Sortproject Report

As this project is using histogramming sorting algorithm to sort numbers in a parallel approach in distributed memories. What I did was using MPI to communicate between processes for data exchanging and using OpenMP for synchronizing different threads.

For rebalacing, I first tried to get total number of processes and MPI rank, then find out the total data count and local data count, to redistribute, I created a window in which to count the size of the new rebalanced array. So it can redistribute data from overload nodes to underload nodes.

In findSplitters, I inserted splitters into all data evenly using distributed histogramming method. I found local min and max, and global min and max, then splitters are generated within the regions.

For moveData, I tried to use fence to realize the one-side communication and put data in the corresponding positions between each splitter.

For sort function, since it is pre defined, so I didn't make any changes upon it.

```
Testing
[==========] Running 5 tests from 4 test cases.
[----------] Global test environment set-up.
[----------] 2 tests from rebalance
[ RUN      ] rebalance.correct
[       OK ] rebalance.correct (169 ms)
[ RUN      ] rebalance.balanced
[       OK ] rebalance.balanced (144 ms)
[----------] 2 tests from rebalance (341 ms total)

[----------] 1 test from findSplitters
[ RUN      ] findSplitters.balanced
[       OK ] findSplitters.balanced (400 ms)
[----------] 1 test from findSplitters (410 ms total)

[----------] 1 test from moveData
[ RUN      ] moveData.correct
[       OK ] moveData.correct (138 ms)
[----------] 1 test from moveData (149 ms total)

[----------] 1 test from sort
[ RUN      ] sort.correct
[       OK ] sort.correct (0 ms)
[----------] 1 test from sort (0 ms total)

[----------] Global test environment tear-down
Finalizing MPI...
[==========] 5 tests from 4 test cases ran. (909 ms total)
[  PASSED  ] 5 tests.
```

Local testing in docker.

Benchmark results

```
Duration_Rebalance 0.144058 s

Duration_Splitters 0.41465164 s

Duration_Move 1.06546 s

Duration_Total 1.62417 s

Duration 1.62417 s
```

I chose one-sided messaging because it is faster than point-to-point calls and it is easy to implement. Within different one-sided communication methods, I chose fence synchronization and MPI_Put to transfer data to processes they belong.

Scaling Study