CSE 570 Introduction to Parallel and Distributed Processing

Name : Ashwin Panditrao Jadhav

UB Name : ajadhav5

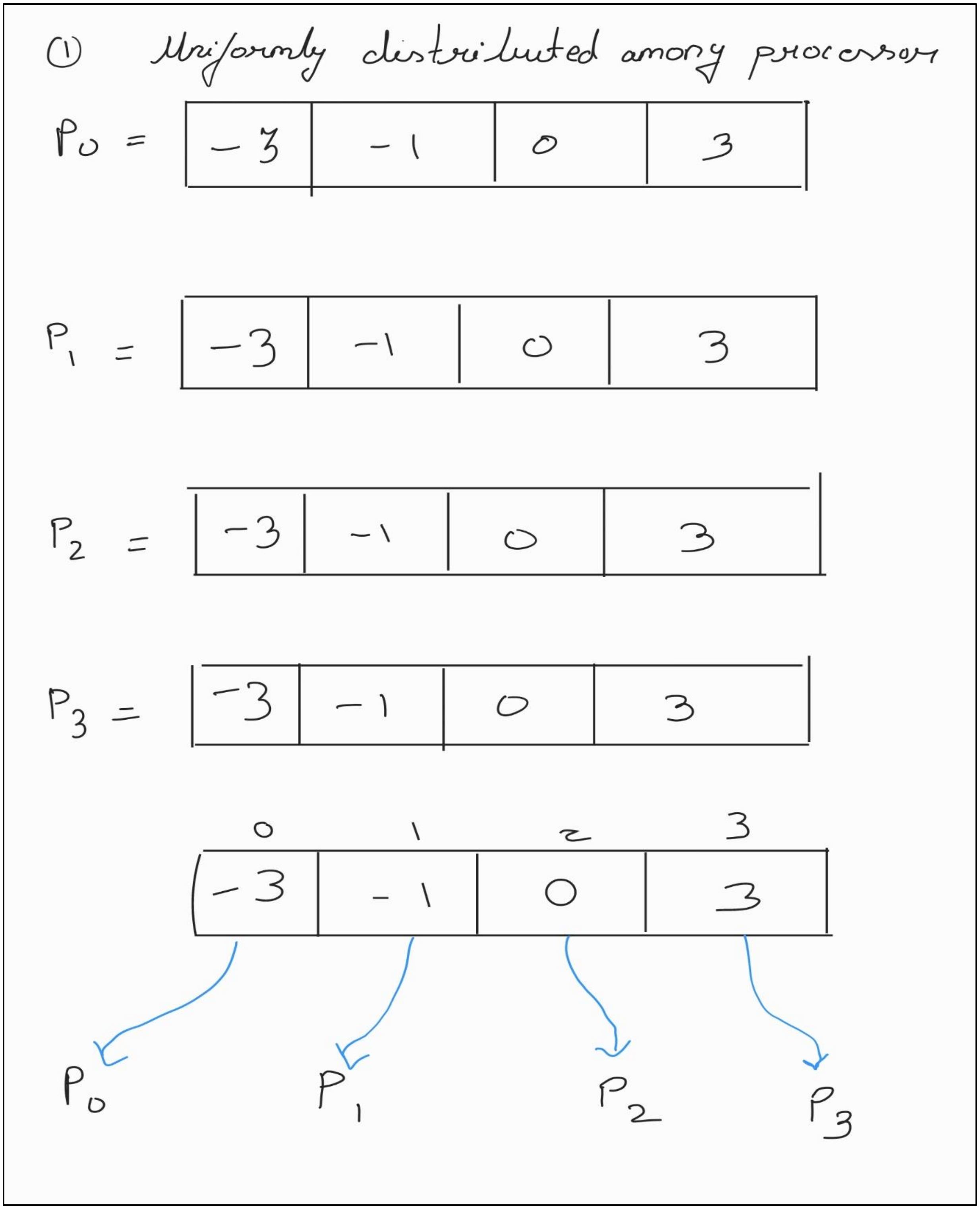
UB Number : 50405435

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 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 find_bucket function to find the index of bucket in which the short int should be placed and then if the rank and the index doesn’t match, the algorithm is using MPI_Send construct to send the short integer to the corresponding rank. 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.
4. Now, after sending the keys the sent keys in local chunks are being erased as well. After MPI_Send each processor is also receiving their corresponding keys using MPI_Recv construct.
5. The algorithm then is using MPI_Barrier to synchronize the processing done at each processor. Finally, each processor is having their keys which are then sorted using sort function to get the final sorted Xi.

Example :

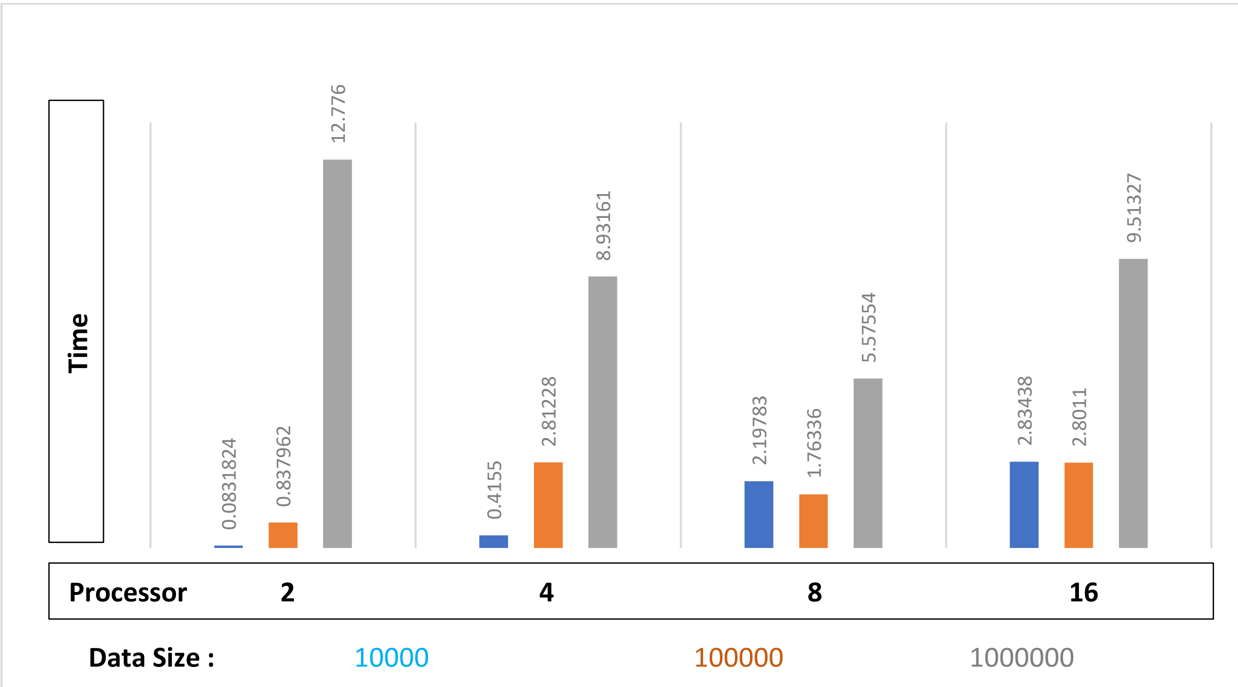


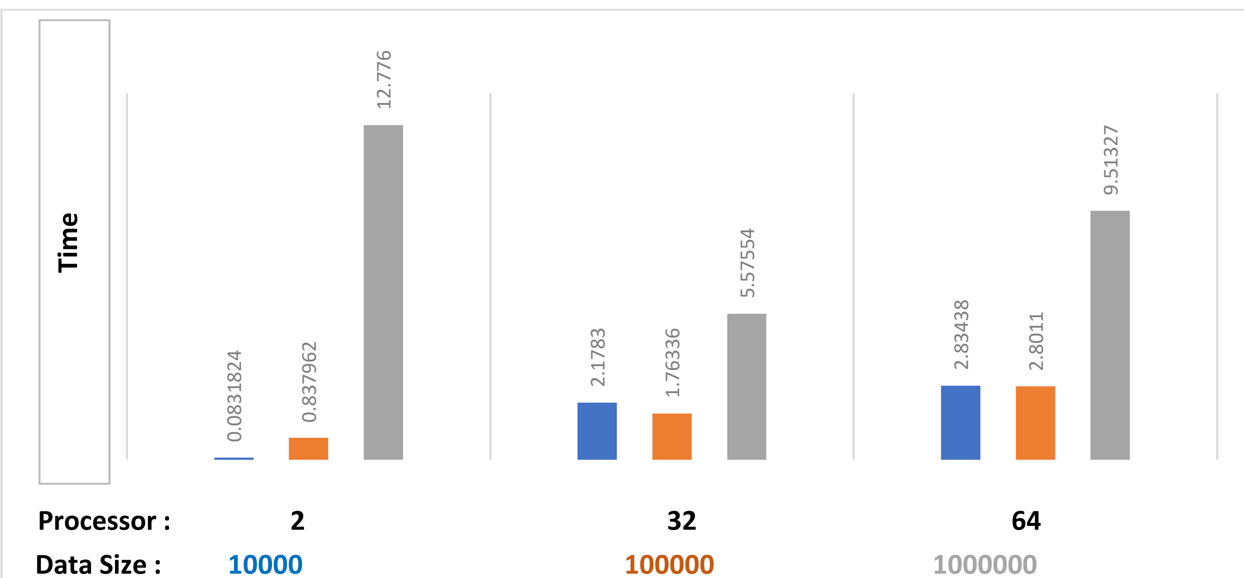
(1) Uniformly distributed among processor
P0 = -3 -1 0 3
P1 = -3 -1 0 3
P2 = -3 -1 0 3
P3 = -3 -1 0 3
0 1 2 3
-3 -1 0 3
P0
P1
P2
P3

(2) After, find_bucket, MPI_SEND and MPI_RECV and sorting

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

Graphs, SpeedUp, Efficiency and Analysis :





Speed Up : Sp = T2 / Tpar

Speed Up in percentage with respect to 2 processors

Data/P	4	8	16	32	64
10000	20.19	37.84	23.4	38.186	23.47
100000	29.7	47.52	29.91	47.52	29.91
1000000	43.04	53.65	34.296	53.89	34.26

Efficiency : Ep = T2 / PTpar

Efficiency in percentage

Data/P	4	8	16	32	64
10000	50.47	37.84	29.34	38.186	29.347
100000	74.49	47.52	30.91	47.5	30.91
1000000	83.39	87.14	74.29	87.37	34.296

Analysis :

Upon checking SpeedUp which is roughly 20-53 % with respect to 2 processors. Seems that more speedup could have been achieved if the algorithm/code would be more strongly scalable. Efficiency, also seems to vary from 29-87 % which is very good when the code runs on 8/16/32 processor with large data.

Is the code Scalable?

Yes, the code is scalable and shows strong scaling. But for large number of processors (like 32 cores) the code loses its strong scalability and shows weak scaling as even after the keeping the data size same and then increasing the number of cores the time required for the execution changes drastically.

Time Complexity :

$$\mathbf{T(n) = (\ \mathit{T} + u*n/p)log(n/p)}$$