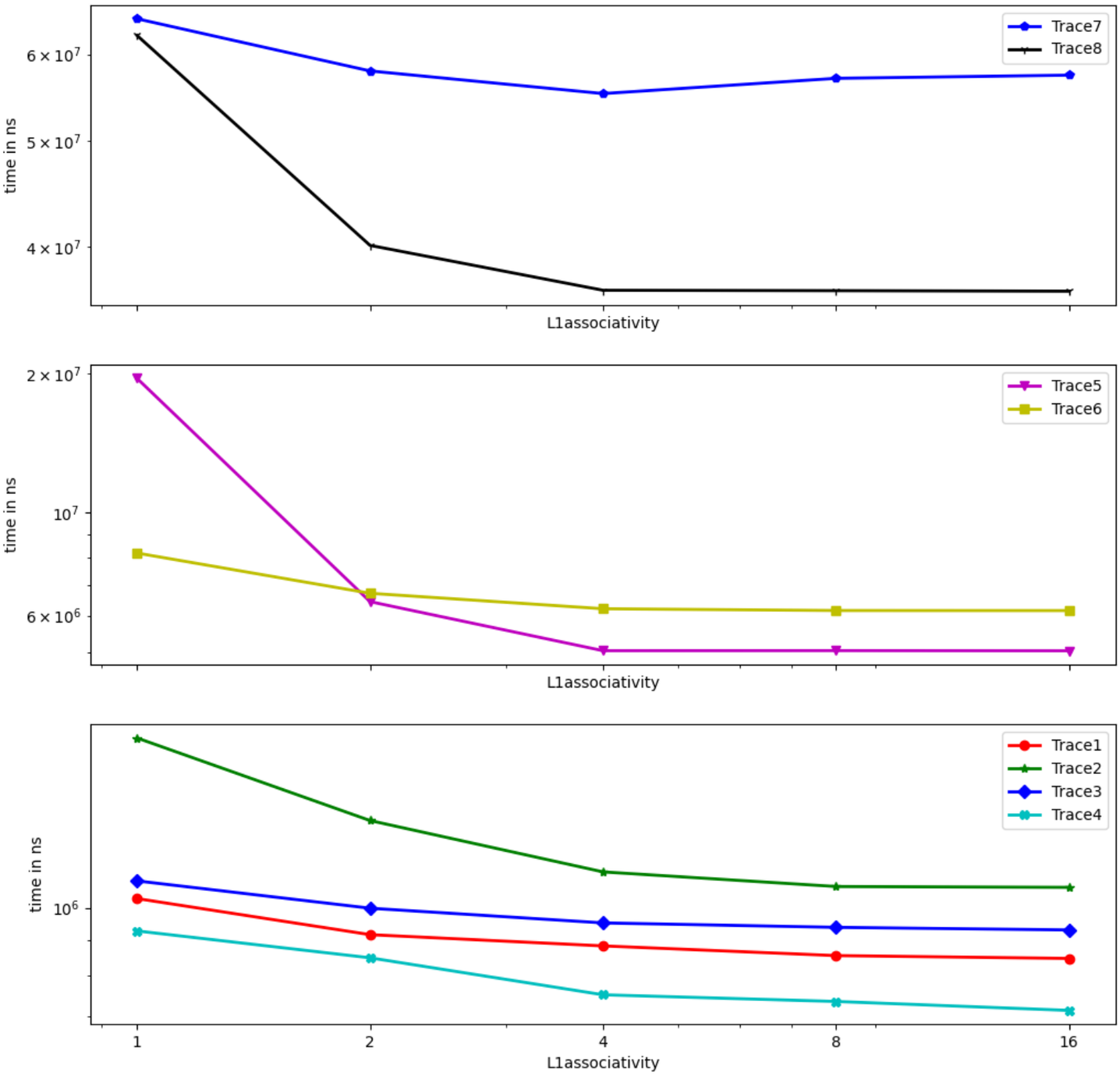

Figure 4: time v/s L1 Associativity

increasing the associativity, we know that the conflict misses will reduce, since there would be more 'ways' in one set. In practice this should also result in increased access time of the cache as there would be more tags to check for each set index, but in our analysis here we have assumed the cache access time to be constant at 1 ns.

1. for traces 5,8 :- there is an especially huge decrease in time on increasing L1 associativity. This implies

that in these traces, conflict misses are more frequent than others.

2. for trace 7:- the graph first slopes downwards but then slightly increases after a certain point. this is due to the fact that the number of sets are reduced on increasing the set associativity.

3. for traces 1,2,3,4,6 :- the standard trend shows a downward sloping time vs associativity graph because of reduced conflict misses.

4. trace 5's overall time reduces more than trace 6's and hence its time becomes lesser at the highest associativity. This must mean that trace 5 originally had more conflict misses than trace 6.

2.5 L2 associativity

increasing the associativity, we know that the conflict misses will reduce, since there would be more 'ways' in one set. In practice this should also result in increased access time of the cache as there would be more tags to check for each set index, but in our analysis here we have assumed the cache access time to be constant at 20 ns.

1. traces 7,8,5,6 :- the downward slope is slight, hence in these traces the reliance on L2 cache and L2's conflict misses were not a major problem, hence increasing the associativity results in slight change.

2. traces 1, 3, 5 :- the graph is decreasing and then nearly constant with L2 associativity, meaning conflict misses are reduced and then we have an optimal associativity value beyond which not much change is observed.

3. trace 2 :- as can be seen in the graph, the time first decreases then increases. The reason here is that on increasing associativity of the L2 cache (while keeping access time constant), we are also simultaneously reducing the number of sets that data can be mapped to. In some cases, this would not be efficient.

   for a simple example :- consider that there are 2 sets originally and we keep set 2's blocks remain constant(while all being used) while set 1's blocks keep changing (eviction then replacement). When we double the associativity in this case and turn it into having just 1 set, this means that the blocks that were originally in set 2 would also get replaced now and we will need to fetch them later again, resulting in increased access time.

3 All full sized graphs

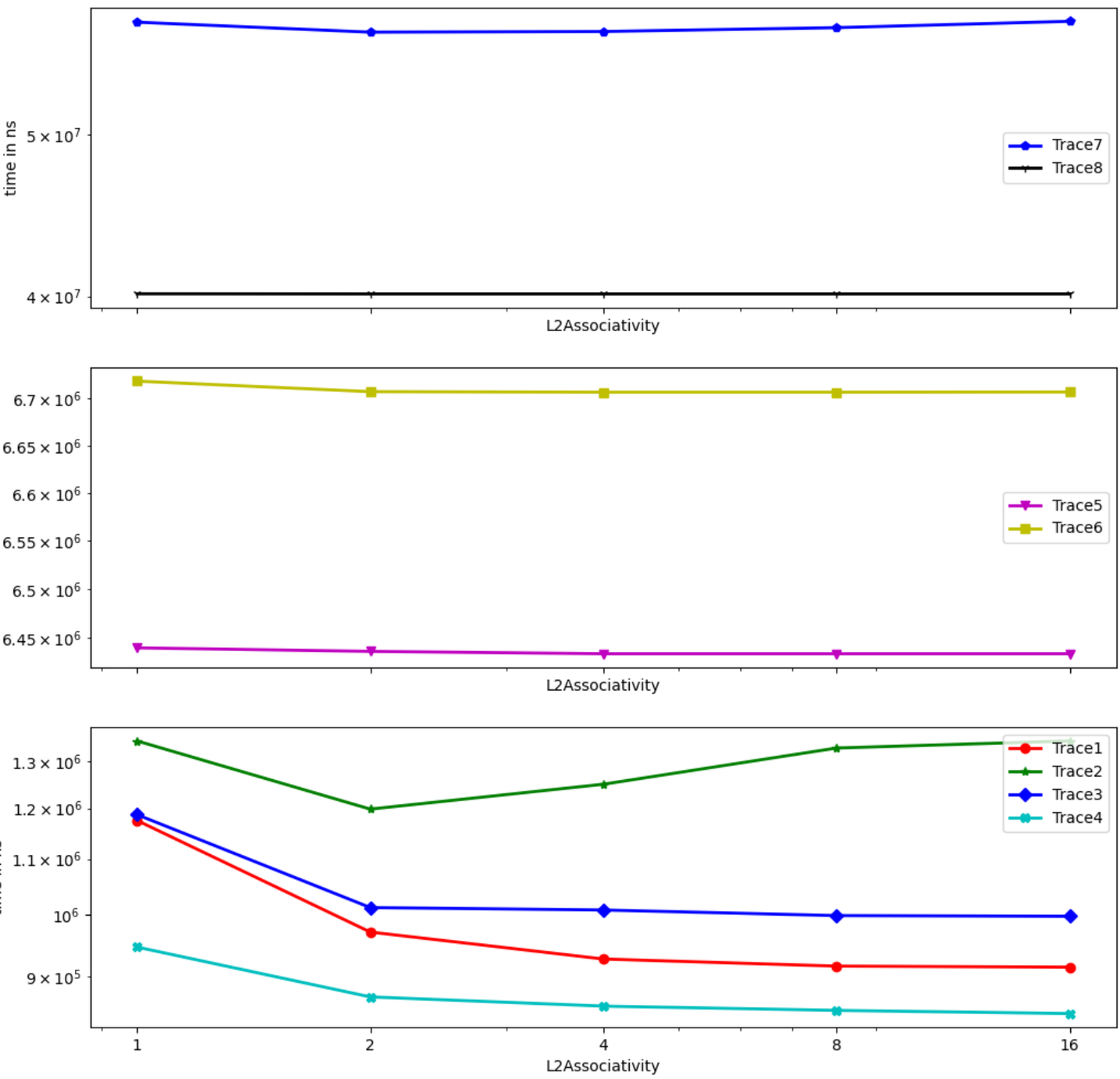

Figure 5: time v/s L2 Associativity

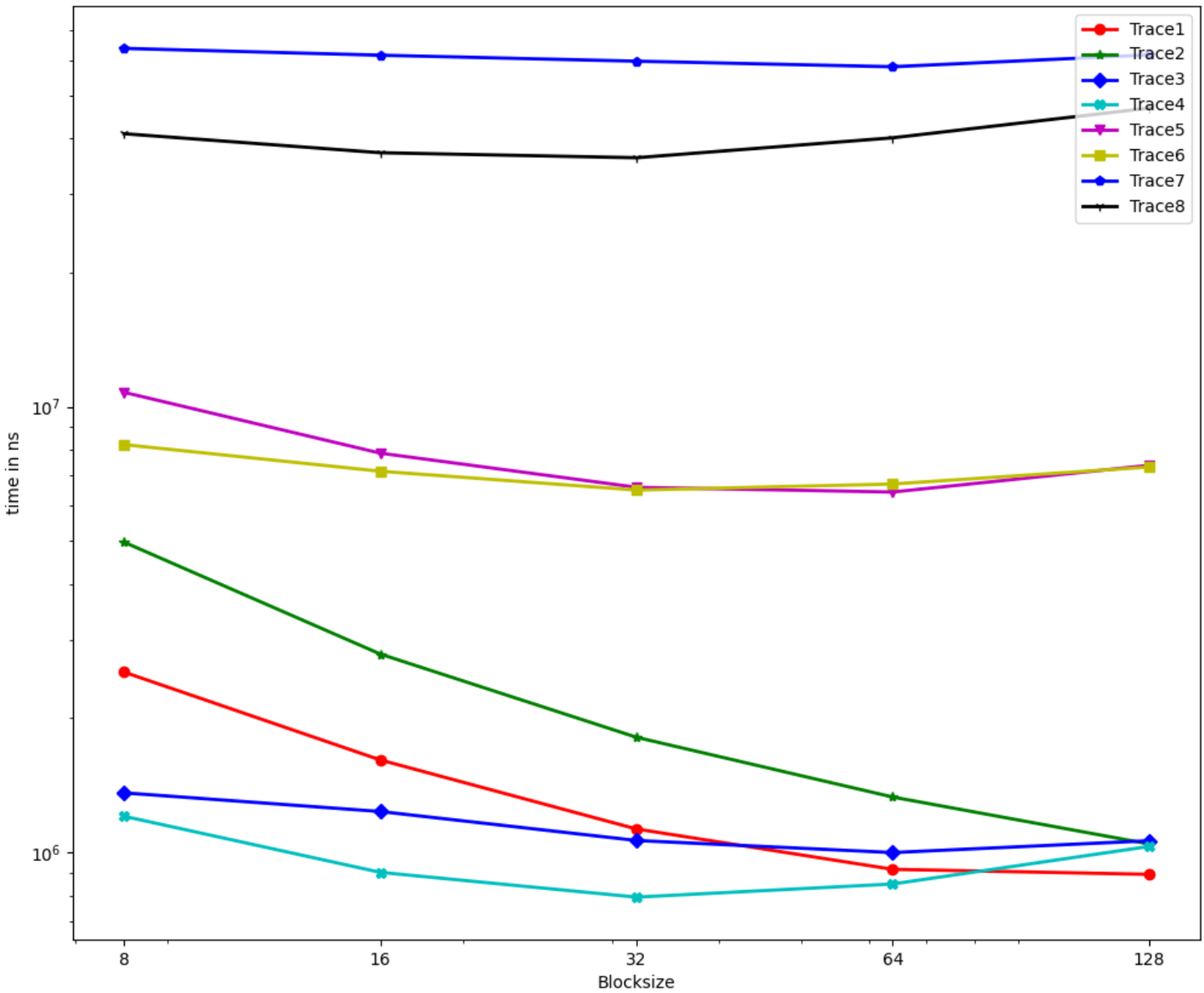

Figure 6: time v/s Block size

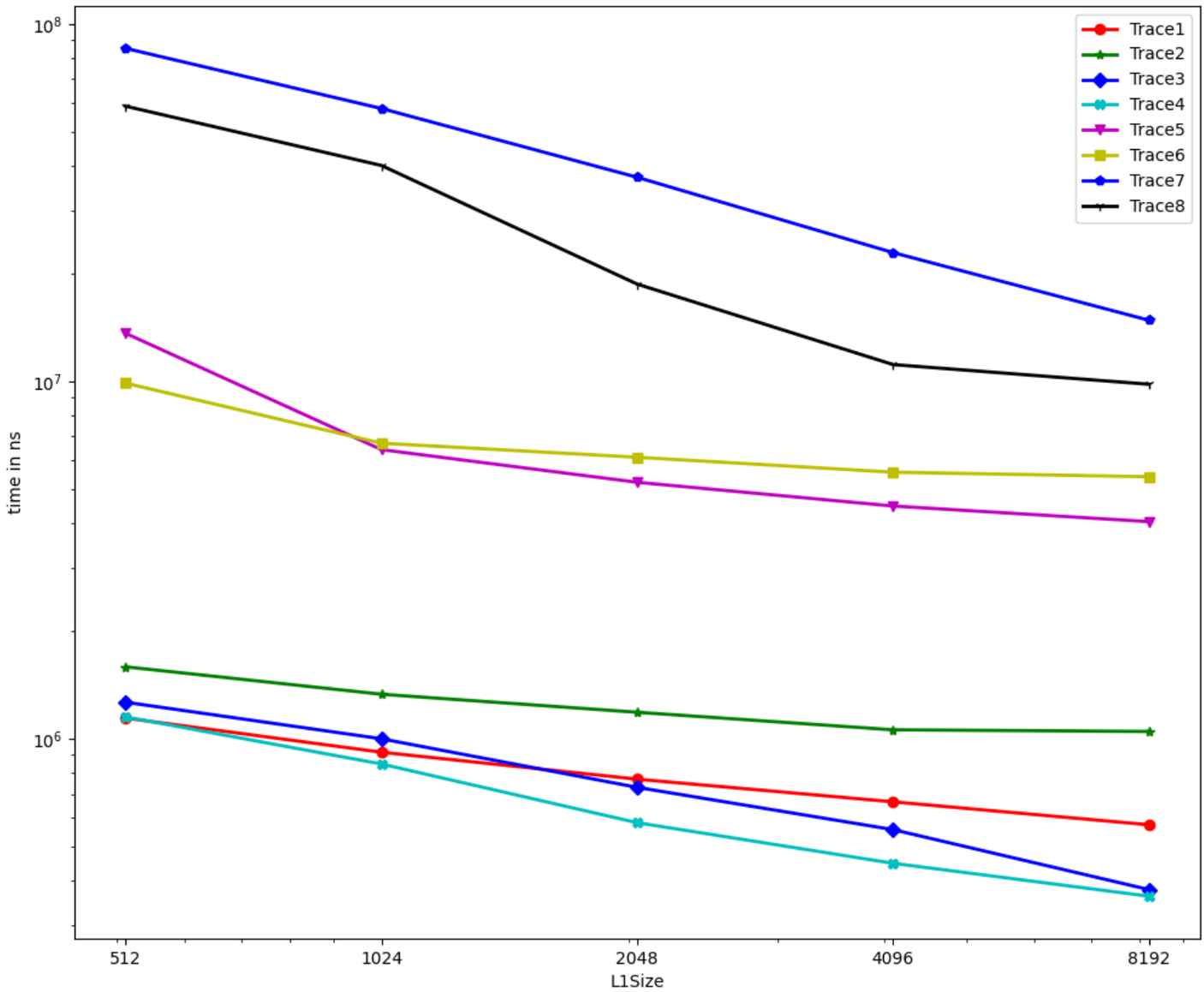

Figure 7: time v/s L1 size

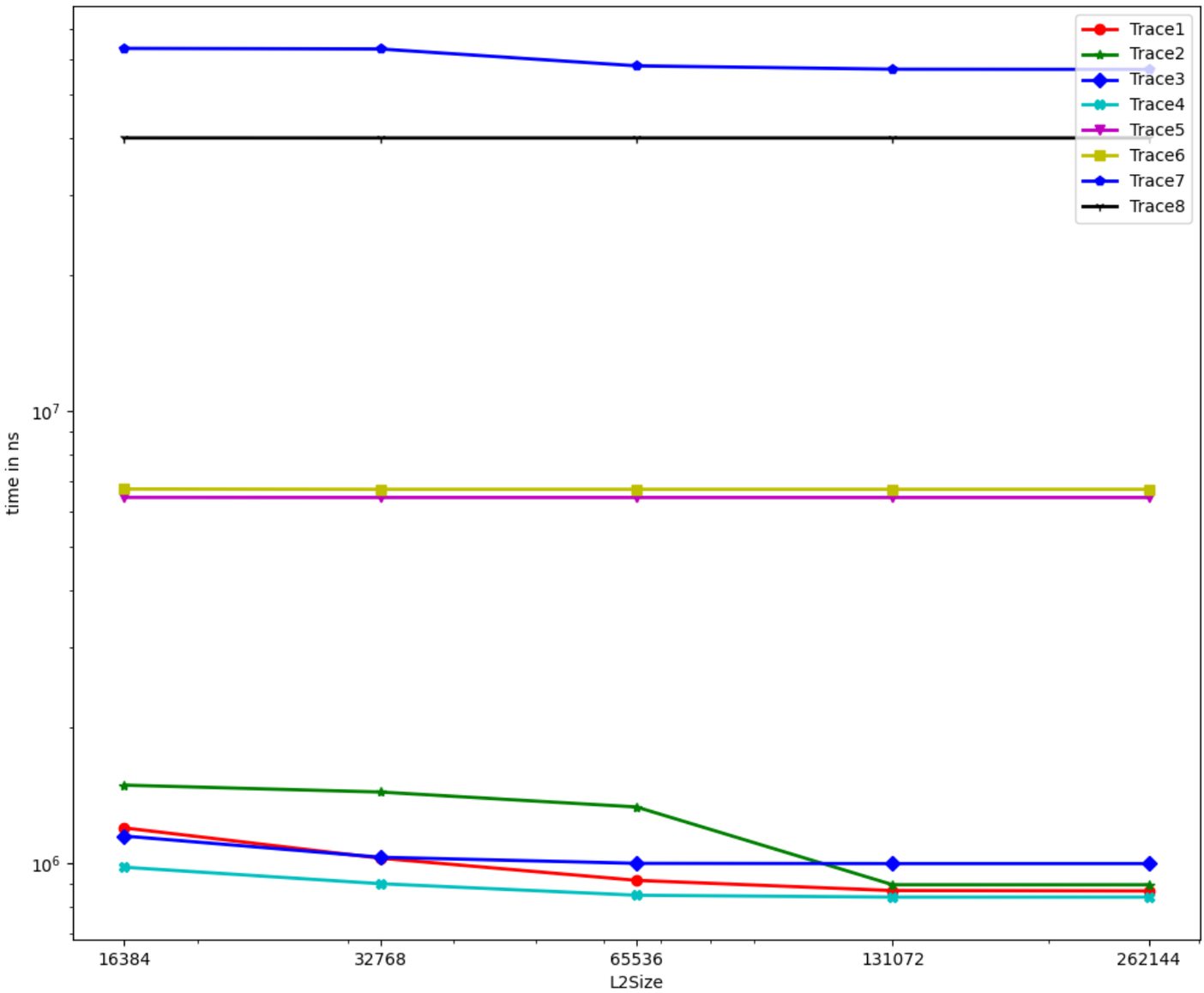

Figure 8: time v/s L2 size

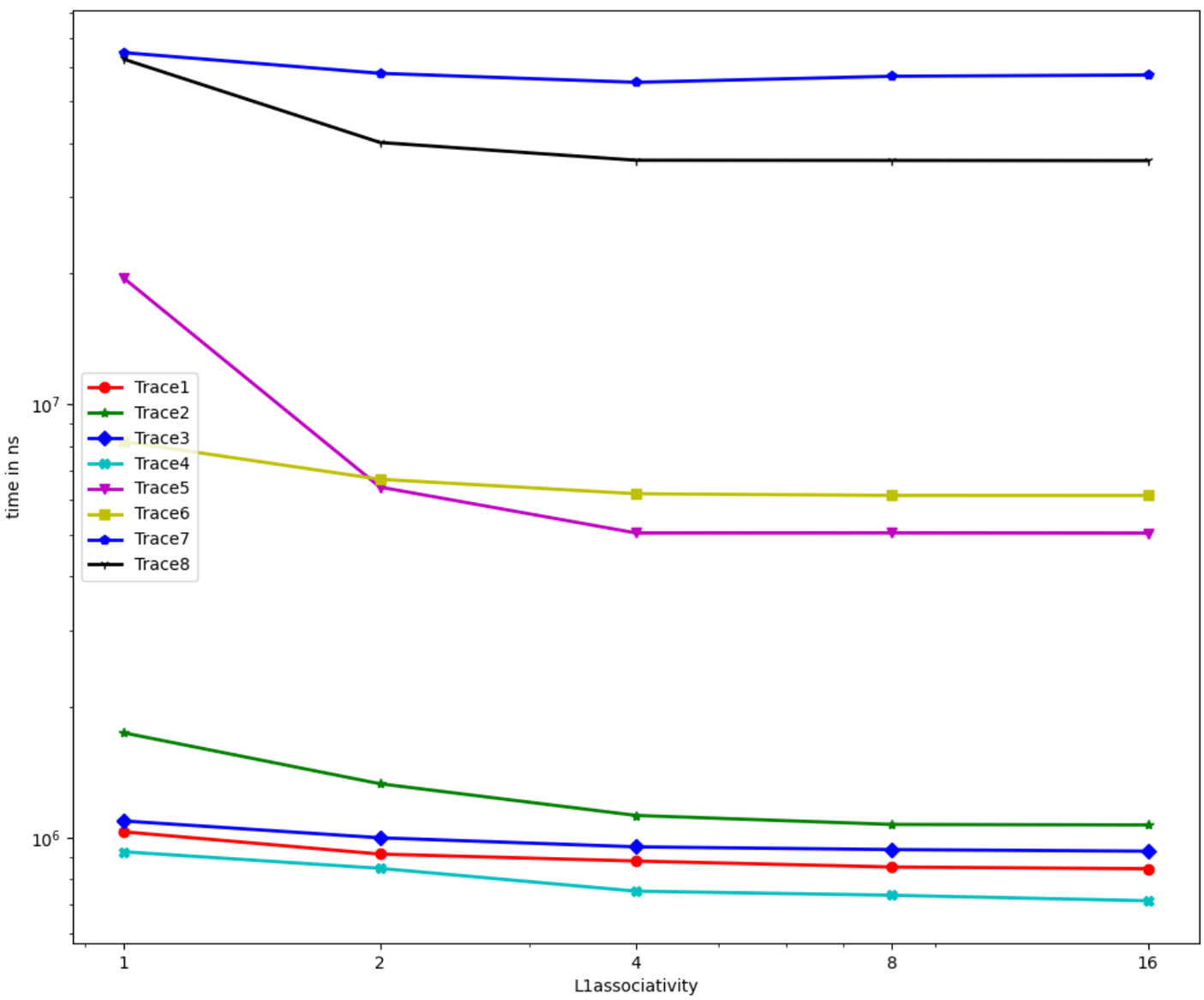

Figure 9: time v/s L1 associativity

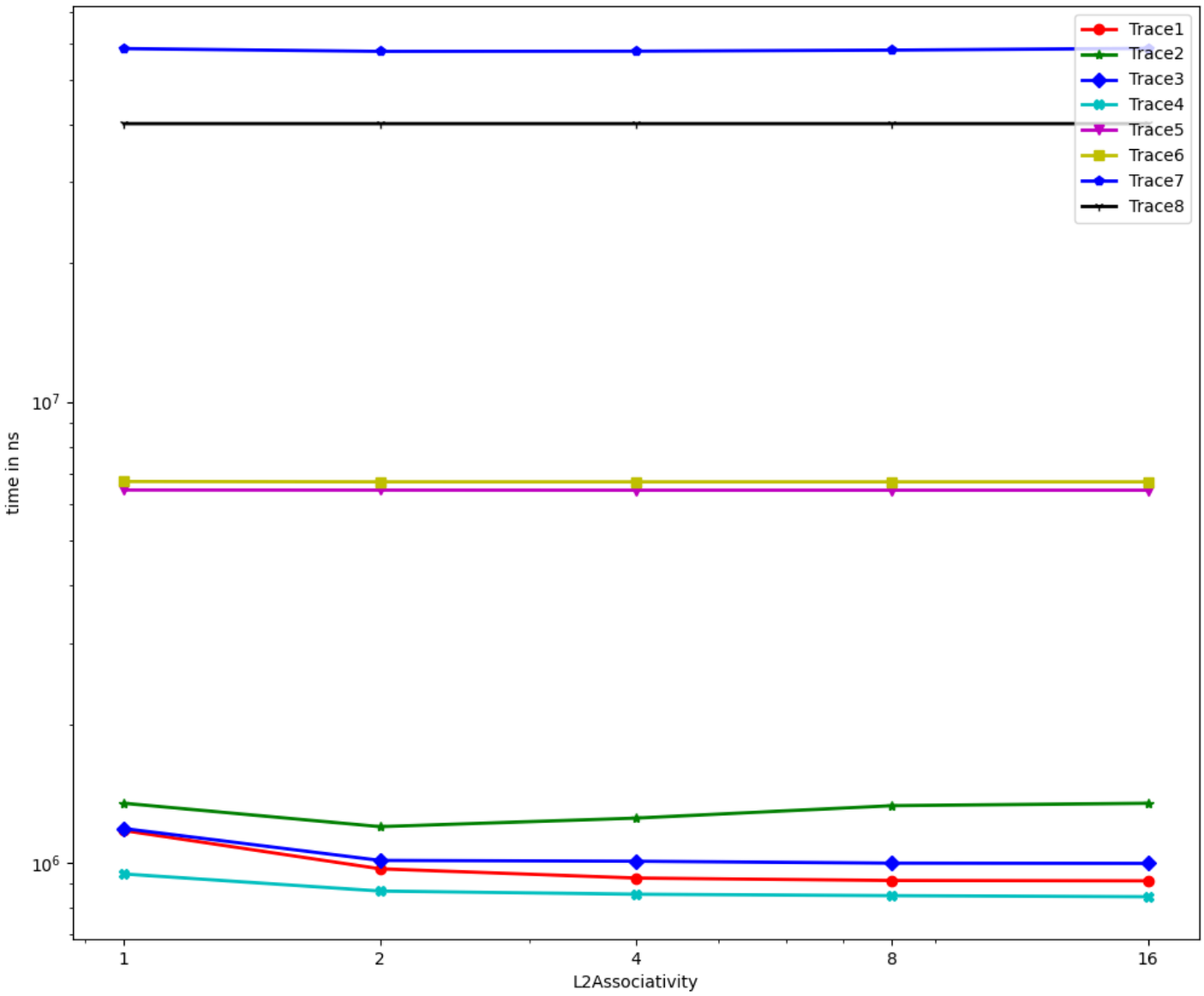

Figure 10: time v/s L2 associativity