Assignment-1, COL380
Satyam Kumar Modi, 2019CS50448

Outline

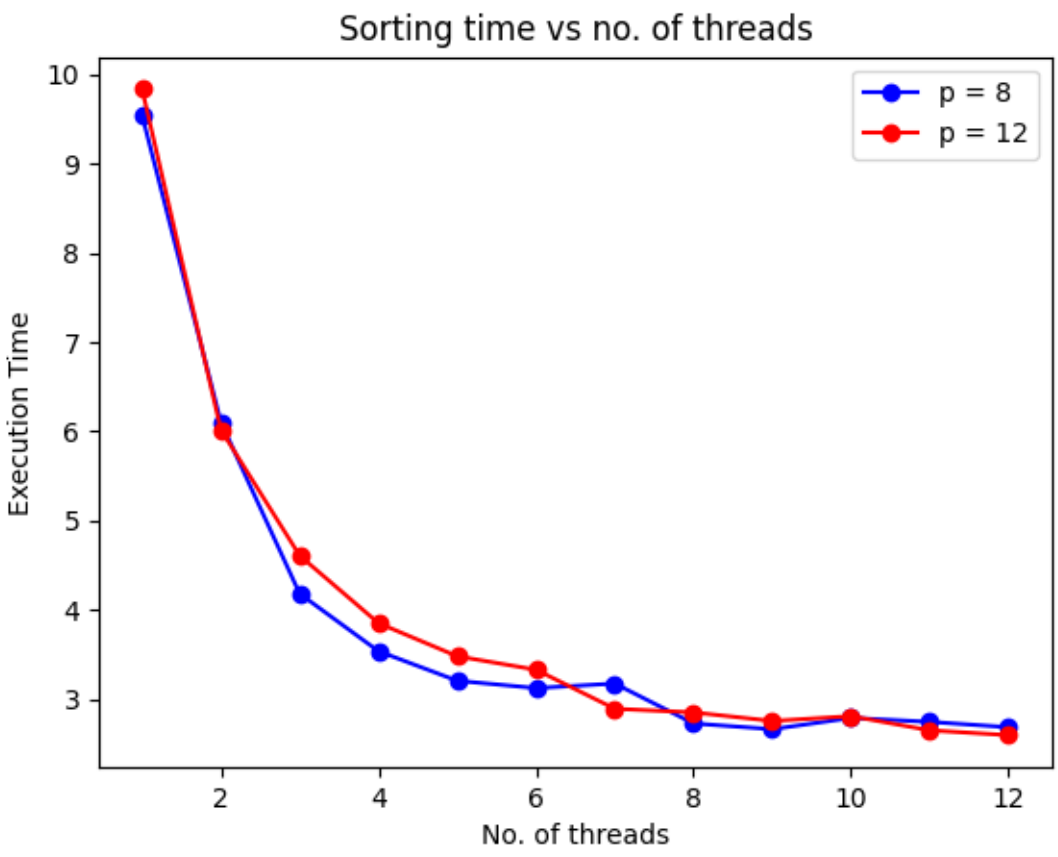
To implement the ParallelSort, we have implemented a ParallelSort and Sequential Sort algorithm which takes num_bucket, data(array), dataSize as input and does an inplace sorting.

ParallelSort was implemented by splitting the array according to the pseudo-splitters and then distributing the data into buckets. The threshold size of bucket was kept 2*(DataSize)/(numBucket) , any array with size less than threshold size is sorted by Sequential Sort which is basically a QuickSort algorithm.

We have also a structure called Partition which takes bucketSize to initialize. This stores the sorted Partition of the array. We concatenate each partition to arrive to the final sorted array.

OpenMp Implementation

First, we have divided the task of sorting different buckets into different threads via the OpenMp task pragma. Then, a barrier(TaskWait) is set to let all the thread complete the sorting. We have also used tasks to copy the value of Partition back to the Parent Array.



We ran the ParallelSort with the size of Array being $3 * 10^7$. We ran the program by varying number of threads from 1 to 12 and varied the number of buckets.

For 8 buckets, we noticed that a single thread took about 9.546 sec to run while this interval scaled down by increasing number of threads, giving us a minimum time of 2.660sec with 9 threads. The time decreased monotonically from thread_num 1 to thread_num 9. It increased afterwards.

For 12 buckets, we noticed that a single thread took about 9.839 sec to run while this interval scaled down by increasing number of threads, giving us a minimum time of 2.593sec with 12 threads. The time decreased monotonically from thread_num 1 to thread_num 12.