Cache Simulation in C++

by Rajat Golechha and Harshit Gupta

May 11, 2023

§1 About the Code and how it works :

§1.1 Read case :

§1.1.1 Read Hit in L1 :

In this case only the LRU needs to be updated.

§1.1.2 Read Miss in L1, hit in L2 :

In this case the LRU is updated, value is taken from L2 and stored in L1, if something is evicted from L1, then it is written back using WBWA policy if it is dirty.

§1.1.3 Double Miss :

In this case the value is loaded from memory to both L2 and L1 and LRU is updaated.

§1.2 Write case :

§1.2.1 Write hit in L1 :

In this case the dirty bit for value is changed and LRU is updated.

§1.2.2 Write miss in L1, hit in L2 :

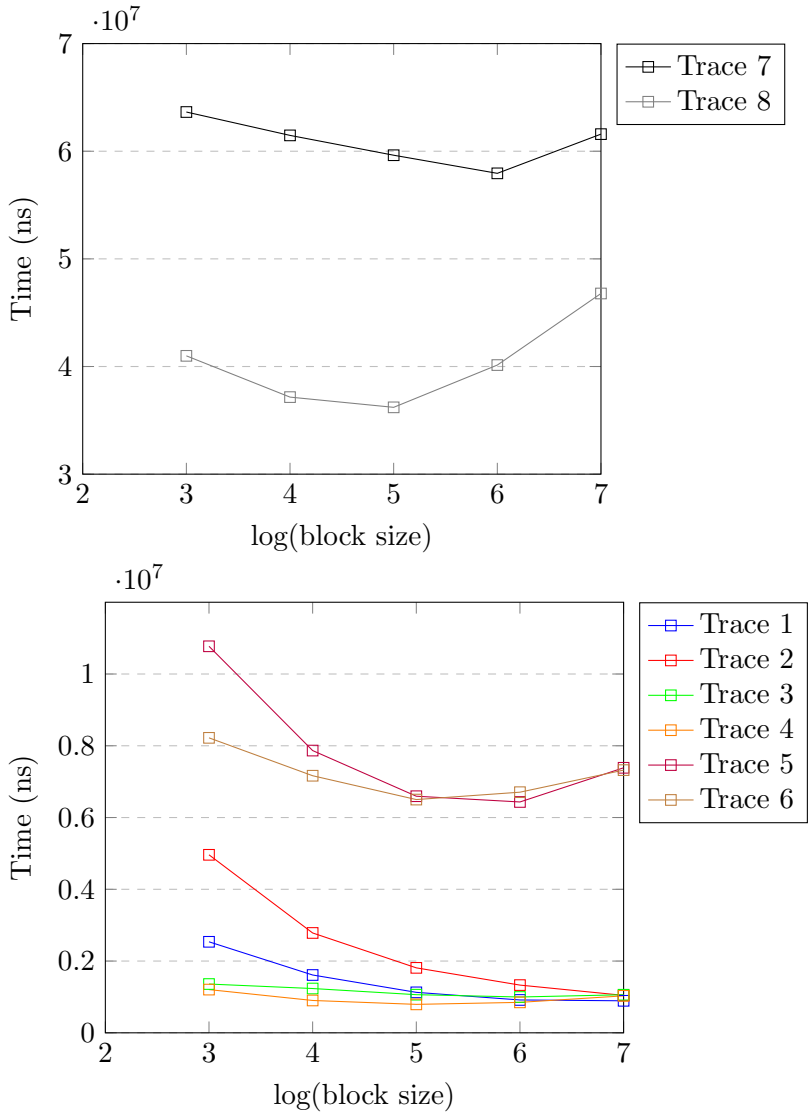
In this case the value is loaded from L2 to L1 and dirty bit set to 1, if something is evicted from L1, then it is written back using WBWA policy if it is dirty .

§1.2.3 Double Miss :

In this case the LRU is updated, value is taken from memory to both L2 and L1 and LRU is updaated and dirty bit set to 1.

§2 Observations on varying parameters

§2.1 Changing of the block size :

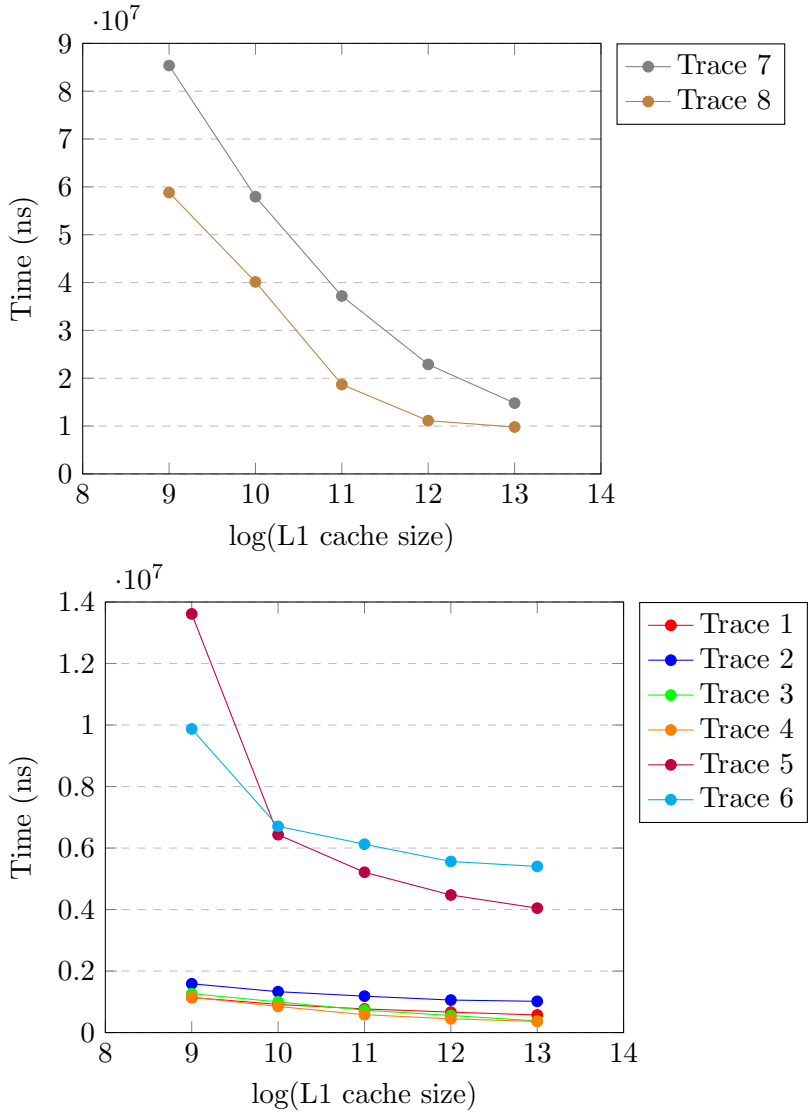


§2.1.1 Explanation of graphs :

The boat-shaped graph typically represents the relationship between cache block size and performance. When we increase the block size, we can take advantage of spatial locality by bringing more data into the cache at once. This can lead to a performance improvement as the processor has to spend less time waiting for data to be fetched from the main memory.
However, as we continue to increase the block size, we eventually reach a point where the number of blocks that can be stored in the cache starts to decrease. This is because the cache has a fixed size and larger block sizes mean that fewer blocks can fit in the cache. This reduction in the number of blocks can lead to an increase in cache conflicts or collisions. Cache conflicts occur when two or more blocks compete for the same cache set, leading to cache misses and reduced performance.
Thus, beyond a certain optimal block size, the performance starts to degrade as the cache becomes less effective at reducing the number of cache misses. The boat-shaped

graph shows the optimal block size for a given cache size and working set size, where the performance first increases and then decreases as the block size increases.
For really small files or programs with small working sets, the cache can have ample space to hold all the necessary data, and increasing the block size can continue to improve performance. However, for larger programs or files with larger working sets, the boat-shaped curve will become more apparent as the performance improvement from increasing the block size reaches a peak and then starts to decline

§2.2 Changing L1 Cache Size

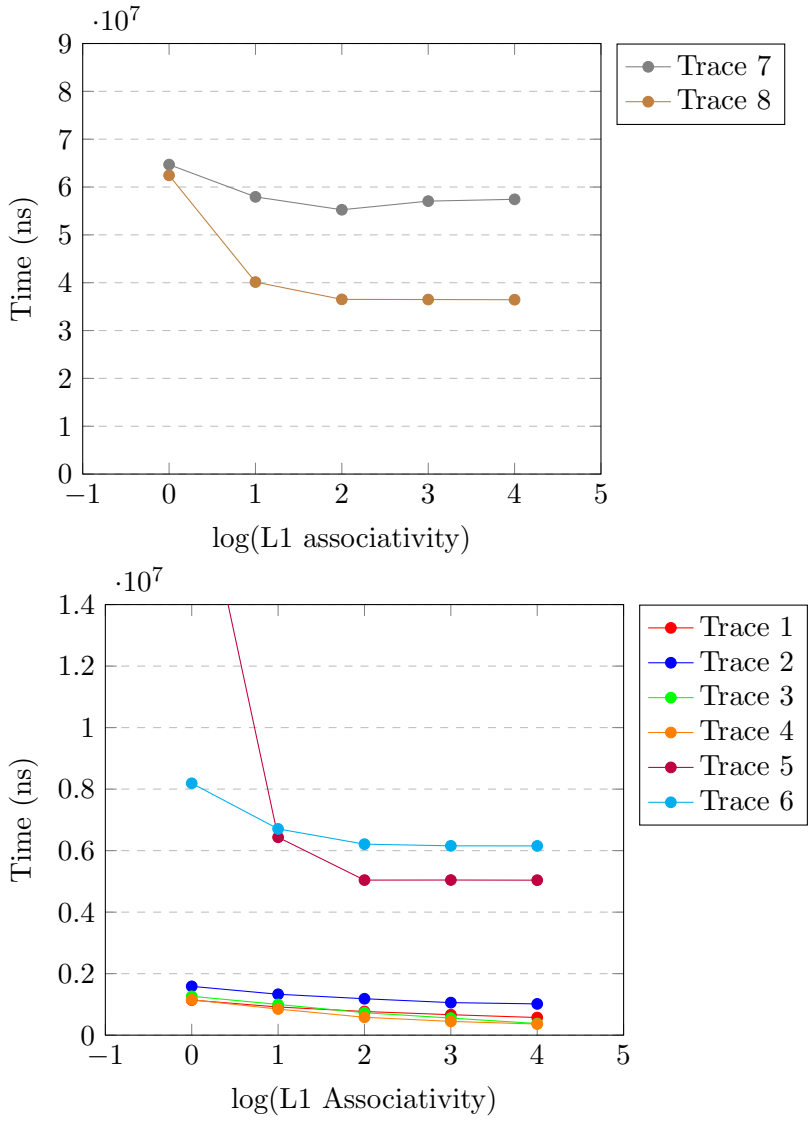


§2.2.1 Explanation of graphs :

To elaborate further, the impact of cache size on performance is dependent on the working set size, which is the set of data that is frequently accessed by the processor. If the working set size is larger than the cache size, then there will be more misses and performance will suffer. Increasing the cache size reduces the number of misses by storing more blocks in the cache, but the improvement in performance diminishes as the cache size increases beyond a certain point.
Additionally, the impact of cache size on performance is highly dependent on the nature

of the program being executed. Programs with small instruction sets may not benefit significantly from an increase in cache size, as they do not require a large amount of data. In contrast, programs with larger instruction sets may benefit more from larger caches as they require more data to be stored and accessed frequently.
In summary, the impact of cache size on performance is complex and depends on a variety of factors, including the size of the working set and the nature of the program being executed. Increasing cache size can improve performance, but only up to a certain point, beyond which the improvement begins to plateau and asymptote for a given input.

§2.3 Changing L1 associativity :

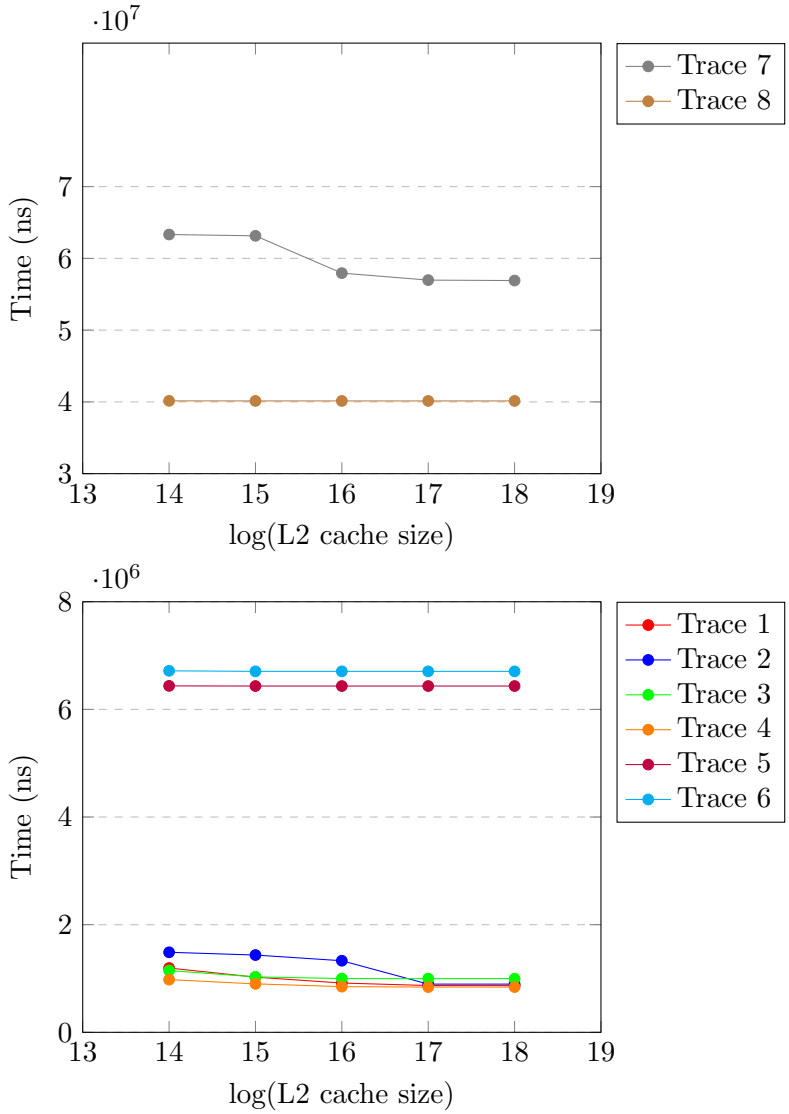


§2.3.1 Explanation of graphs :

Caches with higher associativity can hold more blocks per set, which means that conflicts are less likely to occur and the number of **capacity misses** is reduced. Therefore, increasing associativity from a directly mapped cache to a 2-way associative cache can have a significant impact on performance because it doubles the number of blocks in a set and reduces the likelihood of conflicts. However, increasing associativity from a 2-way to a 4-way cache has less of an impact on performance because there are lesser conflicts

now,.
As associativity continues to increase, the impact on performance diminishes because the number of blocks per set continues to increase, but at a diminishing rate. For example, increasing associativity from a 4-way to an 8-way cache will not have as much impact on performance as increasing from a directly mapped cache to a 2-way associative cache. This is because the higher associativity cache already has fewer conflicts to begin with, so the impact of further reducing conflicts by increasing associativity is less significant. Overall, the impact of increasing associativity on cache performance depends on the degree of conflicts in the cache. The higher the degree of conflicts, the more significant the impact of increasing associativity will be. However, as conflicts are reduced through increases in associativity or other means, the impact of further increases in associativity on performance will be less significant.

§2.4 Changing L2 cache size



§2.4.1 Explanation of graph :

When a processor needs to access data that is not currently in the L1 cache, it will look in the L2 cache before going to main memory. Therefore, increasing the size of the L2 cache

can help reduce the number of accesses to main memory, which can improve performance. However, the impact of increasing the L2 cache size is typically less significant compared to increasing the size of the L1 cache for several reasons.

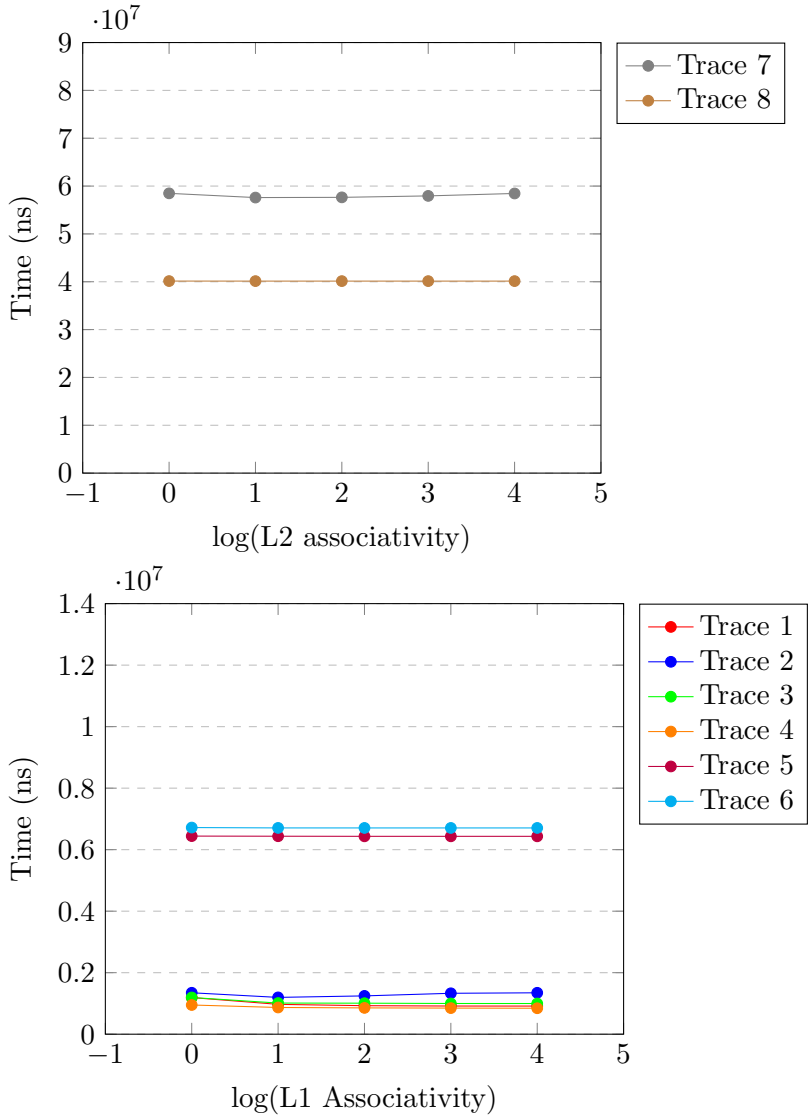
Firstly, smaller instruction sets with fewer instructions may not require much data to be stored in the cache, so increasing the cache size may not provide significant performance improvement. In these cases, the performance gain from increasing the L2 cache size can be negligible.

Secondly, as mentioned earlier, the L2 cache is typically farther away from the processor than the L1 cache. This means that the latency of accessing the L2 cache is higher than the latency of accessing the L1 cache. Therefore, the impact of increasing the L2 cache size is limited by this latency, and increasing the cache size beyond a certain point will have diminishing returns in terms of performance improvement.

Finally, in cases where both the L1 and L2 caches are accessed and a miss occurs in both caches, this is referred to as a "double miss." Double misses are relatively rare and may occur in less than 1% of memory accesses in some workloads. Therefore, increasing the size of the L2 cache will only provide a performance benefit in cases where double misses occur, which are typically infrequent.

In conclusion, increasing the size of the L2 cache can provide some improvement in performance, but the impact is typically less significant compared to increasing the size or associativity of the L1 cache. The effectiveness of the L2 cache also depends on the working set size, the specific hardware architecture, and the frequency of double misses.

§2.5 Changing L2 associativity



§2.5.1 Explanation of the graphs :

Increasing associativity in L2 cache has less significance compared to increasing the size of L2 cache or increasing associativity in L1 cache. This is because L2 cache is farther away from the processor than L1 cache, and is accessed less frequently. Additionally, L2 cache is typically larger than L1 cache, so it already has a lower miss rate. Increasing associativity in L2 cache would only provide a marginal improvement in **conflict miss case** which are very less as L2 is very large, thereby reducing the miss rate. Hence, in most cases, increasing the size of L2 cache provides a better performance improvement than increasing its associativity which gives little to no improvement in performance.

§3 Conclusion :

The work split in this assignment has been:
Rajat Golechha : 2021CS10082 : 50%
Harshit Gupta : 2021CS10552 : 50%