**SORTING ALGORITHM’S VISUALIZER**

**“Fast Parallel Sorting Algorithms”**

**SE-A K201700 K201685**

**Q2)**

In the parallel version of the bucket sorting algorithm the algorithm makes use of more space than the actual product of both processors and time . In serial algorithms there is often a trade off between time-space meaning to solve a problem within certain time limits a minimum of space is needed. space requirements will decrease if we allow more time , much work has been devoted to parallel processors. In parallel algorithms there is a similar trade off between time-processors ,to solve problems in a set amount of processors a minimum time is required, and will be reduced with more processors to help. now presenting an algorithm with a three way trade off time-space-processors. Muller and preparata introduced a network capable of sorting n numbers in O log n time , with 0 n^2 processing elements .

Algorithm 1 , we sort n numbers using n parallel processors int O log n time and duplicate numbers are discarded. for implementation each processor to place the value of i in a bucket ci ,this will cause problem so we find an answer to this problem and its to eliminate duplicates of the same number Then, for each number appearing among the numbers being sorted,only one processor (the one with smallest index) will be active when we place i in bucket ci.each processor then determines whether or not its "buddy" is active within the same area , then the processor with higher rank larger index 0 will deactivate ,if buddy is not active or higher rank the processor will continue It is noted that this bucket-sort algorithm requires space O(mn), time T O(log n), and the use of n processors.

Algorithm 2 is similar to the Algo 1 with that it gives actual ranks of the input numbers , we will keep in track of how many processors were active in each block ,if a processors sees an active buddy then only lower buddy will be active after having only one representative of each number that appears to be in the nums that has to be sorted. Now each of the duplicates has a count of the nums of ct's that are equal to it but of higher index. rank(the D value ) is the difference between these quantities + 1 . In Algo 2 space S = O(mn), time T O(log n + log m) and the use of n processors. In algo 2.3 we assume that an area A of the memory has been initialized from 0 .There is a problem though , several entries that are accessed in parallel will have more than one contents pointing to the same location. To solve this problem we have algo 3.

Algorithm 3 uses n ^ 3/2 processors with time complexity of O log n , we partition the n nums into n ^1/2 groups , within each group for each element j we determine the count ,then we apply bucket sort on the elements using count[j] as key for the jth element in the group effectively using enumeration sort ,all elements do a binary search on the n 1/2 groups ,then finally a bucket sort on all elements n using j as key. Modifying Algo 3 a little gives rise to algo 4 below.

Algorithm 4 , uses n ^ 4/3 processors firstly we partition the input numbers into groups each having n ^1/3 elements , within each group we determine count for each elements , then apply bucket sort to the count which rearranges them into rank order , then we divide the n ^ 2/3 groups into n^ 1/2 sectors having n^1/3 groups, now within sectors we do binary search for each element, then bucket sort of elements making them in rank order , then we calculate the value of count (j,k) once again ,Then evaluate count[j] = the sum (over k) of count[j, k], finally do a bucket sort of all n elements, time complexity for this will be O (k log n) , that uses n ^ 1+1/k processors.

All these algorithms have memory-fetch conflicts but we successfully avoided memory-store conflicts.