Lebanese American University

School of Arts and Sciences

Department of Computer Science & Mathematics

CSC447: Parallel Programming for Multicore and Cluster Systems

Lab #5: Parallel Sorting

Karim El Jammal - 201903088

April 1, 2022

Note: The machine used has an Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz 3.10 GHz processor and 32.0 GB of installed RAM

1 Introduction

The Quicksort algorithm works by repeatedly dividing unsorted sub-arrays into two pieces: one piece containing the smaller elements and the other piece containing the larger elements. The splitter element, used to subdivide the unsorted sub-array, ends up in its sorted location. By repeating this process on smaller and smaller sub-arrays, the entire array gets sorted. The typical implementation of Quicksort uses recursion. The attached implementation replaces recursion with iteration by using a stack. It manages its own stack of unsorted sub-arrays. When the stack of unsorted sub-arrays is empty, the array is sorted.

2 Time Plot

2.1 Serial Quicksort Algorithm

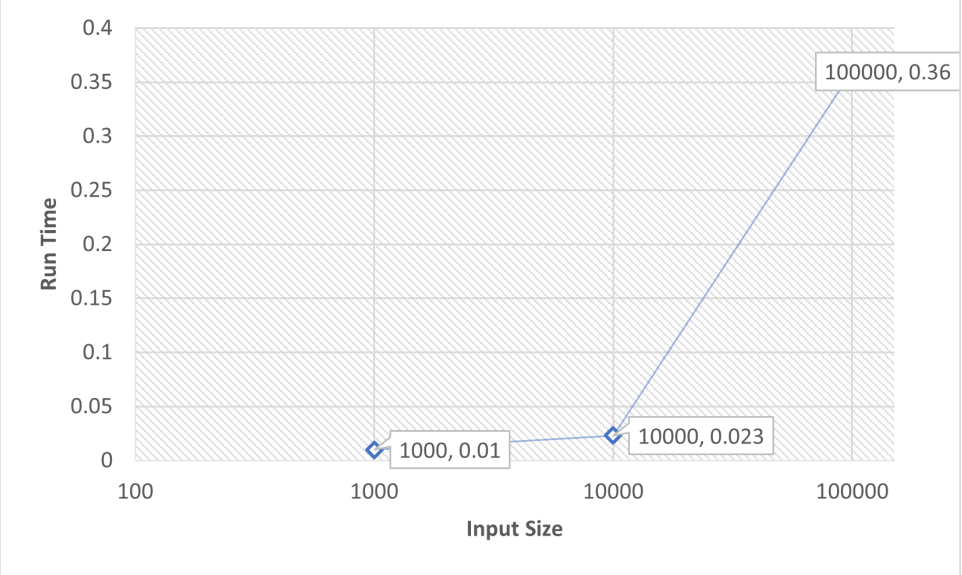

Figure 1: Figure showing the run time for different input size

2.2 Parallelized Quicksort Algorithm with OpenMP

Number Threads: 1

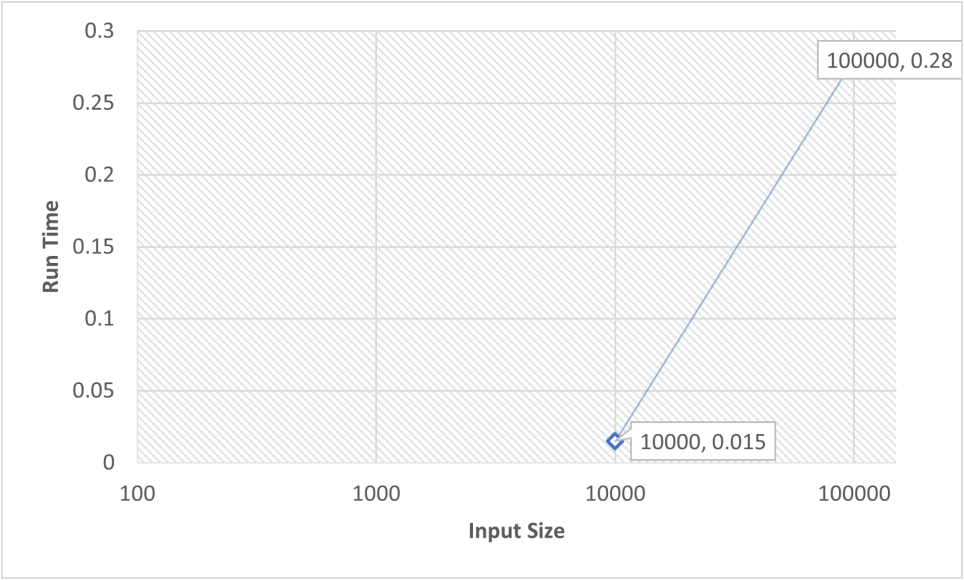

Figure 2: Figure showing the run time for different input size

Number Threads: 2

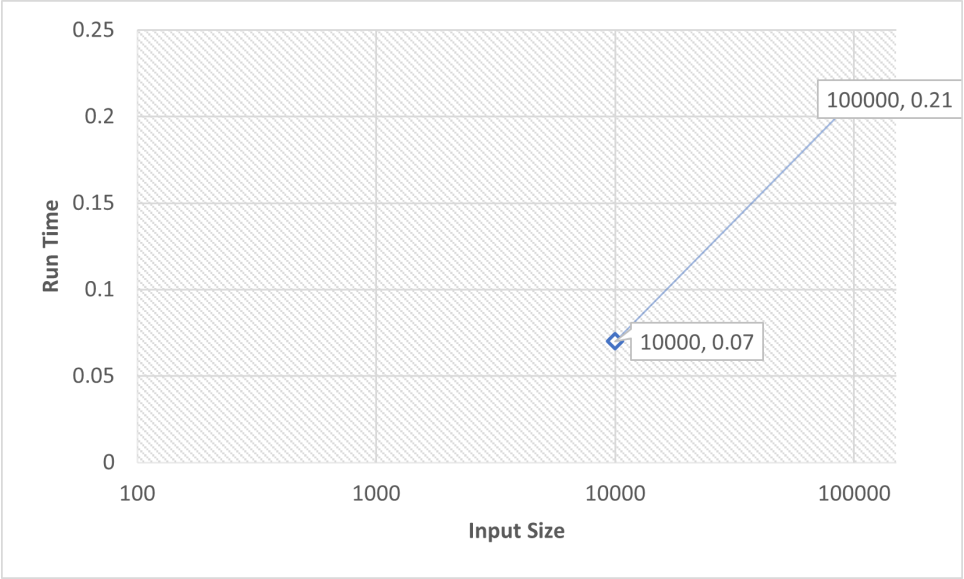

Figure 3: Figure showing the run time for different input size

Number Threads: 8

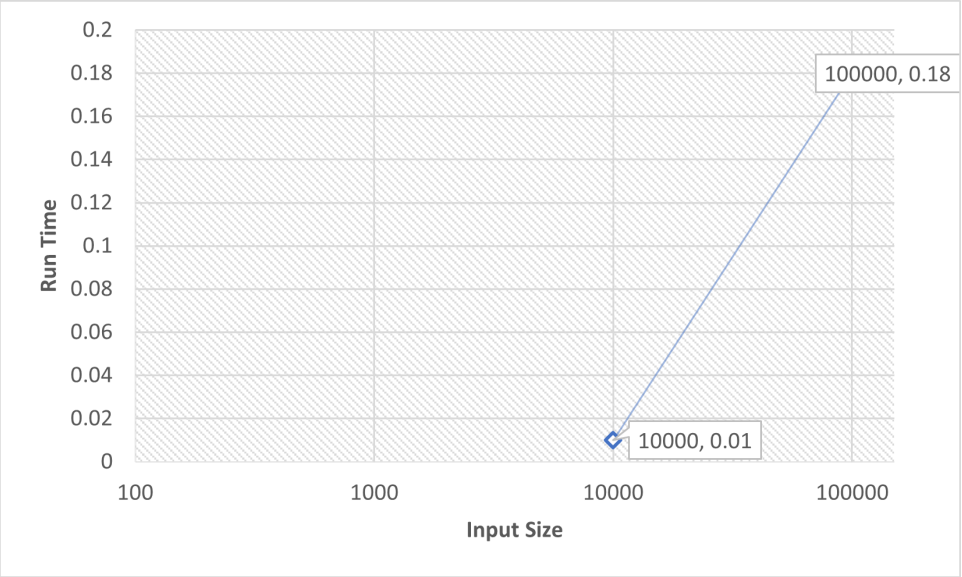

Figure 4: Figure showing the run time for different input size

3 Conclusion

We realize that the serial version of quicksort is not scalable. We can clearly see how the runtime increases when we increase our input. However, when it comes to the OpenMP version, we can notice how the runtime decreases as we increase our input size. It even gets better when we increase the number of threads we are using. For instance, let's take the 100000 sample size, for 1 thread in OpenMP it took 0.28s, the runtime decreased to 0.18s with 8 threads. This is where our speedup occurs.