MPI Assignment

1 Introduction

In this assignment, you will construct distributed solutions to certain problems and implement them using the Message Passing Interface: MPI in any programming language of your choice (C/C++ is strongly recommended owing to the amount of documentation available online).

- Documentation: https://rookiehpc.github.io/mpi/docs/index.html
- Tutorial: `https://rookiehpc.github.io/mpi/exercises/index.html`

2 Instructions

- Answer all the questions.
- You may only use MPI library routines for communicating and coordinating between the MPI processes. You cannot use any form of shared-memory parallelism. The idea is to develop a program that could be deployed on a large message-passing system.
- Any language can be used for all of the following questions. **We recommend C/C++**.
- The final evaluation will be done on Linux, so please ensure that you don't use any platform specific libraries.
- The code you submit will be run alongside a Profiler to check if you are parallelizing the code and not submitting a sequential solution.
- Input must be read by **only one of your spawned processes** from a file. You may then choose to transfer this data between nodes as you wish. The absolute path of this file will be provided as a command line argument to your executable. The output must be printed to standard output.
- You should test your program with varying number of processes and a sequential program across various input sizes and tabulate the run time reported along with any other observations.
- Your programs should execute with any number of processes between 1 and 12. For those who are not able to run with 12 processes, use the following command (for C++):
  `mpiexec -np 12 --use-hwthread-cpus --oversubscribe ./a.out`
- Prepare a report containing the solution approach, highlights of your program, and the results obtained and submit the report along with a README file, the source files, and datasets. Bundle all of them as a single zip/tar ball and submit via moodle.
- Any code that is found to be copied from any source will attract penalty.

3 Problems

3.1 Distributed K-Nearest Neighbours (16 points)

You are given:

- A set P of N points in a 2D plane, where each point P_i is represented by coordinates (x_i, y_i).
- A set Q of M query points, with each query point Q_j also represented by coordinates (x_j, y_j).
- An integer K, representing the number of nearest neighbours to find for each query point.

For each query point in Q, find the K nearest points from the set P based on Euclidean distance. The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) is defined as $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

It is guaranteed that there exists a unique set of exactly K points for each query. That is, if the output corresponding to Q_i is the set O_i, then

- $|O_i| = K$, and
- for all points $t = (a, b) \in P \setminus O_i$, $\text{dist}(t, Q_i) > \text{dist}(t, o) \, \forall \, o \in O_i$.

3.1.1 Input Format

- The first line contains three integers, N, M and K.
- The N lines contain two space-separated floating points numbers x_i and y_i (with at most 2 decimal places) where $P_i = (x_i, y_i)$.
- The following M lines contain two space-separated floating points numbers x_i and y_i (with at most 2 decimal places) where $Q_i = (x_i, y_i)$.

3.1.2 Output Format

For the ith query, print K lines of two space-separated integers each which denote the coordinates of the K nearest neighbours of the ith query point. Print the output in the same order as the queries. That is, the first K lines must be the output corresponding to Q_0, the next K lines correspond to Q_1, and so on (assuming zero-based indexing).

For each query, you may print its K lines in any order.

3.1.3 Example

Input

```
5 3 2
0.12 34
42 1.24
5 -78.4
21.72 -18.76
61.45 74.9
21.41 34.34
-1.47 63.3
41.3 -6
```

In this example, $N = 5$, $M = 3$, $K = 2$, $P = \{(0.12, 34), (42, 1.24), (5, -78.4), (21.72, -18.76), (61.45, 74.9)\}$ and $Q = \{(21.41, 34.34), (-1.47, 63.3), (41.3, -6)\}$.

Output

```
0.12 34
42 1.24
0.12 34
61.45 74.9
42 1.24
21.72 -18.76
```

Explanation:

- The two nearest points to $Q_0 = (21.41, 34.34)$ are $(0.12, 34)$ and $(42, 1.24)$.
- The two nearest points to Q_1 are $(0.12, 34)$ and $(61.45, 74.9)$.
- The two nearest points to Q_2 are $(42, 1.24)$ and $(21.72, -18.76)$.

3.2 Julia Set (16 points)

A Julia Set is a fractal formed by iterating over a complex function. Given a grid of complex numbers and a constant complex number c, determine how many iterations it takes for each complex number in the grid to escape a defined threshold.

The function for the Julia Set is defined as

$$z_{n+1} = z_n^2 + c,$$

where

- z_0 is the initial complex number from the grid, and
- c is a constant complex number provided as input.

For each point in the grid, count how many iterations it takes for the magnitude of z_n to exceed a threshold value of T. If the magnitude of the point does not exceed the threshold T within a maximum number of iterations K, then it is considered part of the Julia Set. The magnitude of z is defined as

$$\sqrt{z_{real}^2 + z_{img}^2}.$$

Write a program using MPI that computes the Julia set after K iterations for $N \times M$ points spaced uniformly in the region of the complex plane bounded by $\mathbf{-1.5 - 1.5i}, \mathbf{-1.5 + 1.5i}, \mathbf{1.5 + 1.5i}, \mathbf{1.5 - 1.5i}$. Threshold $\mathbf{T = 2}$. c is given part of the input, and the uniformly spaced points' complex coordinates are their respective z_0.

3.2.1 Input Format

The first line contains three integers N, M, and k, where $N \times M$ is the dimension of the grid and K is the number of iterations.

The second line contains two floating point numbers indicating the real and imaginary parts of c respectively.

3.2.2 Output Format

Print N lines of M elements ($N \times M$ grid) of ones and zeros with one (1) indicating the number is in the Julia set and zero (0) indicating it is not in the set after K iterations.

3.2.3 Constraints:

$N, M \geq 2$ and $N \times M \times K \leq 10^8$

$-2 \leq c_{\text{real}}, c_{\text{img}} \leq 2$

3.2.4 Example

Input

```
25 25 10
-0.75 0.25
```

Output

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0
0 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

For the point $grid[17][5]$ (1 based indexing), the z progress is given below. As we can see in iteration 10, its magnitude of z does not exceed 2, so it is marked as 1 (in red)

Iteration	**real part**	**imaginary part**	**magnitude of z**
0	-1.0	0.5	1.118033988749895
1	0.0	-0.75	0.75
2	-1.3125	0.25	1.3360973954019968
3	0.91015625	-0.40625	0.9967063067494167
4	-0.0866546630859375	-0.489501953125	0.49711285715391335
5	-0.9821031314786524	0.3348352536559105	1.0376132265690299
6	0.1024119137693571	-0.40768550228983713	0.42035183936699866
7	-0.9057192686954145	0.16649629498895233	0.9208954391955716
8	0.04260637744110829	-0.05159780507577982	0.06691514691975466
9	-0.750847030089984	0.2456032088836172	0.7899950625218356
10	-0.24654967361898028	-0.1188208799416679	0.2736880396943534

For the point $grid[11][7]$ (1 based indexing), the z progress is given below. As we can see in iteration 8, its magnitude of z exceeds 2, so it is marked as 0 (in blue)

Iteration	**real part**	**imaginary part**	**magnitude of z**
0	-0.75	-0.25	0.7905694150420949
1	-0.25	0.625	0.673145600891813
2	-1.078125	-0.0625	1.0799350747267171
3	0.408447265625	0.384765625	0.5611361287407717
4	-0.7312154173851013	0.5643129348754883	0.9236477007438794
5	-0.5337731018461191	-0.5752686364215833	0.7847596627661079
6	-0.7960202797959204	0.8641258489150717	1.1748879813027149
7	-0.863065196916818	-1.1257234000645253	1.4184973414081188
8	-1.2723716393237696	2.193145375901119	2.535511038901609

3.3 Prefix Sum (16 Points)

Given an array a for N integers, compute its prefix sum array. The prefix sum array p of a is an array of length N such that

$$p_i = \sum_{j=1}^{i} a_j.$$

Note that your program's time complexity (ignoring the time taken to send messages) should be $O(N/p)$ where p is the number of processes. $O(N)$ submissions will not be accepted.

Do not use `MPI_SCAN`

3.3.1 Input Format

- The first line of the input contains a single integer, N.
- The next line contains N space-separated floating points numbers with at most 2 decimal places where the i^{th} number is a_i.

3.3.2 Output Format

Print a single line containing N space-separated floating point numbers, where the i^{th} number is p_i.

3.3.3 Example

Input

```
10
7.26 9.21 6.70 4.40 4.40 9.42 -6.87 6.34 -4.25 4.82
```

In this example, $N = 10$ and $a = \{7.26, 9.21, 6.70, 4.40, 4.40, 9.42, -6.87, 6.34, -4.25, 4.82\}$

Output

```
7.26 16.47 23.17 27.57 31.97 41.39 34.52 40.86 36.61 41.43
```

Explanation:

- $p_0 = \sum_{i=0}^{0} a_i = 7.26$.
- $p_1 = \sum_{i=0}^{1} a_i = 7.26 + 9.21 = 16.47$.
- $p_2 = \sum_{i=0}^{2} a_i = 7.26 + 9.21 + 6.70 = 23.17$, and so on.

3.4 Inverse of a Matrix (24 points)

Let A be a non-singular square matrix of n rows and n columns. Find the inverse of A. You must partition A into its rows and store the partitions across the processes (Hint: use `MPI_Scatter`). You should use the row reduction method.

3.4.1 Input Format

- The first line of the input contains a single integer, N.
- The next N lines each contain N space-separated floating points numbers with at most 2 decimal places where the i^{th} number in the i^{th} line is A_{ij}.

3.4.2 Output Format

Print N lines of N floating point numbers each (with exactly 2 decimal places each) such that the the i^{th} number in the i^{th} line is A^{-1}_{ij}.

3.4.3 Example

Input

```
3
-2.39 -5.01 -1.32
3.12 3.77 -6.08
5.57 8.44 4.53
```

Output

```
0.98 0.17 0.51
-0.69 -0.05 -0.27
0.08 -0.11 0.09
```

The inverse of

$$A = \begin{bmatrix} -2.39 & -5.01 & -1.32 \\ 3.12 & 3.77 & -6.08 \\ 5.57 & 8.44 & 4.53 \end{bmatrix}$$

is

$$A^{-1} = \begin{bmatrix} -0.98 & 0.17 & 0.51 \\ -0.69 & -0.05 & -0.27 \\ 0.08 & -0.11 & 0.09 \end{bmatrix}.$$

3.5 Parallel Matrix Chain Multiplication Problem (28 points)

Given a sequence of N matrices: $A_1, A_2, \ldots, A_N$ with dimensions $d_0 \times d_1$, $d1 \times d2$, ..., $d_{(N-1)} \times d_N$ respectively, find the most efficient way to multiply these N matrices together, i.e., determine the optimal parenthesization that minimizes the total number of scalar multiplications needed to compute the product $A_1 A_2 \ldots A_N$.

The problem uses the associative property of matrix multiplication, which allows changing the order of multiplications but not the order of matrices. The goal is to minimize the total number of scalar multiplications, which depends on the order of matrix multiplications. For matrices of dimensions $p \times q$ and $q \times r$, the number of scalar multiplications needed is pqr.

3.5.1 Constraints

- $1 \leq N \leq 5 \times 10^3$
- $1 \leq d_i \leq 10^3 \, \forall \, i \in [0, N]$, $i \in \mathbb{Z}$

3.5.2 Input Format:

The first line of input contains one integer N, representing the number of matrices.

The second line contains $N + 1$ integers representing the d array. The dimensions of the i^{th} matrix(M_i) is $d_{i-1} \times d_i \; \forall 1 \leq i \leq N$

3.5.3 Output Format

One integer representing the minimum number of scalar multiplication needed to multiply the N matrices.

3.5.4 Example 1:

Input

```
4
1 2 3 4 3
```

Output

```
30
```

The minimum number of scalar multiplication is achieved by putting parenthesis in the following way $((M_1 M_2) M_3) M_4)$.

The minimum is $1 * 2 * 3 + 1 * 3 * 4 + 1 * 4 * 3 = 30$

Which is cost(M_1,M_2)+ cost(($M_1 M_2$),M_3) + cost((($M_1 M_2$)M_3), M_4), where cost(A,B), is the cost to multiply matrices A and B

3.5.5 Example 2:

Input

```
4
40 20 30 10 30
```

Output

```
26000
```

The minimum number of scalar multiplication is achieved by putting parenthesis in the following way $((M_1(M_2M_3))M_4)$.

The minimum is $20*30*10 + 40*20*10 + 40*10*30 = 26000$

Which is cost(M_2,M_3)+ cost(M_1,(M_2M_3)) + cost(M_1(M_2M_3), M_4), where cost(A,B), is the cost to multiply matrices A and B

4 Submission Format:

We'll be automating the checking, so make sure you deal with all the edge cases and comply with all the Input-Output specifications and the directory structure given in the submission guidelines below. Not following the given requirements would result in a heavy penalty.

Your submission is expected to be a **<TeamNumber>.tar.gz** file containing a directory with the same name as your Team number (ex: Team65) that holds the following files:

- A directory for each of the mentioned problems with the name: **<ProblemNumber>** containing one source file named **<ProblemNumber>** (e.g. 1.cpp, 1.py)
- You are also required to submit a report, named **Report.pdf** in the root directory of your project, with the following data for every problem you attempt:
  - The total time complexity of your approach
  - The total message complexity of your approach
  - The space requirements of your solution
  - The performance scaling as you went from 1 to 12 processes (use a large enough test case for this)

Example structure:

```
Team65
├── 1
│   └── 1.cpp
├── 2
│   └── 2.cpp
├── 3
│   └── 3.cpp
├── 4
│   └── 4.cpp
├── 5
│   └── 5.cpp
└── Report.pdf
```