How I parallelized the quicksort algorithm

I implemented a parallel quicksort function with the following breakpoints: when only one process remains in the assigned group or when there is no data left to sort.

The process is divided into three main stages:

1. Choosing and Sharing Global Pivot:

Each process selects a local pivot randomly. Using MPI_Allgather, all processes gather the local pivot data so that each process has access to the complete set of pivots. From this data, each process calculates the median to determine the global pivot.

2. Partitioning Data:

Each process partitions its local data into two parts based on the global pivot: values smaller than or equal to the pivot and values greater than the pivot.

3. Grouping and Data Exchange:

The processes are divided into two groups. The first group is responsible for the smaller values (less than or equal to the pivot), while the second group handles the larger values (greater than the pivot). The first group sends its "greater" values to the second group, and the second group sends its "less" values to the first group. This process is repeated recursively for each group until the base condition is met.

This recursive approach ensures that the data is efficiently sorted across all processes.

How I ensured that at the end of the operation each worker had N/P elements

Here is how I ensured that at the end of the operation, each worker had exactly N/P elements:

1. Collecting Data Sizes on Rank 0:

Each worker's data size was gathered on Rank 0 using MPI_Gather. This allowed Rank 0 to know how much data each worker held.

2. Gathering and Organizing Data on Rank 0:

Using MPI_Gatherv, Rank 0 collected all the data from every worker into a single array. The data was stored in order, corresponding to each worker's Rank.

3. Partitioning Data Evenly:

On Rank 0, the entire dataset was partitioned into N/P chunks. This ensured that each worker would receive an equal amount of data.

4. Redistributing Data:

Rank 0 used MPI_Scatterv to redistribute N/P elements to each worker. This ensured every worker received the correct portion of the dataset.

5. Replacing Local Data on Workers:

Each worker replaced its local data with the redistributed N/P elements, ensuring uniformity across all processes.

This process guaranteed that, at the end of the operation, every worker had exactly N/P elements.

Results:

1) P = 1

1-1) N = 1,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 1 ./qs_mpi 1000000 output.txt
Sort Time: 1.6359s
```

1-2) N = 10,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 1 ./qs_mpi 10000000 output.txt
Sort Time: 28.5794s
```

1-3) N = 100,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 1 ./qs_mpi 100000000 output.tx
t
Sort Time: 223.3548s
```

2) P = 2

2-1) N = 1,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 2 ./qs_mpi 1000000 output.txt
Sort Time: 1.4539s
```

2-2) N = 10,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 2 ./qs_mpi 10000000 output.txt
Sort Time: 12.4190s
```

2-3) N = 100,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 2 ./qs_mpi 100000000 output.tx
t
Sort Time: 152.1446s
```

3) P = 4

3-1) N = 1,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 4 ./qs_mpi 1000000 output.txt
Sort Time: 1.2103s
```

3-2) N = 10,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 4 ./qs_mpi 10000000 output.txt
Sort Time: 9.4968s
```

3-3) N = 100,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 4 ./qs_mpi 100000000 output.tx
t
Sort Time: 127.0283s
```

4) P = 8

4-1) N = 1,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 8 ./qs_mpi 1000000 ouput.txt
Sort Time: 0.6231s
```

4-2) N = 10,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 8 ./qs_mpi 10000000 ouput.txt
Sort Time: 7.4181s
```

4-3) N = 100,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 8 ./qs_mpi 100000000 ouput.txt
Sort Time: 69.3883s
```

5) P = 16

5-1) N = 1,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 16 ./qs_mpi 1000000 ouput.txt
Sort Time: 0.5650s
```

5-2) N = 10,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 16 ./qs_mpi 10000000 ouput.txt
Sort Time: 4.6423s
```

5-3) N = 100,000,000

```
joo00032@csel-plate01:/home/joo00032 $ mpirun -np 16 ./qs_mpi 100000000 ouput.tx
t
Sort Time: 44.6141s
```