**HW4 Report**

Sai Namith Garapati

UIN: 832001176

1. (80 points) Complete the MPI-based code provided in qsort\_hypercube.cpp to implement the parallel quicksort algorithm for a *d*-dimensional hypercube with *p*=2*d* processors. 60 points will be awarded if the code compiles and executes the following command successfully.

mpirun -np 2 ./qsort\_hypercube.exe 4 -1

5 points will be awarded for each of the following tests that are executed successfully.

mpirun -np 4 ./qsort\_hypercube.exe 4 -2

mpirun -np 8 ./qsort\_hypercube.exe 4 -1  
mpirun -np 16 ./qsort\_hypercube.exe 4 0  
mpirun -np 16 ./qsort\_hypercube.exe 20480000 0

**Solution:** The MPI-based code that has been provided in the qsort\_hypercube.CPP is modified by including the required 9 calls to MPI routines.

The code compiled successfully, and all the above-provided commands are executed successfully as shown in the output below.

Text

Description automatically generated

1. (5 points) *Weak Scalability Study*: Run your code to sort a distributed list of size *n*×*p* where *n* is the size of the local list on each process and *p* is the number of processes. For your experiments, use *n*=20,480,000 and *p* = 1, 2, 4, 8, 16, 32, and 64. Set type=0. Plot the execution time, speedup, and efficiency of your code as a function of p. Use logarithmic scale for the x- axis.

Note that the size of the list to be sorted is proportional to the number of processes *p*. In order to get speedup for a specific value of *p*, you need to determine the execution time to sort a list of size *n*×*p* with **one** process. As an example, speedup for *p* = 4 is the ratio of execution time for a list of size 81,920,000 with one process (T1) to the execution time for a list of size 20,480,000 with 4 processes (T4).

**Solution:**

The code is run to sort a distributed list of size n x p for p = 1, 2, 4, 8, 16, 32 and 64 and the results are as follows:

(Along with this the code is run for a distributed list of size n x p on 1 process to obtain the speedup)

Text

Description automatically generated

The plot of execution time vs number of processors for p = 1, 2, 4, 8, 16, 32, 64

Chart, line chart

Description automatically generated

The plot of speedup vs number of processors for p = 1, 2, 4, 8, 16, 32, 64

Chart, line chart

Description automatically generated

The plot of Efficiency vs number of processors for p = 1, 2, 4, 8, 16, 32, 64

Chart, line chart

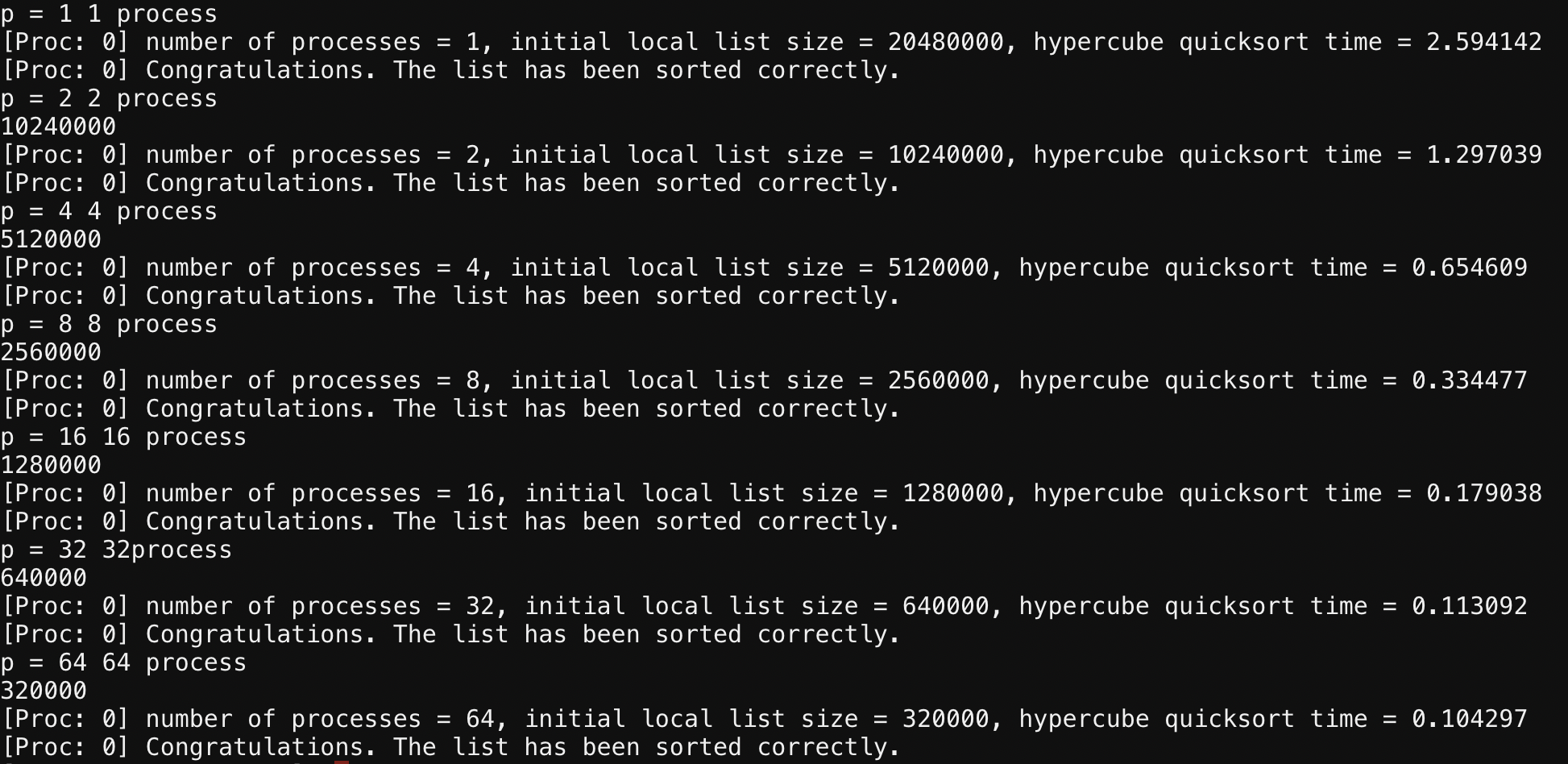
Description automatically generated

1. (5 points) *Strong Scalability Study:* Now run your code with *n*=20,480,000/*p* where *p* = 1, 2, 4, 8, 16, 32, and 64. Set type=0. Plot the execution time, speedup, and efficiency of your code as a function of p. Use logarithmic scale for the x-axis.

Unlike the weak scalability study, here the size of the list to be sorted remains unchanged at 20,480,000 even as you increase the number of processes. To determine speedup for any *p* you need to compare the execution time on *p* processes to the execution time for a list of size 20,480,000 with one process

**Solution:**

The code is run to sort a distributed list of size n/p for p = 1, 2, 4, 8, 16, 32 and 64 and the results are as follows:



The plot of execution time vs the number of processors for p = 1, 2, 4, 8, 16, 32, 64:

Shape

Description automatically generated

The plot of Speedup vs the number of processors for p = 1, 2, 4, 8, 16, 32, 64:

Chart, line chart

Description automatically generated

The plot of Efficiency vs the number of processors for p = 1, 2, 4, 8, 16, 32, 64:

Chart, line chart

Description automatically generated

1. (10 points) Modify the code to sort the list in descending order. Submit the modified code as qsort\_hypercube\_descending.cpp. 2 points will be awarded for each of the tests in Problem 1 that are executed successfully. (Note that the check\_list routine needs to be modified to verify descending order.)

**Solution:**

The code in qsort\_hypercube.CPP is modified to sort a list in descending order and the updated code is saved in qsort\_hypercube\_descending.CPP. The code compiled successfully and the output to the commands in problem 1 are as follows

Text

Description automatically generated