COL730-Assignment1
Mohammed Zia Kamran
2017TT10925

In This Assignment we are supposed to implement sorting Algorithms using Message passing Interface for distributed memory. We had to Implement parallel collective function to quick sort, merge sort, radix sort, and any other sort of our choice.

Merge Sort :

In Merge sort I used an exchange method which simply means for any two processors when interacting the lower rank will take the smaller values and the other one will take the higher value. I did this by send and receive between these two processors and then following merging two lists according to their requirement. Then there are many other ways too but this one was good when we use the parent conception where each processor know whom to send and receive.

The main algorithm is each processor is knowing their parent and co-child (i.e left child or Right child) so in every height from 0 to log2(N) each processor merge their list and then left become parent for that height and child for next height if we consider leaf nodes are at height 0, then repeating this way we reach to the max height where only rank 0 is able to reach and this confirms that rank 0 is now having all the shortest number globally and length is as it was having , also in sorted manner. But still we cannot say anything about others so we have to repeat this till the size-1 of comm (i.e the number of processor -1).
That's it, at the end we will be having all the processors having original length data and in a globally sorted manner. In case of Odd processor, there is no co-child for it so in this case it will be considered as parent and directly sent to the next height without any send or receive.

Radix Sort :

In Radix sort, I have chosen 4 bit to make a bucket as given the key will be 32bit. Then In the the beginning of Algorithm I have made 16 buckets and arranged the data according to MSD (most significant 4-digit) for each processor and then using All gather made a Histogram of distribution in buckets for global data. Then correspondingly allocated the processor rank to a group of data which suits the size of original data, greater than or equal to. So by doing this each

processor from left to right will have partial sorted data (possibly no data in case of the last processor) then each will sort them locally.

After the local sorting now we have each processor having data globally and locally sorted manner but the problem is not the same size as before. So there is still redistribution needed among the processor cyclic movement of data is required but that increases the overhead.

Quick Sort :

For Quick sort, I called a recursive function and divided the whole communication into two groups using MPI_split. In each recursive call there is key broadcasted from rank 0 of that group and others are dividing their data into two parts according to pivot. Then similar to radix sort this time we made two distributions one of all lewer elements then pivoted to the upper half number of processor and larger elements from all to the lower half of the processor. We defined Middle of the processor on the basis of linear distribution of lower to greater data.

Letsay S denotes the all smaller than equal pivot and L denotes all larger than pivot and N is the total element in this group. So Mid = ceil(S*p/N), this gives the number of processors required for smaller elements and others for larger elements.

Best Sort :

For best sort I just used the merge sort algorithms because it has lesser overheads than the other two. I have tried some examples on different processors and different data, mergesort is giving better results than others in my algorithm.