Assignment 1: Message Passing Interface

Distributed Systems

January 13, 2023
Due: 11:55 PM, January 30 2023

1 Introduction

In this assignment you will construct parallel 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`
- References:
  - `https://pages.tacc.utexas.edu/~eijkhout/pcse/html/index.html`,
  - `http://condor.cc.ku.edu/~grobe/docs/intro-MPI-C.shtml`
- Tutorial: `https://rookiehpc.github.io/mpi/exercises/index.html`

2 Requirements

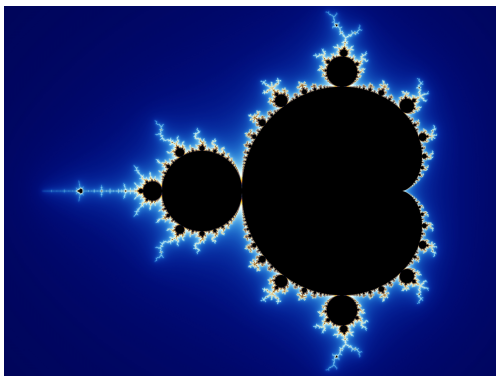
- You may only use MPI library routines for communicating and coordinating between the MPI processes. You cannot use any form of shared-memory parallelism. The idea is to develop a program that could be deployed on a large message-passing system
- We'll be doing the final evaluation on Linux, so make sure you don't use platform-specific libraries in your solutions. Please use standard libraries only.
- The code you submit will be run alongside a Profiler to check if you are parallelizing the code and not submitting a sequential solution.
- 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++):
  `mpirun -np 12 -\ -use-hwthread-cpus --oversubscribe ./a.out`

3 Problems

3.1 Problem 1: Mandelbrot Set (20 points)

The Mandelbrot set is a fractal that is defined as the set of all complex numbers c for which the sequence z does not diverge to infinity where z is defined as the sequence of complex numbers $\{z_0, z_1, z_2, ...\}$ such that $z_0 = 0$ and $z_{i+1} = z_i^2 + c$.

The magnitude of z is its distance from the origin. If the magnitude of z ever becomes greater than or equal to 2 its subsequent values will grow without bound and we know that c is not a point in the Mandelbrot Set. If we iterate K times and find that the magnitude of z_K is still less than 2, we can conclude c is a point in the Mandelbrot set.



You have to write a program using MPI that computes the Mandelbrot set after K iterations for $N * M$ points spaced uniformly in the region of the complex plane bounded by $-1.5 - i$, $-1.5 + i$, $1 + i$, $1 - i$ in a distributed manner.

Input:
The input contains only one line with three integers, the number of points along x axis: N and the number of points along y axis: M and the number of iterations: K
Output:
You are expected to output a Grid of $N * M$, ones and zeros with one indicating the number is in the Mandelbrot set and zero indicating if it s not after K iterations.

Constraints:
$N >= 2$ and $M >= 2$
$N * M * K <= 1000000$

Sample Input:
16 16 1000

Sample Output:
0 0 0 0 0 0 0 0 0 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 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 0 1 1 1 1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 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 1 0 0 0 0 0 0

3.2 Problem 2: Tony Stark and Pym Particles (40 points)

Tony Stark was using Pym Particles to find a way for the Avengers to travel through time and bring everyone back. Since Pym particles are scarce, he first wanted to do a simulation before building the final prototype. You are tasked to write a program using MPI that does this simulation in a parallel manner. The properties of the system are as follows:

- The Pym particles are isolated in a 2D grid of size $N * M$ points equally spaced apart, each point is connected by links to the neighboring 4 points. The bottom leftmost point is $(0, 0)$ and top right most point is $(N - 1, M - 1)$

- The simulation is done in discrete time steps. After each time step, the state at a given point can be determined by the state of the point itself and neighboring points, before the time step.
- Pym Particles can only travel in one of the four directions: up, down, left, right.
- In each time step one Pym particle moves in the direction of its velocity by one unit. A particle with an upwards velocity will after the time step maintain that velocity, but be moved to the neighboring site above the original site.
- The Pauli Exclusion principle prevents two particles with the same direction of movement to be at the same site. A site can contain at most one Pym particle for each direction(4 in total). Note that the exclusion principle does not prevent two particles from travelling on the same link in opposite directions, when this happens, the two particles pass each other without colliding.
- If two, and only two, particles collide head-on at a lattice point, for example a particle moving to the left meets a particle moving to the right, the outcome will be two particles leaving the site at right angles to the direction they came in. e.g. A particle at point $(3, 2)$ going R and a particle at point $(5, 2)$ going L, after one time-step would be together at $(4, 2)$ with their respective direction of movement the same and after one more time-step would be at $(4, 1)$ and $(4, 3)$ going D and U respectively.
- If a particle reaches the end of the lattice it bounces back. E.g After i time steps if there is a particle at point $(4, 0)$ with velocity in the D direction, it would be bounced back to the point $(4, 1)$ with a new velocity in the U direction after time step $i + 1$.

Given a 2D Grid of size $N * M$ and the initial positions of the Pym Particles and their directions of movement find the final positions of the Pym particles after T time steps.

Input:
The first line of the input contains four integers N, M - the size of the Grid, K the number of particles in the simulation and T the number of time steps.
The next K lines contain two integers i and j and a character c - describing the position of a Pym particle, $0 <= i < N$ and $0 <= j < M$, c can be one of $\{U, D, L, R\}$ for each of the directions. No two particles start at the same position with the same velocity.

Output:
You should output K lines. In each line output the final positions of all the particles x coordinate and y coordinate and the direction of movement $\{U, D, L, R\}$ after T time steps.

Constraints:
$1 <= N * M <= 1e6$
$1 <= K * T <= 1e6$

3.3 Problem 3: Jar of Words (40 points)

Jar is an employee at Google, owing to the buzz around ChatGPT, his manager tasked him to improve the existing search algorithms at google.

Given the all the search keywords, Google stores them in a Binary Search Tree to improve the lookup speed. Jar noticed that this Binary search Tree need not be optimal since different keywords would be queried at different frequencies.

The goal is to find the BST with the minimum expected search time, given a set of keys and their associated frequencies of being searched for. The expected search time for a sub-tree is calculated as the sum of the product of the frequency of searching for a given key and the depth of that key in the tree. The depth of the root node is taken to be 1.

But since this is google the data is so much that we need to develop a parallel algorithm to do this computation.

Given a set of N pairs of key and its search frequency count, Jar has to do the following things, implement merge sort to order the set based on the key. Use the sorted set to create an optimal binary search tree based on the frequency count such that the total average cost of all searches is minimized. All the keys are unique.

Input:
The first line of the input contains an integer N - the number of nodes in the BST.
The next N lines contain two integers u and v - describing a key and its frequency count. $1 <= u <= N$

Output:
In the first line you should output the final average cost of the most optimal BST. In the next line for each node

the key of the parent node in the final binary search tree. Output zero in case the node is the root node

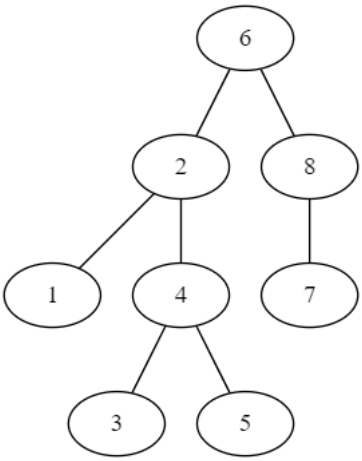
Constraints:
2 <= Number of Nodes <= 500
1 <= Frequency Count <= $1e7$

Example:

Sample Input:
8
5 7
3 6
7 5
6 17
1 16
4 8
2 13
8 28

Sample Output:
238
2 6 4 2 4 0 8 6

Explanation:
Here is the final optimal Binary Search Tree



4 Submission Instructions

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 **<RollNumber>.tar.gz** file containing a directory with the same name as your roll number 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 **README.md** 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