# COL380 REPORT

SOURAV(cs1190404)

January 2022

## 1 Design Choice

In this design we use the simple procedure that a given in the assignment document that first divide data into p buckets of n/p+1 size up to n%p and after that we just follow the following procedure:
1. Divide A into $A_0, A_1, \ldots, A_{p-1}$, p buckets of size n/p $\pm$ 1 each as follows. Each $A_i$ contains contiguous elements of A.
2. From each bucket $A_i$, select first p elements as pseudo-splitters. Let $R = [r_0, r_1, \ldots, r_{p*p-1}]$ be the sorted list of p 2 pseudo-splitters. This sorting may use ParallelSort or SequentialSort.
3. Select p-1 equally spaced splitters from R as follows. Let $S = [s_0, s_1, \ldots, s_{p-2}]$ be the selected splitters such that $s_j = R[(j+1) * p]$ for j in 0 to p-2.
4. (Using tasks) Split A into p partitions $B_0, B_1, \ldots, B_{p-1}$ such that for any element a in partition $B_i, s_{i-1} < a \leq s_i$. Assume $s_{-1} = -\infty ands_p = \infty$.
5. Let $n_i$ denote the number of elements in partition Bi. Sort each partition $B_i$ in a separate task which uses SequentialSort$(B_i, n_i) if n_i < Threshold$, and ParallelSort$(B_i, n_i, p)$ otherwise. SequentialSort is sequential sorting of your choice implemented in a task.
6. Return concatenation of sorted partitions $B_i$

The overall orde if oarallelism is O(num of threads).

### 1.1 HPC computation

In this parallel sort algorithm implementation, we have basically parallelised the tasks of partitioning of the original data stream i.e. selection of $p^2$ elements from the original array and then selecting p-1 elements (equally spaced) from that array to partition into buckets. Scalability is basically the tendency of the HPC to deliver high computational power when the amount of resources is increased. In case of HPC we have to see that on increasing the number of cores i.e. basically hardware the capacity of the whole system should also be increase proportionally and hence the time of execution should decrease. To demonstrate strong scaling we have to basically show that on increasing the number of threads the execution time should decrease basically. Now I have drawn the graph between the number of cores used and time taken to execute

by HPC. (I was able to run my code up to 24 cores in HPC because of large amount of queuing time for higher than 24 cores (took more than 28 hours)). The graph between the number of cores and execution time is shown below:

## 2 Data of Graph

N=10000000,P=24,number of Threads = 3

| Number of CPUs | Tmie taken in ms |
|---|---|
| 1 | 4914.46 |
| 2 | 3912.63 |
| 3 | 3786.46 |
| 4 | 3745.2 |
| 5 | 3697.72 |
| 6 | 3748.87 |
| 7 | 3722.7 |
| 8 | 3737.02 |
| 9 | 2703.48 |
| 10 | 2679.57 |
| 11 | 2702.49 |
| 12 | 3633.37 |
| 13 | 3588.05 |
| 14 | 3584.47 |
| 15 | 3568.88 |
| 16 | 3588.87 |
| 17 | 3577.67 |
| 18 | 3587.71 |
| 19 | 3590.87 |
| 20 | 3593.54 |
| 21 | 3596.76 |
| 22 | 3597.18 |
| 23 | 3598.85 |
| 24 | 3599.98 |

N=10000000,P=24,number of Threads = 11

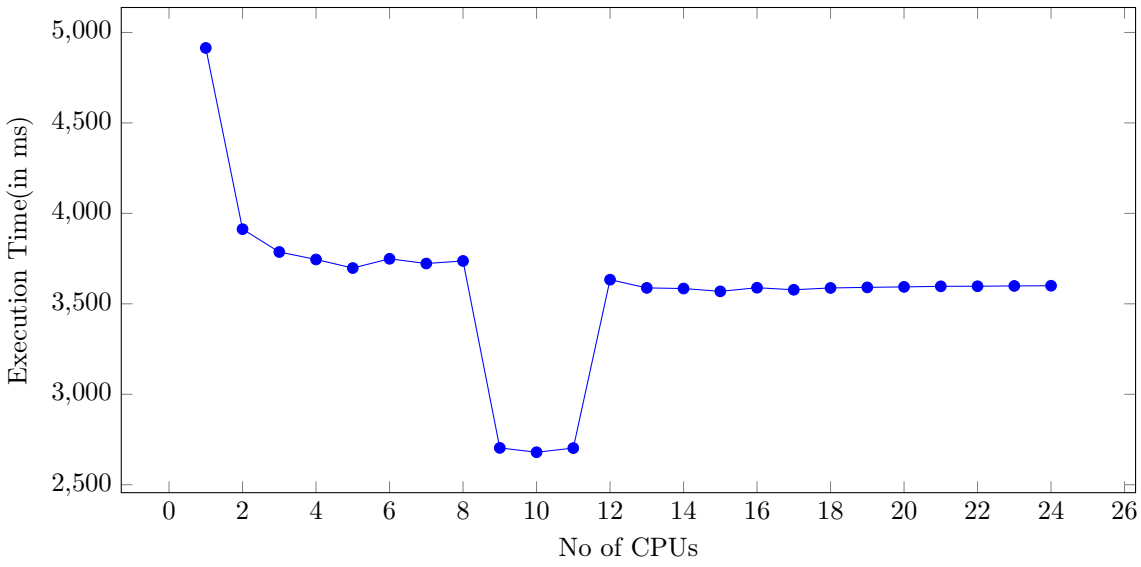
| Number of CPUs | Tmie taken in ms |
|---|---|
| 1 | 4911.77 |
| 2 | 3862.06 |
| 3 | 3858.78 |
| 4 | 3572.14 |
| 5 | 3424.02 |
| 6 | 3426.32 |
| 7 | 3376.77 |
| 8 | 3347.88 |
| 9 | 2417.21 |
| 10 | 2399.56 |
| 11 | 2429.69 |
| 12 | 3203.92 |
| 13 | 3214.53 |
| 14 | 3201.56 |
| 15 | 3170.22 |
| 16 | 3179.52 |
| 17 | 3167.95 |
| 18 | 3172.73 |
| 19 | 3176.45 |
| 20 | 3181.11 |
| 21 | 3185.22 |
| 22 | 3189.65 |
| 23 | 3194.58 |
| 24 | 3199.89 |

N=10000000,P=24,number of Threads = 24

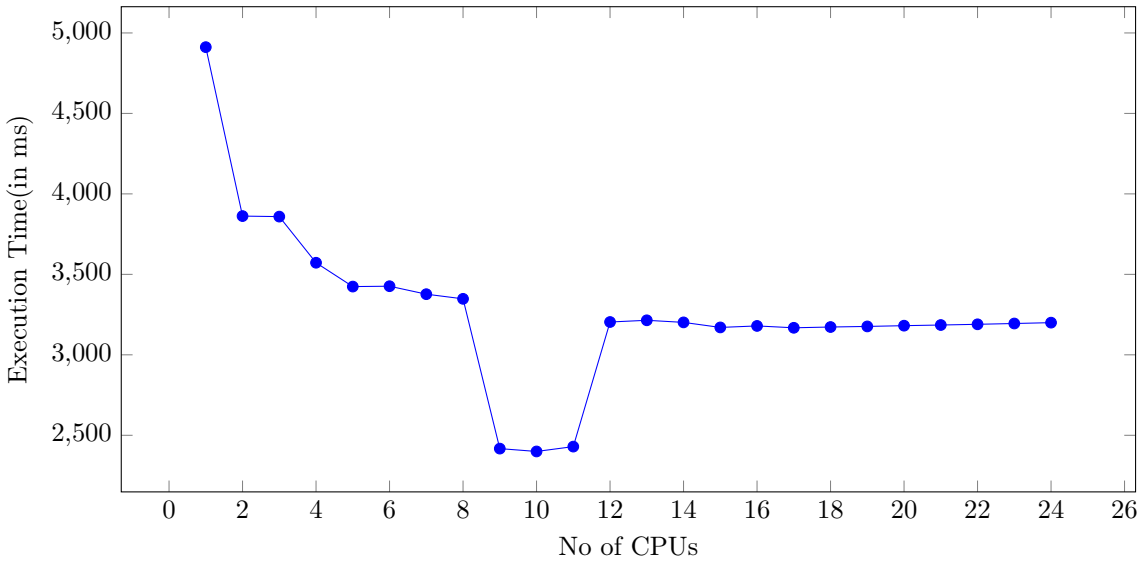
| Number of CPUs | Tmie taken in ms |
|---|---|
| 1 | 4917.61 |
| 2 | 3870.66 |
| 3 | 391.55 |
| 4 | 3562.45 |
| 5 | 3430.52 |
| 6 | 3423.23 |
| 7 | 3356.92 |
| 8 | 3339.43 |
| 9 | 2407.01 |
| 10 | 2420.29 |
| 11 | 2390.65 |
| 12 | 3233.66 |
| 13 | 3147.46 |
| 14 | 3138.08 |
| 15 | 3171.13 |
| 16 | 3175.97 |
| 17 | 3158.74 |
| 18 | 3157.13 |
| 19 | 3152.67 |
| 20 | 3147.65 |
| 21 | 3143.45 |
| 22 | 3140.89 |
| 23 | 3136.54 |
| 24 | 3132.49 |

# 3 Graphs

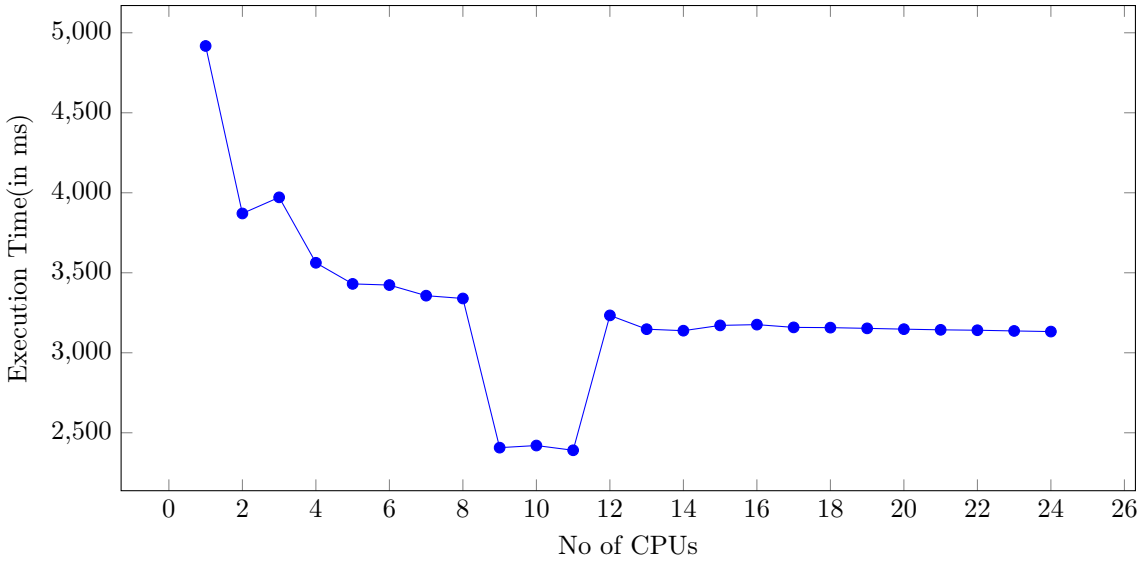
**i)** N=10000000 ,P =24 , Number of Threads = 3




**ii)** N=10000000 ,P =24 , Number of Threads = 11



**iii)** N=10000000 ,P =24 , Number of Threads = 24




**iv)** CPUs VS Execution Time