Algorithms Summary Q2
20k-0425

The research paper uses serial bucket sort algorithm(O(logn)) to derive various other concurrent sorting algorithms that work on integer arrays. The model of computation is SIMD(Single Instruction Multiple Data) where eac processor can has a common memory core and also possess a local memory chunk. The second method this paper presents is an algorithm that can sort n integers on O(klogn) time, using $n^{1+(1/k)}$ processors where K is and constant of integer type.

The algorithms proposed give an excellent example of the space-time complexity trade off where the time complexity can be drastically reduced but at the cost of requiring additional space to function.

An overview of the 4 algorithms that are proposed in this research paper are as follows;

Algorithm 1: It is a basic implementation of parallel bucket sort where the time complexity is O(logn) for array [0, m-1]. Due to the method used to allocate memory and thus create buckets each duplicate number has to be discarded or memory conflicts arise as different processors will compute the same element n number of time n being the occurrence of duplicate values.

Algorithm 2(2.1-2.3): The basic running of this algorithm is the same as algorithm 1 but it address the issue with duplicate values by assigning a count and a flag to each processor that has computed on a duplicate value thus removing the dependency of memory conflicts that may have arose. The algorithm, unlike algorithm 1, runs on O(logn + log,) time and on O(mn) space.

Algorithm 3: Uses an algorithm proposed by Gavril which merges two sets in O(logn) time. This implementation used $n^{3/2}$ processors, the array is partitioned into 2 subparts ($n^{1/2}$) each having $n^{1/2}$ elements. These groups are then sorted individually by bucket sort and then are merged in using $n^{1/2}$ processors. The total number of processors used in this algorithm are $n^{3/2}$ and no memory conflits arise.

Algorithm 4: it modifies algorithm 3 so that the number of groups that are formed in partitioning of the array are of size $n^{2/3}$ with each having $n^{1/3}$ elements. This reduces the processor requirements to $n^{4/3}$ and maintains the time complexity of O(logn)

In conclusion, the 4 sorting algorithms optimize the time complexity for sorting an array in a parallel manner and minimize the vertical wastage of processing resources.