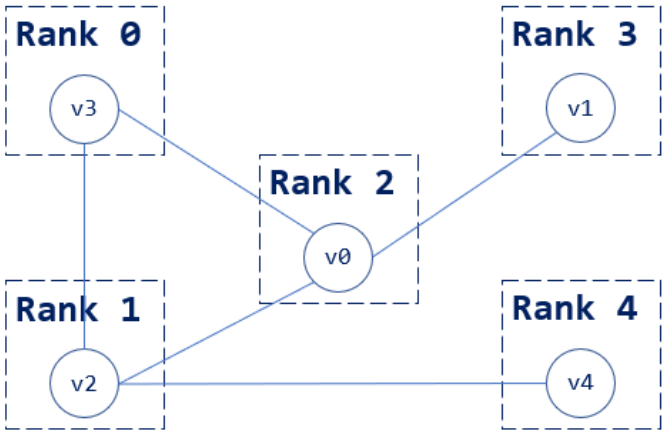
CSE531 - Programming Lab 2: MPI

Due: March 23, 22:00.

In this assignment, we are going to implement an all-pair shortest path algorithm in MPI + OpenMP in a fully distributed manner.

Problem Description

A fully distributed programming model means that during the computation, each MPI process controls one (or more) vertex and does local computation (e.g., calculates the new shortest distance of its own vertices) and local communication (i.e., only sends messages to its outgoing neighbors). Specifically, we partition a graph into subgraphs and assign each subgraph to a process. A process creates a set of threads and lets each thread handle a vertex's computation. Thus, the MPI processes are only responsible for communication, and threads are doing the actual computation work.



For example, each rank owns one vertex. Rank 2 can only communicate with rank 0, 1, and 3 but cannot communicate with rank 4 since there's no edge between the two vertices (subgraph). Similarly, rank 4 can only communicate rank 1 in this example.

One way to implement an all-pair shortest path algorithm fully distributedly is to apply Dijkstra's algorithm (a single-pair shortest path algorithm) on each vertex. The following shows the pseudo-code for Dijkstra's algorithm.

```
 1  function Dijkstra(Graph, source):
 2
 3      for each vertex v in Graph.Vertices:
 4          dist[v] ← INFINITY
 5          prev[v] ← UNDEFINED
 6          add v to Q
 7      dist[source] ← 0
 8
 9      while Q is not empty:
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
18
19      return dist[], prev[]
```

Note that if your implementation is based on the Floyd–Warshall algorithm, make sure you cannot initialize the cost from the node that is not directly connected.

Input / Output Format

1. The following files are provided on Canvas:

```
seq.cc      # DONT MODIFY: sequential version
apsp.cc     # TODO: a template for your MPI+OpenMPI implementation
Makefile    # TODO: a template for your Makefile

# input: V-E.in; output: V-E.out
50-1225.in
50-1225.out
100-4000.in
100-4000.out
500-40000.in
500-40000.out
1000-400000.in
1000-400000.out
2000-1200000.in
2000-1200000.out
```

2. The program accepts 2 parameters: ./executable $input_file $output_file.

```
$ mpirun -np $N -hostfile $hostfile ./apsp 100-4000.in 100-4000.my.out
$ diff 100-4000.my.out 100-4000.out
```

3. Input:
   The first line of an input test case consists of 2 integers V and E separated by a single space, which represents the number of vertices and the number of edges of this graph. Each of the following E lines consists of 3 integers i, j ($i \neq j$), and W, separated by a single space between any two numbers. It means that the distance between vertex i and vertex j is W.

Notice that:

- $1 \leq V \leq 2000$
- $V - 1 \leq E \leq 4000000$
- $0 \leq i, j < V$
- $0 \leq W \leq 100$

4. Output:
   For the output file, list the shortest-path distance of all vertex pairs. Let V represent the total number of vertices. The output file should consist of V lines, each consisting of V numbers and separated by a single space. The number at the ith line and jth column is the shortest-path distance from the ith vertex to the jth vertex.

Notice that:

- The output records must be sorted by the vertex id.
- Distance(i, j) = 0, where i = j

Report

The report must contain the following:

1. Title, name, PSU ID
2. Explain your implementations in the following aspects:
    - What algorithm did you choose to implement? Why?
    - Efforts you've made in your program
3. Experiment & Analysis
    - System & compiler spec (e.g., e5-cse-135-01~04 GCC 4.8.5)
    - Show the correctness of **all** datasets. Specifically, show that your program can work when the number of vertices is not divisible by num_process * num_threads.
    - Scalability plots using the 500-40000.in dataset
        - using 1~2 nodes (with all threads=cores in each node) on **gold6348 on DevCloud**
        - using 1~4 nodes (with all threads=cores in each node) on **Lab135 machines**
    - Each plot must contain at least 4 data points, and make sure your plots are properly labeled and formatted.
    - Discussion based on plots.
    - Any other discussions or analyses are encouraged. Make sure to explain how and why you do these experiments.

Rubrics

1. Correctness (50%)
    - 5 datasets (each 10%)
    - Your implementation should output the correct result
    - Your implementation should follow the restriction of the communication patterns.
2. Performance (15%): Based on the fastest version using **four Lab135 machines under the 500-40000.in dataset** among all students.
3. Report (35%)

Submission

Upload these files to Canvas:
Please do not upload any dataset!
Any corrupted files will be regarded as a failure of submission.

```
Makefile
apsp.cc
Lab2_Report.pdf
```

Reminders

1. Since we have limited resources, please start your work ASAP. Do not leave it until the last day!
2. Copying any codes from the Internet is not allowed, but discussions are encouraged.
3. Office hour holding by Scott: Tuesday 15:00-16:00 @ Westgate Bldg W341