Bucket sort works in two phases. In the first phase, we place the elements into n buckets where the jth bucket holds all elements whose first m binary digits correspond to the number j. For instance, if n = 210, bucket 3 contains all elements whose first 10 binary digits are 0000000011. When j < l, the elements of the jth bucket all come before the elements in the l th bucket in the sorted order. Assuming the each element can be placed in the appropriate bucket in O(1) time, the phase requires only O(n) time. Since it is assumed that the elements to be sorted are chosen uniformly, the number of elements that fall into a specific bucket follows a binomial distribution B(n, 1/n). In the second phase, each bucket is sorted using any standard quadratic time sorting algorithm such as Quick sort and Insertion sort or recursively using bucket sort. Each bucket is concatenated in order produces the sorted order for the elements. Under the uniform distribution of the input, bucket sort falls naturally into the balls and bins model in which the elements are balls, buckets are bin, and each ball falls uniformly at random into a bin. This follows a binomial random variable B(n, 1/n) whose expectation is n. Therefore, the expected time spent in the second phase of bucket sort is only O(n).

In the parallel version of bucket sorting algorithm, k threads are used to perform the sorting. The original array of N elements is partitioned into k sets each has N/k elements. This is preparation step before each thread is assigned to put N/k elements in the buckets based on their values. After all the elements have been put in the M buckets, the next step is partitioning the buckets into k sets each has M/k buckets. Now each thread is assigned to sort M/k buckets by using any standard quadratic time sorting algorithm such as Quick sort and Insertion sort or recursively using bucket sort.

Parallel Bucket Sorting Algorithm.

N: number of unsorted elements

M: number of buckets,

k: number of threads

1: Initialize an array of empty “buckets”.

2: Partition the original array into k sets, each has N/k elements.

3: In parallel, each thread performs the following: Go over N/k elements in the original array and put them in the bucket based on their values.

4: Partition buckets into k sets, each has M/k buckets.

5: In parallel, each thread performs the following: Sort M/k non-empty buckets.

6: Visit the array of buckets in order and put the sorted elements back to the original array