Faculty of ICT, Mahidol University

Parallel Sorting Algorithm: Parallel Ranksort via MPI

By

6488004 Kittipich Aiumbhornsin

6488089 Pattaravit Suksri

6688246 Mohammed Borhan Hussaini

Submitted to

ITCS443 Parallel and Distributed Systems

Asst. Prof. Dr. Sudsanguan Ngamsuriyaroj

Dr. Ekasit Kijsipongse

Dr. Ittipon Rassameroj

A report submitted

as the partial fulfillment of the requirements for the assignment

October 2023

Topic: Parallel Rank sort via MPI

Program Explanation Diagram

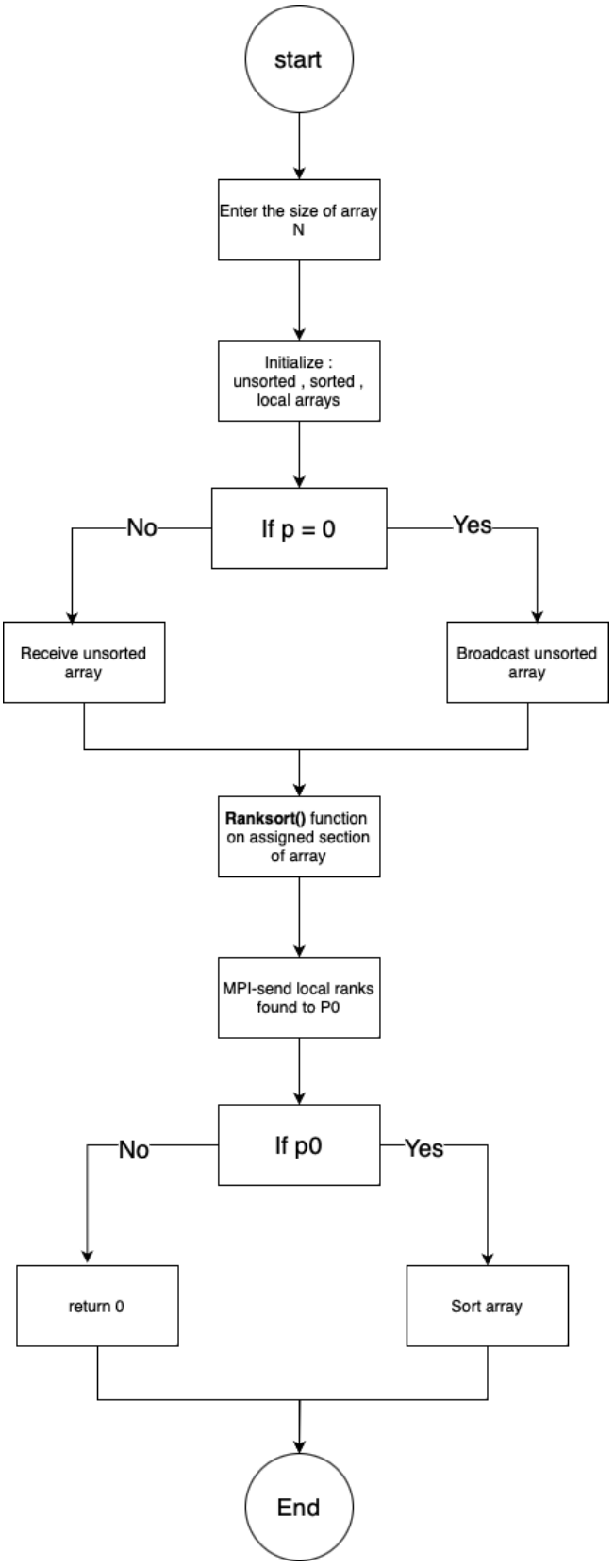


Diagram explanation :
The code begins with assigning memory for the whole unsorted, sorted arrays and the array for local ranks in each process. Process 0 will fill the unsorted array with random numbers, then broadcast it to all processes so each has a copy. After that, each process finds the rank of all the elements in their assigned section of the larger array and saves them in a smaller local array. At last, each process will send their arrays with their ranks to process 0 and it will then iterate through each array of ranks and place the unsorted numbers in the sorted list accordingly.

How to run the program

- Extract the zip file.
- On the terminal, change directory to where the zip file has been extracted using command **cd** and follow with the directory path.
- List the files in the directory by using command **ls** and find file name **parallelRankSortMPI2023.c** to make sure that the source code is available.
- On the terminal type command **mpicc -o ranksort ranksort.c** and click enter, to compile the program.
- Then, to run the program, on the same terminal type command

  mpirun -np 2 ranksort and click enter. (Note: For running on 2 nodes/processors. If you would like a different number, replace the number 2 in the command with that number.)
- After that, it will prompt to get the amount of numbers to be generated and sorted, key in only the number in integer you want.
- Wait and see the cool result!

Testing results

N	Time (Seconds)				
N = 10000	0.53	0.14	0.07	0.04	0.03
N = 100000	53.44	13.36	7.02	4.44	3.35
N = 500000	1414.3	352.78	184.85	116.19	87.64
N = 1000000	5345.27	1353.31	723.43	452.5	337.6
N = 2000000	21430.65	5337.5	2656.05	1662.89	1249.3
Processors	1	4	8	12	16

Note: The name of the program run when testing is not the same as the one in the submitted version because we have renamed it.

N = 10,000

```
-bash-4.1$ mpirun -np 1 pleaserun

Execution time for n = 10000: 0.530000 seconds
```

```
-bash-4.1$ mpirun -np 4 pleaserun

Execution time for n = 10000: 0.140000 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 10000: 0.070000 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 10000: 0.070000 seconds
```

```
-bash-4.1$ mpirun -np 16 pleaserun

Execution time for n = 10000: 0.030000 seconds
```

N = 100,000

```
-bash-4.1$ mpirun -np 1 pleaserun

Execution time for n = 100000: 53.440000 seconds
```

```
-bash-4.1$ mpirun -np 4 pleaserun

Execution time for n = 100000: 13.360000 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 100000: 7.020000 seconds
```

```
-bash-4.1$ mpirun -np 12 pleaserun

Execution time for n = 100000: 4.440000 seconds
```

```
-bash-4.1$ mpirun -np 16 pleaserun

Execution time for n = 100000: 3.350000 seconds
```

N = 500,000

```
-bash-4.1$ mpirun -np 1 pleaserun

Execution time for n = 500000: 1414.295813 seconds
```

```
-bash-4.1$ mpirun -np 4 pleaserun

Execution time for n = 500000: 352.780342 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 500000: 184.847727 seconds
```

```
-bash-4.1$ mpirun -np 12 pleaserun

Execution time for n = 500000: 116.195460 seconds
```

```
-bash-4.1$ mpirun -np 16 pleaserun

Execution time for n = 500000: 87.640000 seconds
```

N = 1,000,000

```
-bash-4.1$ mpirun -np 1 pleaserun

Execution time for n = 1000000: 5345.273322 seconds
```

```
-bash-4.1$ mpirun -np 4 pleaserun

Execution time for n = 1000000: 1353.317483 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 1000000: 723.435721 seconds
```

```
-bash-4.1$ mpirun -np 12 pleaserun

Execution time for n = 1000000: 452.504312 seconds
```

```
-bash-4.1$ mpirun -np 16 pleaserun

Execution time for n = 1000000: 337.645396 seconds
```

N = 2,000,000

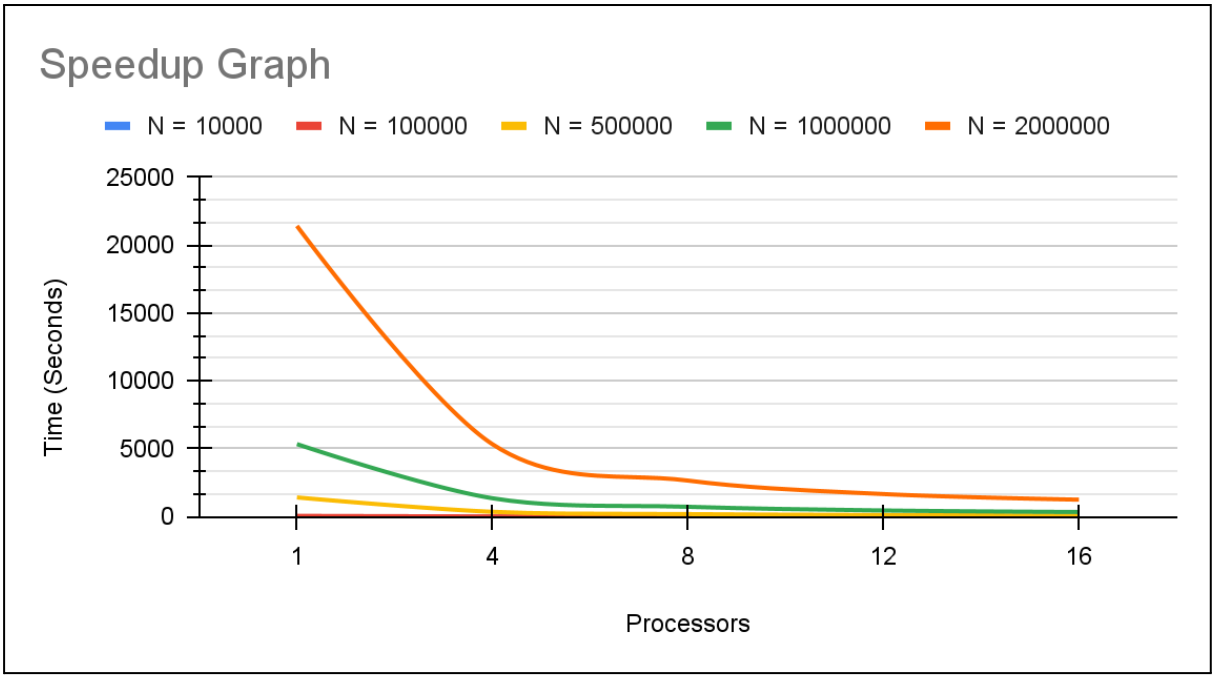
```
-bash-4.1$ mpirun -np 1 pleaserun

Execution time for n = 2000000: 21430.653346 seconds
```

```
-bash-4.1$ mpirun -np 4 pleaserun

Execution time for n = 2000000: 5337.518835 seconds
```

```
-bash-4.1$ mpirun -np 8 pleaserun

Execution time for n = 2000000: 2656.054326 seconds
```

```
-bash-4.1$ mpirun -np 12 pleaserun

Execution time for n = 2000000: 1662.893452 seconds
```

```
-bash-4.1$ mpirun -np 16 pleaserun

Execution time for n = 2000000: 1249.324593 seconds
```



As the table shown above, we can see that as the amount of the processor increases, the faster the sorting process can be done. However, the amount of the items to be sorted is also another factor. As is known, sequential rank sort has a time complexity of O(n^2), and this is proved by the result times for 1 processor increasing quadratically, eg. 10,000 to 100,000 has a 100x increase with a 10x increase in n.