CSE 570 Introduction to Parallel and Distributed Processing

Name : Ashwin Panditrao Jadhav

UB Name : ajadhav5

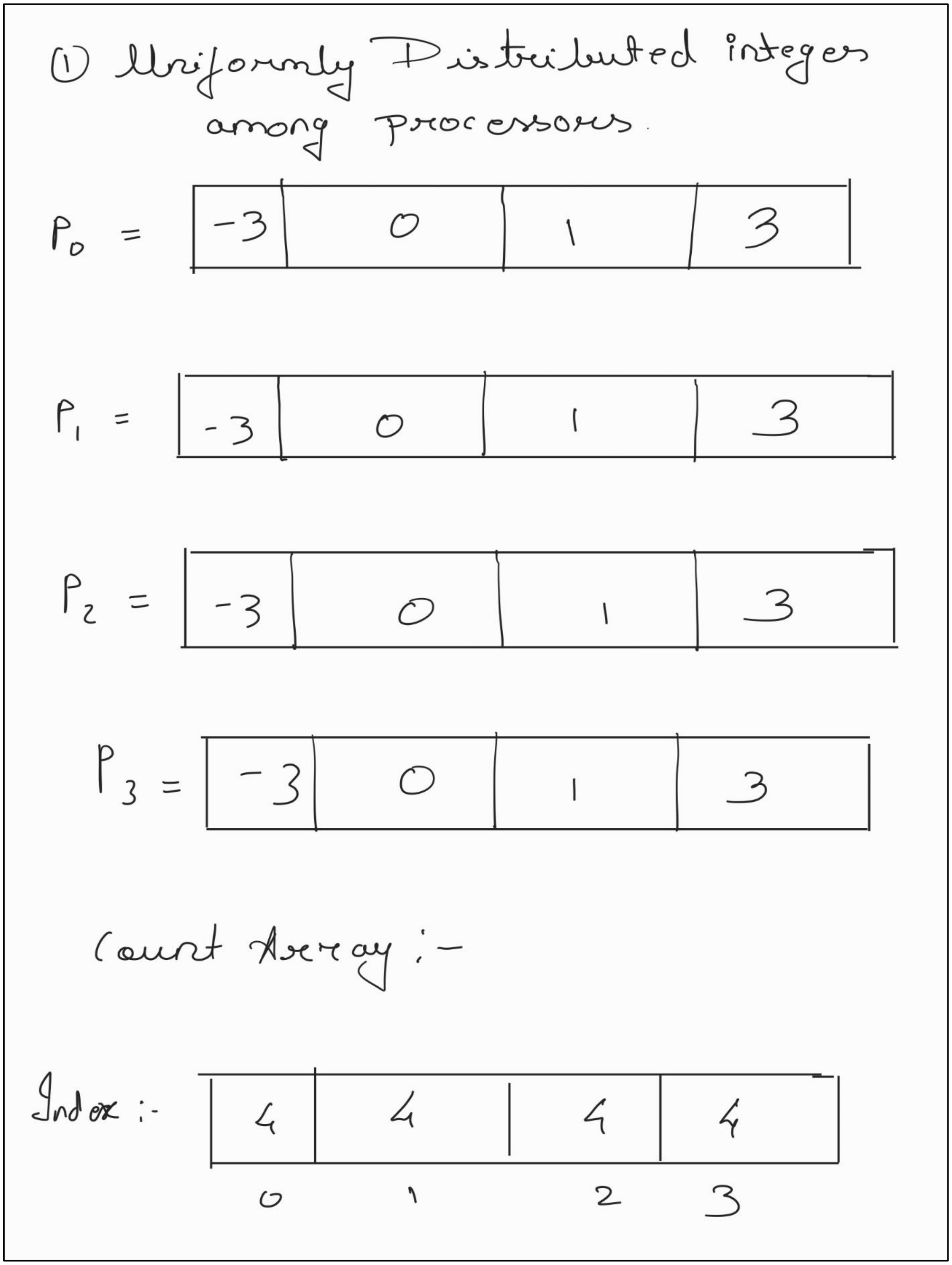
UB Number : 50405435

A1: Sorting Small Integers with MPI

Algorithm for the problem.

1. We have to sort Small Integers parallelly using MPI (Message Passing Interface) in such a way that when keys like xi = xj then both keys are assigned to the same processor and for keys i < j all keys assigned to processor i are smaller than all keys assigned to processor j.
2. The algorithm is using MPI to sort the keys without making any use of comparison operator (< or >). At first, each processor is getting a uniformly distributed chunk of X i.e. X/p chunk of small integers.
3. Then the algorithm is sorting their each piece of chunk using sort function in C++. After that the algorithm is using counting sort to calculate the frequencies of each integer occurring in the array.
4. The algorithm is using MPI_Reduce construct to calculate the frequency of integers at each processor and then using MPI_Bcast to send the frequency data to each processor. After that each processor is re-populating their local array according to the frequencies. In this way all the keys with xi == xj are kept into same processor and also for keys i < j all keys assigned to processor i are smaller that all keys assigned to processor j.
5. Finally, each processor is having their keys which are then sorted using sort function to get the final sorted Xi.

Example :



① Uniformly Distributed integers among processors.

P_0 =

-3	0	1	3

P_1 =

-3	0	1	3

P_2 =

-3	0	1	3

P_3 =

-3	0	1	3

Count Array :-

Index :-

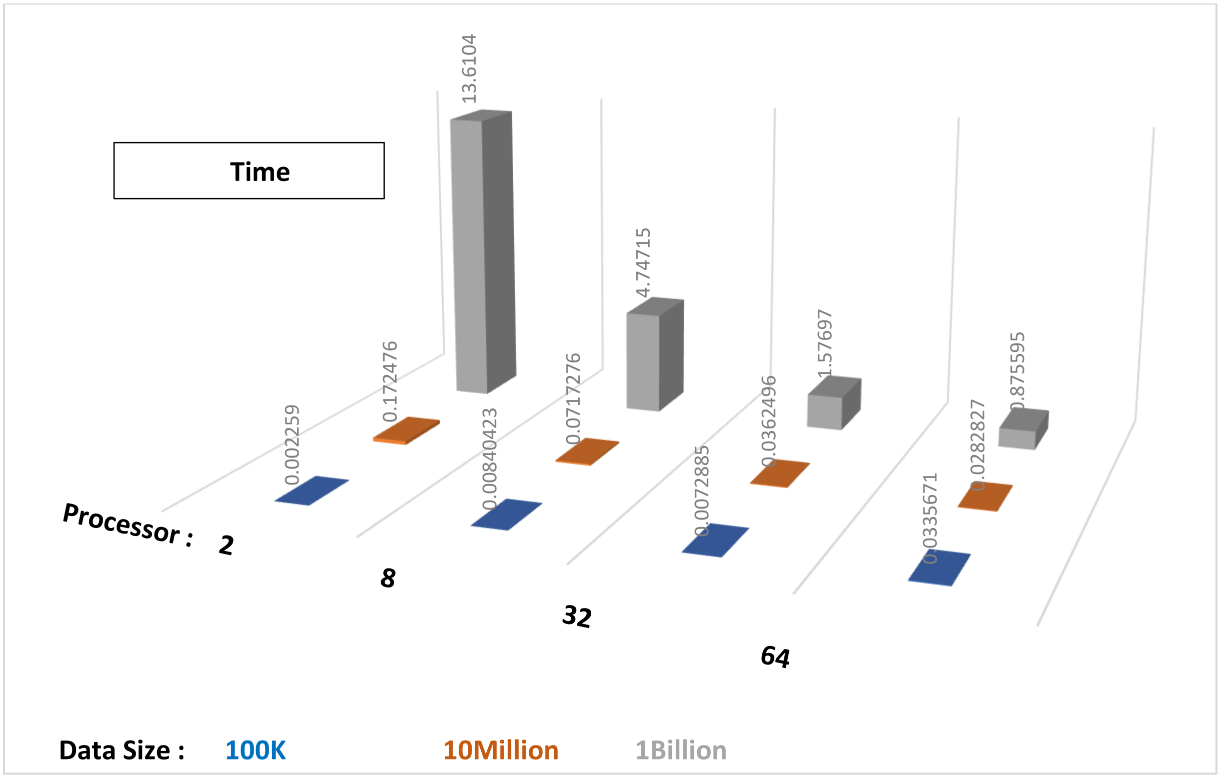
4	4	4	4
0	1	2	3

lalc then Reduce and Broad-cast the frequencies to all processors.

After, that we get the frequencies and regenerate the sorted elements in each processor.

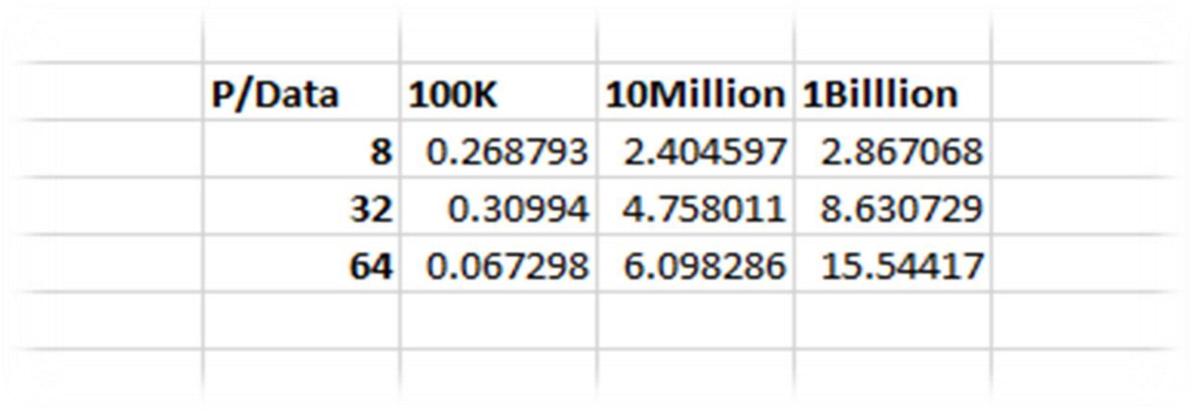
P_0 =	-3	-3	-3	-3
P_1 =	0	0	0	0
P_2 =	1	1	1	1
P_3 =	3	3	3	3

Graph, SpeedUp and Analysis :



Speed Up : Sp = T2 / Tpar

Relative Speed Up of 8/32/64 cores with respect to 2 cores



P/Data	100K	10Million	1Billlion
8	0.268793	2.404597	2.867068
32	0.30994	4.758011	8.630729
64	0.067298	6.098286	15.54417

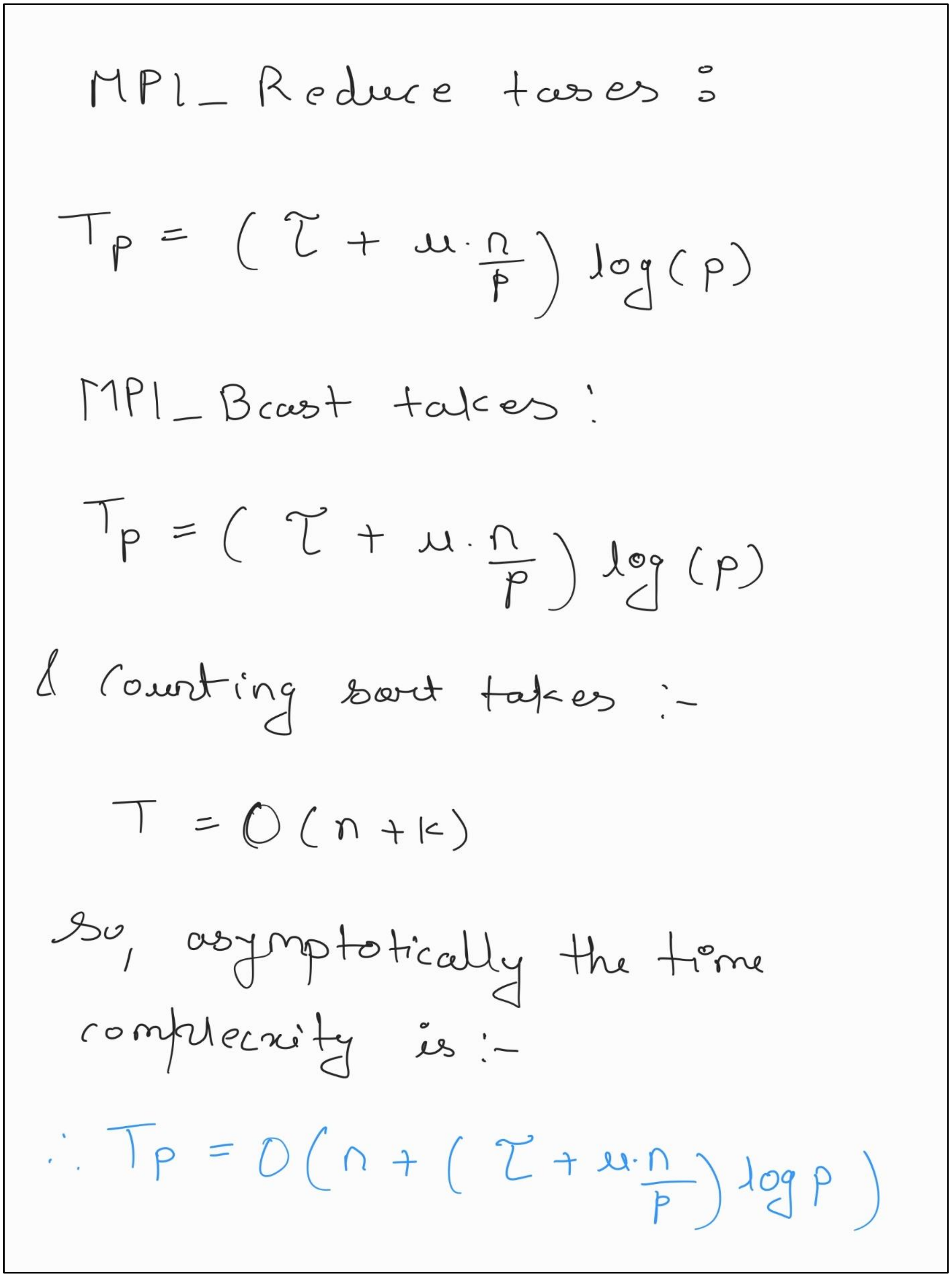
Analysis :

Upon checking SpeedUp which is roughly 26-86 % with respect to 2 processors. Seems that for lower amount of data the speedup achieved is less but for large data sets and large number of processors the speedup achieved is very good.

Is the code Scalable?

Yes, the code is scalable and shows strong scaling. But when large datasets are ran on small number of processors the code/algorithm loses its strong scalability and shows weak scaling as even after the keeping the data size same and then decreasing the number of cores (for 2 cores) the time required for the execution increases drastically.

Time Complexity :



MPI_Reduce takes :

$$T_p = \left(\tau + \mu \cdot \frac{n}{p}\right) \log(p)$$

MPI_Bcast takes :

$$T_p = \left(\tau + \mu \cdot \frac{n}{p}\right) \log(p)$$

& Counting sort takes :-

$$T = O(n + k)$$

so, asymptotically the time complexity is :-

$$\therefore T_p = O\left(n + \left(\tau + \mu \cdot \frac{n}{p}\right) \log p\right)$$