Assignment 2: Parallel Graph Centrality Computation Using MPI

COL380

March 7, 2025

Due Dates

- **Due Date 1:** 11:59 PM, March 12, 2025
- **Due Date 2:** 11:59 PM, March 17, 2025

Objective

In this assignment, you are given an undirected graph representing social connections in a University, where nodes are individuals, undirected edges represent relationships, and colors represent the houses they belong to. Each node has a color represented by a number. Your task is to compute partial degree centrality measures in parallel using MPI and then use this score to identify the top k most influential nodes for each color.
Degree Centrality: This measure indicates how connected a node is within the graph. Unlike the standard degree centrality, in this problem, only connections to nodes of specified colors are counted. For a node $'u'$ and color $'c'$, the degree centrality is computed as:

$$\text{Degree Centrality}(u, c) = \text{count of } v \text{ ,where } v \in \text{neighbors of } u \text{ and color}(v) = c$$

Remember, color(u) need not be c. The graph is undirected and u is a neighbor of v if there is an edge between u and v.

After computing the centrality scores for each color, you will find the top k most influential nodes in the graph for each color. (k $\ll$ |v| and c $\ll$ |v|, where |v| the number of vertices in the graph)

Problem Statement

You will define the 3 functions as follows:

Function 1:

```
vector<vector<int>> degree_cen(vector<pair<int,int>> partial_edge_list,
        map<int,int> partial_vector_color,int k)
```

Inputs:

- *partial_edge_list* : A list of outgoing edges from a subset of vertices.
- *partial_vector_color* : A mapping of vertices in the subset to their corresponding colors.
- k : The number of nodes to be returned for each color.

Outputs:

- A vector of vectors containing the top k nodes in the given graph based on their color degree centrality for each color. You will have to find the number of distinct colors for this. each color is represented by a non-negative integer (zero or any positive integer). You will only consider the color if at least one node of that color exists. The color code need not be consecutive. Example c = 3 might include color codes [0, 2, 3].
- The order of vectors should be in lexicographic order of colors, i.e., in the above case of [0,2,3] the first vector corresponds to color 0, the second to color 2, and the last to color 3.

Instructions:

1. Compute the color degree centrality for each node u for each color c, which is the number of edges connecting u to nodes of the color c. Use MPI for parallel processing as all the processes only have a part of the graph information, not the whole graph.
2. For each color i (where i is the color code of at least one vertex), select the top k nodes with the highest color degree centrality. In the case of ties, the node with the lexicographically smaller label is ranked higher.
3. Ensure that the resulting vectors are ordered lexicographically by color.
4. Optimize the function for performance, as runtime will be measured only for this function.
5. You need to send the final results to the rank 0 process as we will only consider the rank 0 returned value as the final output. All other rank outputs will be ignored.

Function 2:

```
void init_mpi(int argc, char* argv[])
```

This function should initialize MPI. The driver code won't directly initialize MPI and will call this function instead at the beginning and assume the initialization is done inside this function. MPI initialization uses *argc* and *argv* which will be passed directly.

Function 3:

```
void end_mpi()
```

This function should finalize the MPI. We will call this at the end of the code.

Solving an Example Graph

Let's manually compute the color degree centrality, and find the top k nodes for each color for the following graph:

Graph:

- Nodes: $\{1_1, 2_1, 3_2, 4_1, 5_2\}$ (The subscript represent the color codes.)
- Edges: $\{(1,2), (1,3), (2,3), (2,4), (3,4), (3,5), (4,5)\}$

Step 1: Degree Centrality

For each node, we count the degree centrality for each color:

$$\begin{aligned}
&\text{Degree Centrality}(1,1) = 1(\text{to } 2)\\
&\text{Degree Centrality}(2,1) = 2(\text{to } 1,4)\\
&\text{Degree Centrality}(3,1) = 3(\text{to } 1,2,4)\\
&\text{Degree Centrality}(4,1) = 1(\text{to } 2)\\
&\text{Degree Centrality}(5,1) = 1(\text{to } 4)
\end{aligned}$$

$$\begin{aligned}
&\text{Degree Centrality}(1,2) = 1(\text{to } 3)\\
&\text{Degree Centrality}(2,2) = 1(\text{to } 3)\\
&\text{Degree Centrality}(3,2) = 1(\text{to } 5)\\
&\text{Degree Centrality}(4,2) = 2(\text{to } 3,5)\\
&\text{Degree Centrality}(5,2) = 1(\text{to } 3)
\end{aligned}$$

Step 3: Returning top k

Let's say $k = 2$: You will return [[3,2], [4,1]] (Note the order)

Evaluation Criteria

- **Correctness**: The accuracy of the computed centrality measures and top k nodes.
- **Performance**: Efficiency and scalability of the parallel implementation with different numbers of nodes (up to 6 or 8) and processes per nodes (up to 6 or 8).
- **Report**: Clarity and depth of performance analysis.

Starter Code

The starter code is provided here.
You are given a "check.cpp" file, a "template.hpp" file, and a "template.cpp" file. You are not allowed to change the "check.cpp" and "template.hpp" files. You have to implement your functions in the "template.cpp" file, without changing the signature of the functions.
"check.cpp" has the read graph file code and write to output file code. It also calls the three functions implemented in "template.cpp".
You are also given sample input folders and the corresponding output.txt files. Read file and write output file functions are already present in the "check.cpp" file. This "TestCase1" case is for V = 1000, E = 20000, k = 10, and C = 5. And SmallTestCase is V = 40, E = 160, K = 10, C = 2. We will test on much larger test cases.

Report

Your report should list your optimizations done and a discussion on performance in a text file named `readme.pdf`, and a CSV file showing your timings for degree_cen function across different values of k on varying numbers of cores.

Format for Final Scalability Analysis: The performance table records timings (in milliseconds, up to two decimal points) for varying the number of nodes (Use 2 processes per node for the analysis) for the test cases given by varying k and other test cases you might use (if any), in CSV format. Note the timings for all the optimizations you mentioned in the report. The last rows should correspond to your final submission. Below is the formatted version for clarity:

Optimizations	V	E	C	k	1 node	2 nodes	3 nodes	4 nodes
$O1$	$V1$	$E1$	$C1$	$k1$				
$O1$	$V1$	$E1$	$C1$	$k2$				
$O2$	$V1$	$E1$	$C1$	$k1$				
$O2$	$V1$	$E1$	$C1$	$k2$				

Here V = Number of vertices, E = Number of Edges in the given graph, C = Number of distinct colors, and k = Number of top nodes to return.

Execution Instructions

Module Requirements

Before running the code, you must load **ONLY** the following modules:

1. `module load compiler/gcc/9.1.0`
2. `module load compiler/gcc/9.1/mpich/3.3.1`

Run `module purge` before loading any modules so that just the two modules are loaded.

Running the Code

To Compile:

```
mpic++ -o check check.cpp template.cpp
```

To run:

```
mpirun -np (Number_of_cores) ./check (Input_folder_name) (Output_file_name) (k)
```

Example: mpirun -np 4 ./check TestCase1 output.txt 10
Do not run this directly on the login node. You may use a PBS script.

Submission Instructions

Submit the following files on Gradescope:

```
|-- template.cpp
|-- readme.pdf
|-- data.csv
|-- Makefile
```

Note that if you submit more or less than these 4 files in the final submission, you will get a 0. Please stick to the given file names. In the make file mention the compile statement so that when we run "make compile", we get an executable named check. This is done so that you can add your own optimization flags to the compile statement.
The first submission is for correctness check, so only submit "template.cpp" file (Only 1 file).