Chapter 10 - exercise 1

•Write a MPI program that is able to create a graph and print out all the edges at each vertex.
    •Unless you want to get into classes and multiple files I recommend using a simple struct to represent each vertex.
    •For now just have each vertex store a single integer.
    •For the edges C++ comes with a vector class: std::vector - cppreference.com
•Create a graph with 8 vertices and setup some edges between them (hard coding is fine).
    •Run it with 8 processors and send each processor the information for one of the vertices and have it print out the value stored and the edges there.

Grading Rubric
_____ (3 Points) Ability to represent a graph
_____ (3 Points) Ability to send vertex information to a specific processor

Answer:

Approach

1. Graph Representation:
- I defined a Graph class and a Vertex class to represent the vertices and edges.
- Each Vertex stores an integer value (the vertex ID) and a vector of integers to store the connected vertices (edges).
- I initialized the graph with 8 vertices, and each vertex has a set of edges connecting it to others in the graph. For simplicity, I hardcoded the edges between specific vertices.

2. MPI Parallelization:
- I initialized MPI with MPI_Init, allowing the parallelization of the program. The rank of each process determines which vertex it is responsible for.
- The MPI_Comm_rank function returns the rank of the process, and this rank is used to assign each processor a vertex from the graph.
- The root process sends the number of edges and the edges themselves to each process using MPI_Send.
- Each process receives the number of edges and the edges using MPI_Recv.
- Each processor prints the vertex's value and its edges.

3. Edge Setup:
- The edges between vertices are added using the add_edge function. This

function connects two vertices in a directed manner, meaning the edge is unidirectional.

4. Expected Output:
- Each process should print out the vertex it represents along with the vertices it is connected to. Since we are running the program with 8 processes, each process will handle one vertex and output its edges.

Code

```
----
#include
<mpi.h>
// Include the MPI library
#include
<iostream>
// Include the iostream library for input and output
#include
<vector>
// Include the vector library

using namespace
std;
// Use the standard namespace cause I don't want to type std::
before every cout, cin, vector, etc.

class Vertex
{
// Define the Vertex class
public:
    int
value;
// Value of the vertex
    vector<int>
edges;
// Edges of the vertex

    Vertex(int val) : value(val)
{}
// Constructor to initialize the vertex value
};

class Graph
{
// Define the Graph class
```

```
public:
    vector<Vertex>
vertices;
// Vertices of the graph

    Graph(int n)
{
// Constructor to initialize the graph with n vertices
        for (int i = 0; i < n; ++i)
{
// Iterate over the number of vertices

vertices.emplace_back(i);
// Add a new vertex to the graph
        }
    }

    void add_edge(int u, int v)
{
// Function to add an edge between two vertices

vertices[u].edges.push_back(v);
// Add edge from vertex u to vertex v
    }

    const Vertex& get_vertex(int i) const
{                                                                  /
/ Function to get a vertex by index
        return
vertices[i];
// Return the vertex at index i
    }

    void print_graph() const
{
// Function to print the graph
        cout << "Adjacency List for the Graph: " <<
endl;                                                  // Print
the header
        for (const auto& vertex : vertices)
{                                                                 //
Iterate over each vertex
            cout << "Vertex " << vertex.value << ":
";                                                     // Print
```

```
the vertex value
            for (int edge : vertex.edges)
{                                                                   /
/ Iterate over each edge of the vertex
                cout << edge << "
";
// Print the edge
            }
            cout <<
endl;
// Print a new line
        }
    }
};

int
main()
// Main function
{
    MPI_Init(NULL,
NULL);
// Initialize the MPI environment

    int
world_rank;
// Variable to store the rank of the process
    MPI_Comm_rank(MPI_COMM_WORLD,
&world_rank);
// Get the rank of the process

    int
world_size;
// Variable to store the number of processes
    MPI_Comm_size(MPI_COMM_WORLD,
&world_size);
// Get the number of processes

    int n =
8;
// Number of vertices
    Graph
g(n);
// Create a graph with 8 vertices
```

```
    if (world_rank == 0)
{
// Only the root process initializes the graph
        g.add_edge(0,
1);
// Add edge between vertex 0 and 1
        g.add_edge(0,
2);
// Add edge between vertex 0 and 2
        g.add_edge(0,
3);
// Add edge between vertex 0 and 3
        g.add_edge(1,
4);
// Add edge between vertex 1 and 4
        g.add_edge(1,
5);
// Add edge between vertex 1 and 5
        g.add_edge(2,
6);
// Add edge between vertex 2 and 6
        g.add_edge(2,
7);
// Add edge between vertex 2 and 7

        for (int i = 1; i < world_size; ++i)
{                                                       //
Send vertex information to each process
            int num_edges =
g.get_vertex(i).edges.size();
// Get the number of edges for the vertex
            MPI_Send(&num_edges, 1, MPI_INT, i, 0,
MPI_COMM_WORLD);                                        // Send
the number of edges to the process
            MPI_Send(g.get_vertex(i).edges.data(), num_edges,
MPI_INT, i, 0, MPI_COMM_WORLD);           // Send the edges to
the process
        }
    } else
{
// Other processes receive the vertex information
        int
num_edges;
```

```
// Variable to store the number of edges
        MPI_Recv(&num_edges, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);                     // Receive the number
of edges from the root process

g.vertices[world_rank].edges.resize(num_edges);
// Resize the edges vector to hold the edges
        MPI_Recv(g.vertices[world_rank].edges.data(),
num_edges, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE); // Receive the edges from the root process
    }

    // Each process prints its vertex and edges
    if (world_rank < n)
{
// Check if the process rank is less than the number of
vertices
        const Vertex& v =
g.get_vertex(world_rank);
// Get the vertex for the process rank
        cout << "Process " << world_rank << " has vertex " <<
v.value << " with edges: ";               // Print the vertex
value
        for (int edge : v.edges)
{
// Iterate over each edge of the vertex
            cout << edge << "
";
// Print the edge
        }
        cout <<
endl;
// Print a new line
    }


MPI_Finalize();
// Finalize the MPI environment
    return
0;
// Indicate successful execution
}
```

Sample Output

When running the program with 8 processes (one for each vertex), the output might look like this:

** I reordered it by the process id so its easier to read

Output reordered:

```
sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
3 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 1 has vertex 1 with edges: 4 5
Process 2 has vertex 2 with edges: 6 7

sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
4 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 1 has vertex 1 with edges: 4 5
Process 2 has vertex 2 with edges: 6 7
Process 3 has vertex 3 with edges:

sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
8 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 1 has vertex 1 with edges: 4 5
Process 2 has vertex 2 with edges: 6 7
Process 3 has vertex 3 with edges:
Process 4 has vertex 4 with edges:
Process 5 has vertex 5 with edges:
Process 6 has vertex 6 with edges:
Process 7 has vertex 7 with edges:
```

The actual output :

```
sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
8 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 1 has vertex 1 with edges: 4 5
Process 3 has vertex 3 with edges:
Process 5 has vertex 5 with edges:
Process 4 has vertex 4 with edges:
Process 6 has vertex 6 with edges:
Process 2 has vertex 2 with edges: 6 7
```

```
Process 7 has vertex 7 with edges:
sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
4 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 2 has vertex 2 with edges: 6 7
Process 1 has vertex 1 with edges: 4 5
Process 3 has vertex 3 with edges:
sajjadalsaffar@Sajjads-MacBook-Air-4 chapter 10 % mpirun -np
3 ./exercise1
Process 0 has vertex 0 with edges: 1 2 3
Process 1 has vertex 1 with edges: 4 5
Process 2 has vertex 2 with edges: 6 7
```

Each process prints the value of the vertex it is handling, along with the list of connected vertices (edges). The order of the output might vary depending on how the MPI processes execute, but each process will correctly print its vertex and edges.

Conclusion:

In conclusion, this MPI program successfully models a graph with 8 vertices and parallelizes the task of printing out the adjacency list for each vertex. Each MPI process is assigned a specific vertex, and it prints that vertex’s value and its corresponding edges.