CITS3402 Assignment 2 2019: Shortest Paths

Nicholas Pritchard

September 2019

1 Details

- Due date: Fri 25/10 11:59pm
- Worth: 25%
- You are allowed to undertake this assignment as a pair or individually

The goal of this project is to design an implement parallel algorithms to solve the all-pairs-shortest path problem for a number of large graphs using either Dijkstra's or Floyd-Warshall algorithms. You will use `C` with the Message-Passing Interface (MPI) to write parallelized code. You will also need to observe and comment on how speedup is affected by changing the size of the problem and the number of nodes you use. A reasonable range for the size of your problems is 1024 to 4096 vertices. A reasonable range for the number of processors is one to 16.
Your submission will include:

1.1 Code

A copy of all source files, job scripts and build-files used. Source files must include your names and student numbers at the top. Your program must compile, execute correctly and be well documented. Make sure to include comments to explain any MPI function the first time it is used.

1.2 Report

Alongside your code you will also need to write a report addressing the following points:

- How your approach partitions data and processing
- A table showing speedup for graphs of 2048 vertices vs $1, 2, 4, 8, 16$ processors
- A table showing speedup for 4 processors vs graphs of $256, 512, 1024, 2048, 4096$ vertices.

When discussing the performance of your approach, try to relate performance to various factors in your computation (number of compute nodes, load balancing, communication overhead etc.)

2 Code Requirements

File Reading

Your solution should read a matrix description from file. It is acceptable to implement this with or without flags, for example:

```
mpirun -n 4 ./my_solution -f file.in
```

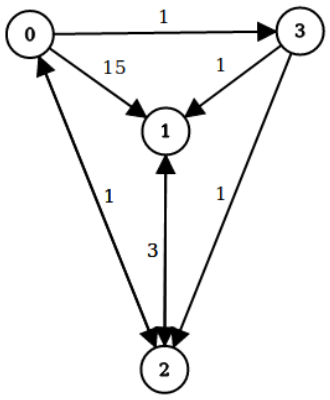
is acceptable as is

```
mpirun -n 4 ./my_solution file.in
```

The input file format is essentially a flattened adjacency matrix for a directed, weighted graph. All weights are positive integers. These are binary files (not text files) and will need to be read in as such. More specifically, each file contains:

- The number of vertices (`numV`)
- `numV x numV` integers specifying the weights for all connections

You can read all the information needed by reading the first integer to determine how many vertices there are in the graph, then read `numV x numV` integers into a freshly allocated array. For example the graph (assuming nodes are labelled from 0 upwards)



corresponds to the following adjacency matrix

$$\begin{pmatrix} 0 & 15 & 1 & 1 \\ 0 & 0 & 3 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{1}$$

which would be specified by the file `example.in`

```
4 0 15 1 1 0 0 3 0 1 3 0 0 0 1 1 0
```

Being a binary file however, you would not be able to 'read' the files yourself as such. We have added an `.txt` files for all of our example inputs if you would like to read them yourself to make sure you are correct.

File Writing

The output of either algorithm should be a matrix containing the length of the shortest path from vertex i to vertex j. This file has the same form as the input files. That is, a single array length `numV*numV + 1` where the first integer is the number of vertices in the graph. The solution for the `example.in` file would be `example.out`

```
4 0 2 1 1 4 0 3 5 1 3 0 2 2 1 1 0
```

You can choose to write this file by default, or enable this feature with command line arguments. In any case, specify your choice at the top of your code file.

3 Hints

- When timing your program for speedup analysis do not count the time taken for file operations.
- Passing command line arguments to MPI programs can be a little tricky compared to standard programs, be careful.
- Dijkstra's algorithm is made very concise by using a priority queue. Since C has no default data structure library you are not *required* to implement one. A simpler implementation will be accepted.
- A hint for parallelizing Dijkstra's is that it is a single-source sortest path algorithm
- A hint for parallelizing the Floyd-Warshall algorithm is that it can be viewed as iterating over 2D distance matrix.

Marking Rubric

	Critera	Highly Satisfactory	Satisfactory	Unsatisfactory
File Reading (5)	Program is able to read input files as specified.	Reads in adjacency files correctly.	Attempts to read adjacency files.	No attempt is made to read adjacency files.
File Writing (5)	Program is able to write output files as specified.	Write output files correctly and efficiently.	Attempts to write output files but are either poorly formatted or written inefficiently.	No attempt is made to write output files.
Dijkstra's Algorithm or Floyd Warshall (60)	Dijkstra's all-pairs shortest path or the Floyd-Warshall algorithm is implemented correctly in a parallel fashion. MPI calls are explained.	Algorithm is implemented correctly and efficiently. Comments indicate an understanding of the parallelism present.	Algorithm is either incorrect for some cases or inefficient. Parallelism is attempted but not efficiently.	Algorithm is implemented incorrectly for all cases or missing. Parallelism is not attempted.
Report (30)	Addresses all required points and is presented clearly. Testing is thorough and clear.	Report addresses all points effectively and is presented clearly. Performance testing is thorough and meaningful.	Report does not address all points required or some minor readiblity issues.	Report addresses few points or is very difficult to read.

A Graphs

Graphs and graph theory provides a natural way to model many problems. Power-grids, transport systems, social structures and nervous systems can all be modelled by large graphs for example. For completeness we provide a definition of a graph below:

- A ***graph*** G is specified by two sets V and E
- V is a finite set of points called vertices
- E is a finite set of *edges*
- An ***edge*** e is an ordered pair of two vertices (u, v) indicating that vertex u is connected to vertex v.
- Edges can be weighted or un-weighted. For our assignment, edges are weighted by positive integers. This forms an extra set of weights w to go along with our vertices and edges.
- Graphs can be directed or un-directed (for every (u, v) there exists an edge (v, u))
- Vertices u and v are ***adjacent*** to each other if there is an edge (u, v) or (v, u) present
- A ***path*** from vertex u to vertex v is a sequence of vertices following edges that exist in our graph.
- The ***shortest-path*** from vertex u to vertex v in a weighted graph is a path with minimum sum-weight

B All-Pairs Shortest Paths

For a weighted graph, the ***all-pairs shortest paths*** problem is to find the shortest path between all pairs of vertices.

C Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest paths from a starting vertex s to all other vertices in the graph. It incrementally selects edges that appear closest to a vertex to build a set of paths, updating the length of a path if a superior option is found. For more reading, you can start with 1, 2 or 3. An *all-pairs* shortest path implementation is found by running Dijkstra's for each possible source.

D Floyd-Warshall Algorithm

The Floyd-Warshall algorithm is an all-pairs shortest path algorithm which considers the shortest paths possible over zero 'hops' upwards. Trivially a path of a single hop is the adjacency of all vertices. The computation of all subsequent paths is based on the following observation.

For three vertices i, j, k the shortest path from i to j either includes the paths i to k plus k to j or it does not. This boils down to exceedingly concise pseudo-code. For more reading consult 1 2 3