LAB 2 :

Making parallelism with a selected number of threads, specifying thread data working-set statically, exploring different way to coordinate threads results: purely shared data, local private data, global private data. Evaluating the effect of number of threads on performance. Exploring the problem of false sharing.

We aim to develop a parallel application to count the occurrences of the number 3 in a large array of integers. The counting task will be divided among the provided threads, with each thread handling an evenly sized sub-array (determined by the thread rank).
The size of the array is defined as a variable N (if you need to exceed the capacity of the largest int, use the long long C type). The number of threads is denoted as NT (you can assign it the number of cores on your computer)

A- We now aim to investigate the performance of different approaches for coordinating counting. For each of the following cases, write a distinct C function:

Case 1: Write a sequential version of the occurrences counting.

Case 2: Write a parallel version using a globally shared counter (ensure it is protected).

Case 3: Write a parallel version using a local counter per thread and then combine the results

Case 4: Write a parallel version using a global array of counters, with one element per thread (each element accessed privately by each thread), and then combine the results.

B – Select the most efficient implementation from the previous question and assess the impact of the number of threads on performance. To facilitate testing flexibility, we recommend passing the number of threads as a command-line argument (using the char **argv array).

Note: for precise milliseconds metering, use the **`timeInMilliseconds()`** function provided in 'timing.c' file on BB. Repeats your tests many times, and increase the dimension N of the array to affect bigger jobs to threads.