Algo Summary

Parallel Sorting Algorithm

This study presents various methods for concurrently sorting an array of integers using n processors and a parallelized version of the serial bucket sort algorithm, which takes O(logn) time. The single instruction, multiple data (SIMPD) model of computation is used, in which each processor has access to both a shared memory and a smaller local memory.

Furthermore, a second procedure is presented which is capable of sorting n integers in *O(klogn)* time, using n1+1/k processors (where k is an arbitrary integer).

A good illustration of the trade-off between time, space, and the quantity of processing units used in parallel computations is the proposed parallel bucket sort algorithm. At the expense of taking up more space, the algorithm significantly reduces computation time and the number of processors needed.

The article describes four algorithms in total, the first of which (Algorithm 1) performs a basic implementation of the parallel bucket sort on n parallel processors in time O(logn) for integers in the range [0, m-1]. The disadvantage of this implementation is that duplicate numbers must be discarded because of how processors are temporarily allocated to place each integer in its proper bucket; otherwise, memory conflicts would occur because of multiple processors simultaneously storing different numbers in the same bucket.

The structure of Algorithm 2.1-2.3 is identical to Algorithm 1, but it avoids the need to eliminate duplicate numbers from the array by keeping track of the number of active processors rather than just flagging them with a flag bit. There will only be one active processor for each memory area, preventing memory conflicts while bucketing elements into their proper locations. This algorithm requires n processors and executes in O(logn + logm) time on O(mn) space.

The foundation of Algorithm 3 is an expansion of a Gavril algorithm that merges two sets in O(logn) time. For this specific implementation, n3/2 processors are required. It divides the array's elements into n1/2 groups, with n1/2 elements in each group. Bucket sorting is used to sort the components of each group, and the sorted groups are then combined using a logarithmic time method with n1/2 processors per component for a total of n3/2 processors, all while avoiding memory conflicts.

The final suggested algorithm, Algorithm 4, is a straightforward modification of Algorithm 3, requiring just n4/3 processors while preserving logarithmic time complexity. The array was divided into groups as part of the algorithm's update. There are n2/3 groups created each with n1/3 items in size.

To sum up, these sorting algorithms provide a simple method for quickly sorting an array of integers while maximizing parallel run-time and reducing vertical resource waste.