ID1217 CONCURRENT PROGRAMMING

ASSIGNMENT -1

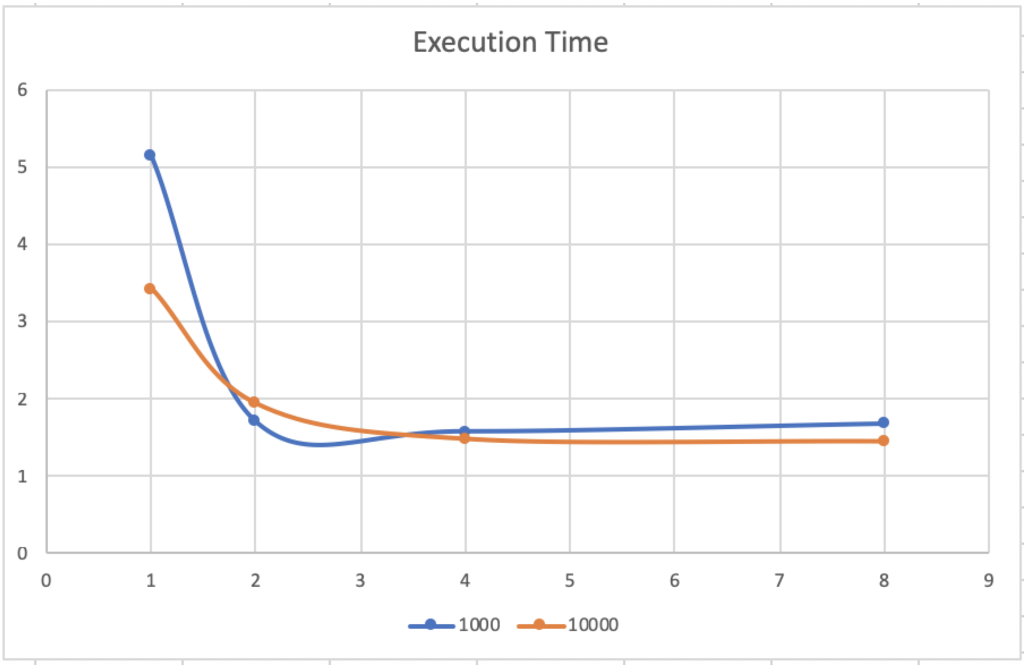
# Critical Sections. Locks, Barriers, and Condition Variables

SUBMITTED BY-
KISHORE BAKTHA

## PROBLEM 1

**a)** In order to find the maximum and minimum elements along with their respective positions, I have defined arrays which stores the corresponding maximum and minimum elements along with their positions of each of the workers. Finally, after the barrier, thread 0 compares the values computed by each of the thread and obtains the maximum and minimum elements.
The problem in this solution is false sharing due to the sharing of arrays.

Size- 10000*10000
Number of Threads-1. Execution time- 0.341824sec
Number of Threads-2. Execution time- 0.193952sec
Number of Threads-4. Execution time- 0.148561 sec
Number of Threads-8. Execution time- 0.145366 sec



b) In order to avoid the use of barriers and temporary arrays, I have defined a global variable called sum which is accessed within a critical section and the value is computed based on the local values computed by each thread. The main thread prints the final result after waiting for the threads to complete.

Size- 10000*10000

Number of Threads-1. Execution time- 0.293851sec
Number of Threads-2. Execution time- 0.152723 sec
Number of Threads-4. Execution time- 0.132536 sec
Number of Threads-8. Execution time-0.133001 sec

c) For implementing the bag of tasks, we have a global variable called row which obtains a task from the bag and implements it. This is accessed within a critical section. Finally, at the last stage, each worker's local value is computed and checked, and the final result is updated. The value of total is computed in the critical section.

Size- 10000*10000

Number of Threads-1. Execution time- 0.294096 sec
Number of Threads-2. Execution time- 0.145292 sec
Number of Threads-4. Execution time- 0.130776 sec
Number of Threads-8. Execution time-0.130707 sec

**PROBLEM 2**

**a)** In order to implement quicksort using recursive parallelism, one recursive function is executed by the calling thread and then another recursive function can be executed by creating one more thread. In this way, new threads are created recursively to compute the result. Finally, at each stage, we wait for the thread to finish executing before the final result is returned. The main thread calls a function which is responsible for creating different threads and returning the output. The arguments are passed with the help of a structure.

Size- 1000
Execution time- 0.02496 sec

Size- 10000
Execution time- 0.22957sec

Size- 100000
Execution time- 2.46272sec

**PROBLEM 6**

In order to find the palindromic words, the file size is initially 25143 words. So therefore, I have split the file such that all the words are stored in a 2-D array of size [8381] [3]. The threads then choose the strip size according to the corresponding ID. Each thread takes a word, and first computes the reverse of the word. If it is, it then checks whether the word is present in the file or not. If it is, the word is then outputted to a results.txt file. The more the number of threads, the shorter the strip size and thus shorter the amount of work to be performed by each thread. The words are stored using the 2-D string matrix. I have also used a set for checking whether the word is present in the file or not. Each thread stores the values it computes in the sum array. Finally, the total number of words is calculated by summing up all the values.

Number of Threads-1
Execution Time-0.037712 sec

Number of Threads-2
Execution Time-0.019602 sec

Number of Threads-4
Execution Time-0.014882 sec

Number of Threads-8
Execution Time-0.015302 sec