Lab Exercise 1 – Concurrency

Objective

Analyzing the consequences of not properly synchronizing the updates of shared data, as well as the costs associated with synchronizing.

Implementation

Multiple threads are created in an array which access a shared counter, initialized s 0, in a loop. In each iteration, each thread reads the shared counter to a local (stack) variable, increments it, and stores it back to the shared counter.

Program A – TaskA

This program is an unsynchronized implementation of different threads sharing the same data.

Program B – TaskB

This Program is a synchronized implementation of different threads sharing the same data.

Both the program is executed 5 times with different number of threads (2, 4, 8, 16, 32) and in both of the programs the body of the main function is wrapped in a loop which runs the main program 20 times so that we get 20 results for the analysis. So, with different 5 number of threads and 20 results for each execution we get a total of 100 results.

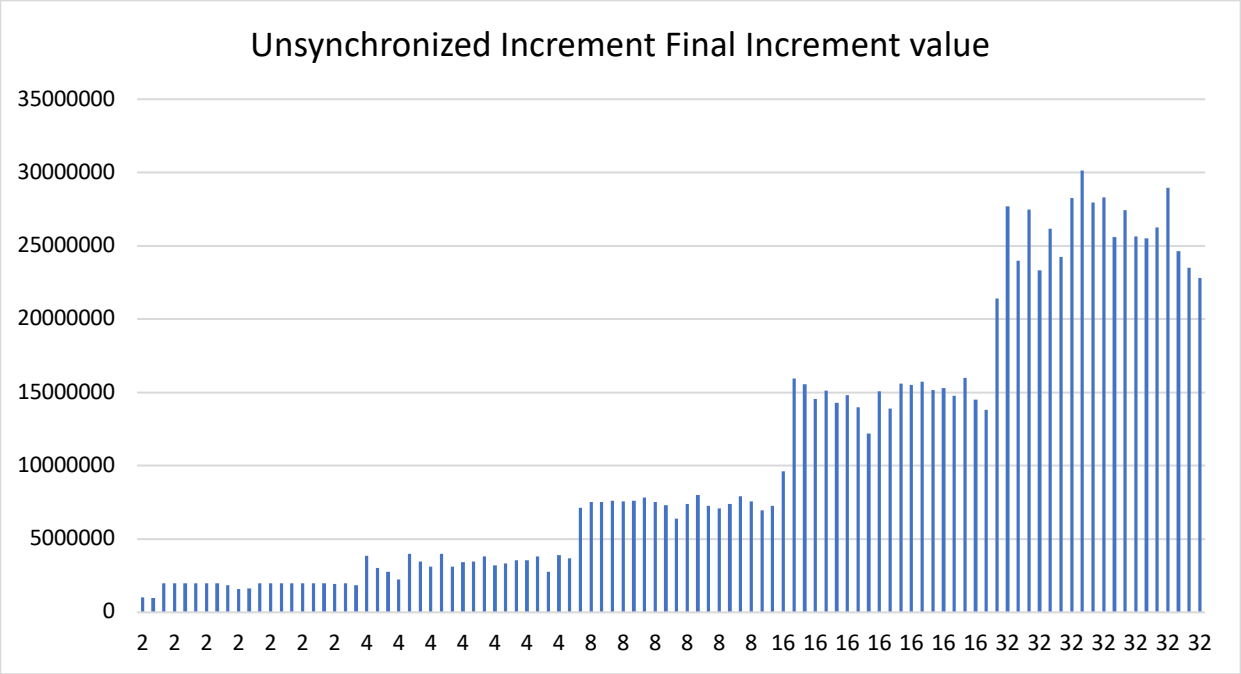
Output

A data of the output is collected in a CSV file which is then used for the analysis.
The following data is collected in the CSV file as the output of the program:
NumberOfThreads: The number of threads created for the execution of the program.
FinalCounter: This is the final value of the counter. (the number of increments made to the counter)
TotalTime: The total time taken for the execution of the program. (for each loop of the main body)
AvgCountPerThread: This is the average successful increment done by each thread.
AvgTimePer1000000: This is the average execution time taken to increment up to 100000 in each execution.

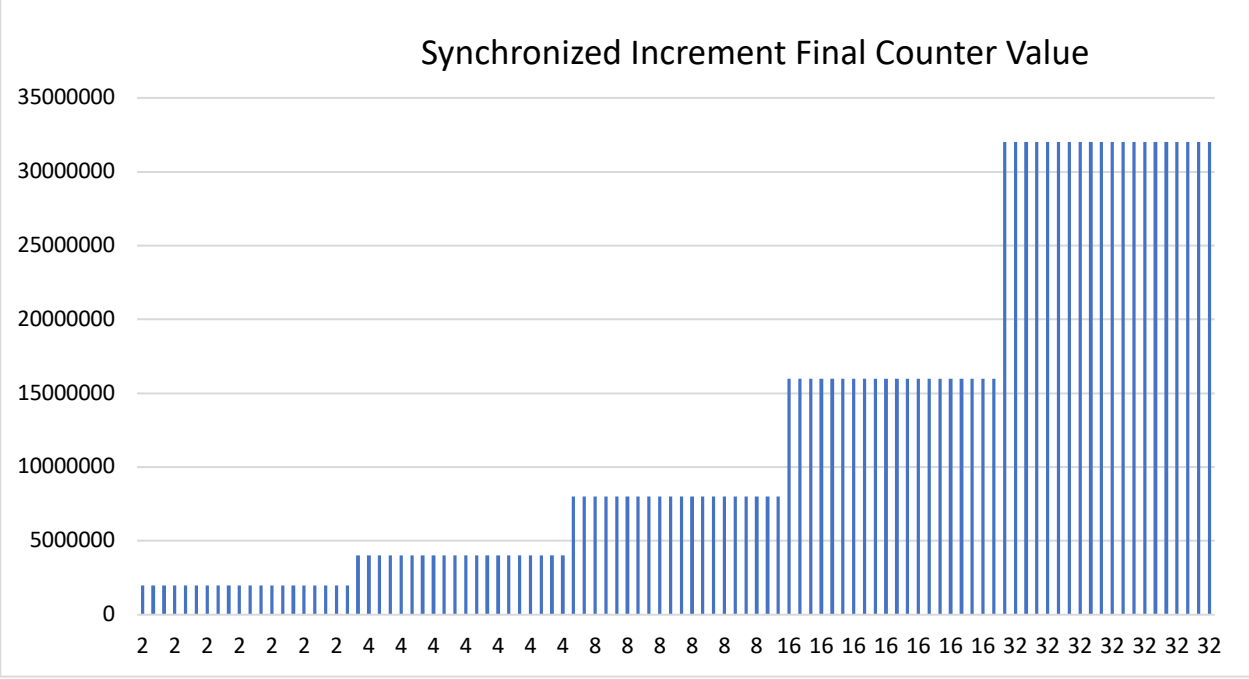
Analysis

The final counter value in the unsynchronized increment was mostly less than n*1000000. This was because of multiple threads trying to manipulate the same data exactly at the same time.

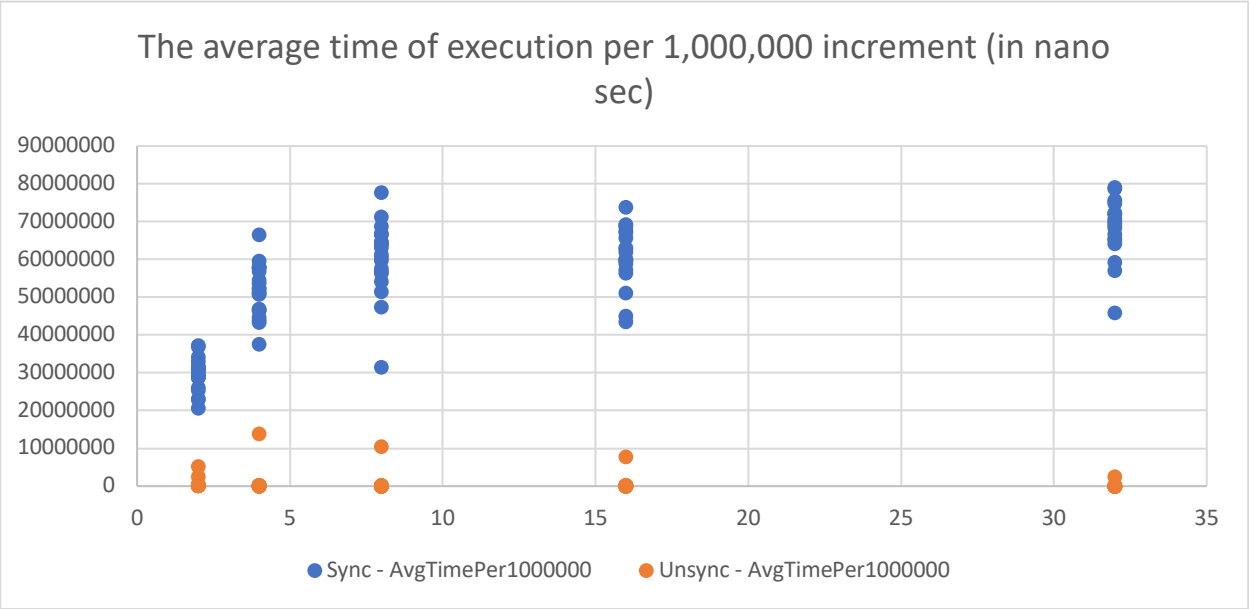
There are a few times where the counter value is n*1000000, this is because the program completed with just one thread, before another thread was even started. The following chart shows the final counter value of each execution. The chart clearly shows the final counter value is mostly lower than n*1000000 and is very random.



The following chart shows the final values for the synchronized increment. The chart shows that the values are accurate. Unlike unsynchronized increments, the threads wait until the other thread completes its operation and does not try to manipulate the same data at the exact same time. So, synchronized operation is beneficial for the case of accuracy.

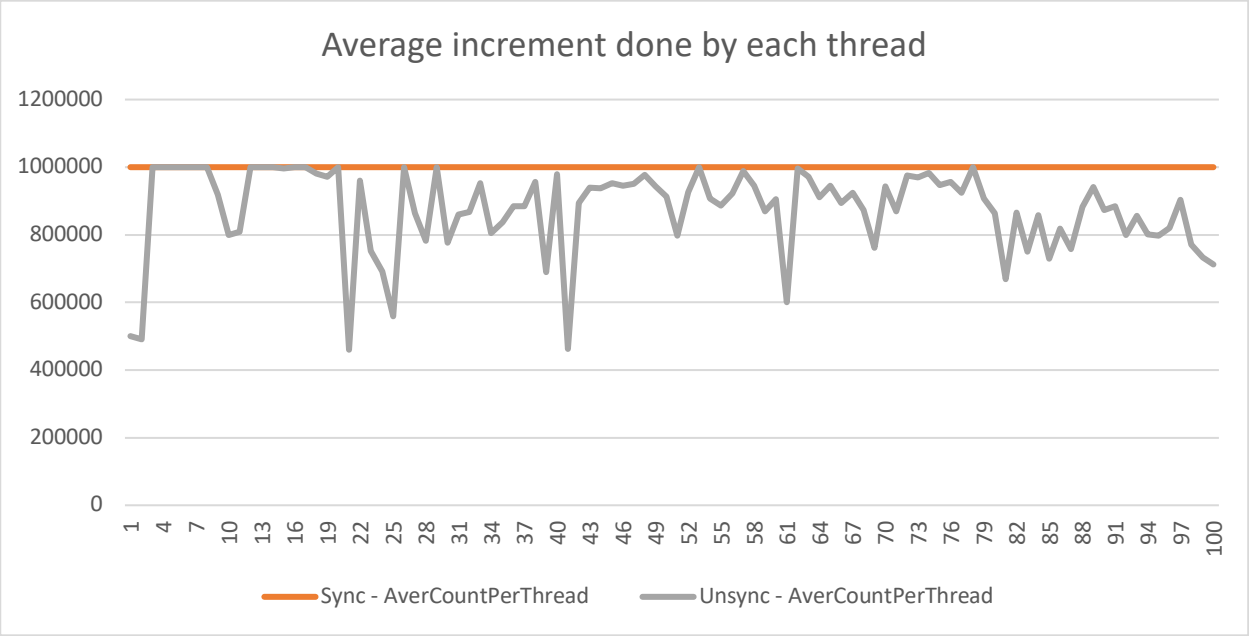


The following chart shows the average execution time per 1,00,000 increment of the counter with different threads.



The chart clearly shows that the average execution time of the unsynchronized threads are quite lower than the synchronized thread execution time. Also, the execution time gradually is increasing as the number of threads is increasing.

The following chart shows the comparison between synchronized and unsynchronized average successful increment done by each thread.



The above chart shows that each thread in the synchronized increment is equally incrementing the process while in the unsynchronized increment the threads have a random number of

increments. This is happening because the threads are trying to execute an increment operation to the same data at the exact same time. But in the synchronized increment, the threads have to wait until the other thread completes its operation. This is also the reason why the execution time in the synchronized increment is taking longer than the unsynchronized one.

Conclusion

The consequences of not properly synchronizing the updates of shared data would be that we would not get and accurate result. The costs associated with synchronizing would be the higher execution time. So, if we are looking for accuracy and can compromise with the time cost, then certainly synchronizing would be the safe deal.