CSE 6010 Assignment 4
Parallel Graph Generation and Depth First Search

Due Dates:

- Due: 11:00 AM, Friday, October 27, 2017
- Revision Due (optional): 11:55 PM, Monday October 30, 2017
- No late submissions will be accepted

This assignment involves an introduction to parallel computing and graph analytics. You will work on an individual basis for this assignment (no teams). Each student should complete all parts described below.

Graphs arise in many applications in science, engineering, and the social sciences. A graph consists of a set of vertices (nodes) and edges (links) where each edge connects two vertices. Here, we assume undirected edges and there are no edges from a vertex to itself. A *connected component* (or just component) is defined as a subgraph of the original graph where there is a path between every pair of vertices in the subgraph, and there are no edges between a vertex in the component and any vertex not in the component. Here, we are interested in the size of the largest component of a randomly generated graph, i.e., the number of vertices in the component containing the most vertices.

A second aspect of this assignment is to give a brief introduction to writing parallel code, i.e., software designed to execute on multiple processors in a computer. Here, you will use a software library called *OpenMP* to write your parallel code.

The assignment contains three main parts:

- Write a *parallel* program to generate a random graph and measure the performance of the graph generation program as the number of processors is varied.
- Augment the graph generation program with a sequential (single processor) depth-first-search program to determine the size of the largest connected component of the graph that was generated.
- Perform a sequence of experiments to evaluate the size of the largest component as a certain parameter used to generate the graph (discussed below) is varied, and compare your results with expected results reported in the published literature.

Each of these is described next.

1. Parallel Random Graph Generation

We are specifically interested in random graphs constructed using the Erdős-Rényi model (first introduced by Gilbert in 1959). This model has two parameters: N is the number of vertices in the graph, and P is the probability there is a link between any two vertices in the graph. P is the same for all pairs of vertices. N and P should be passed to the program as command line arguments. Create the graph in memory; do not write it to a file as this will affect execution time.

Your program should be designed for execution on multiple processors using OpenMP. You should come up with a strategy for parallel execution, and implement it using OpenMP primitives. Your code must be designed to work so the number of processors can be easily

changed without making significant changes to the code. You have access to a compute cluster called "flamel" to run your parallel code.

Once you have gotten your parallel program to run, you should run it with different numbers of processors (starting with 1 processor) and measure the time required to run your graph generation program. These measurements should consider the graph generation part only, i.e., without the analysis part described later. *Speedup (k)* is defined as the execution time of the program on a single processor divided by the execution time on *k* processors. Plot your measurements in a graph showing speedup as a function of k. Show speedup for a few different values of N. Vary k and N to enable you to characterize the general shape of the speedup curve (or until you are not able to run experiments with more processors) for different network sizes.

In addition to reporting the results of the performance measurements, generate sample outputs for some small graphs in order to argue that your program produces credible results for different values of P.

2. Depth First Search

The second part will involve implementing the depth-first search algorithm to determine the size of the largest component of the random graph. It is clear that if *P*, the probability there is a link between a pair of vertices is 0.0, the graph contains *N* unconnected vertices, and the largest component of this graph has size 1, independent of *N*. Further, if *P* is 1.0, there is an edge between every pair of vertices, and the largest component is the entire graph, so the largest component has *N* vertices, i.e., is proportional to the size of the network. For this part of the assignment your goal is to examine how the size of the largest component of these randomly generated graphs changes as *P* is increased from 0.0 to 1.0.

Implement the depth-first search (DFS) algorithm to determine the size of the largest component (not to be confused with breadth-first search, or BFS; although one could use BFS to determine the largest component, here you will need to implement DFS). DFS will identify all connected components of the graph. The program should print the size of the largest component.

You will need to generate sample output on some small graphs to verify that your program produces correct results. Document these experiments in your project report to provide evidence your code works correctly.

The DFS program can be a sequential implementation; although one could envision different ways to implement DFS in parallel, this is beyond the scope of the current assignment. Because the program is sequential, you are free to complete your experiments running both the graph generation code and DFS code sequentially on your own computer, although the code should be runnable on flamel so that the TAs can verify it executes correctly.

3. Analysis of Results

Plot the size of the largest component for different values of *P*. Use as large a value of *N* as your programs can process in a reasonable amount of time. Conduct a literature search to determine what results you should expect in terms of the size of the largest component as P is varied, and compare this with what you observe from your program. Discuss your results in the report, including citations to the literature as needed.

Finally, from your literature search identify an application where these computational results might be of interest. Discuss specifically how aspects of the application map to the network and connectivity results computed by your program, and why one might be interested in these computational results.

4. Final Comments

You are free to choose the data structure used to represent the graph but you should understand the tradeoffs involved and be prepared to justify your design choice. You should use the same data structure for both the graph generation and graph analysis programs.

Turn in a report including your results, and your software in a single zip file. Your software must be well documented and include comments so the code is easy to understand. You should include a README file with instructions on how to compile and run your program.

A reminder you must adhere to the Georgia Tech honor code, and the collaboration policy stated in the course syllabus. Specifically, you are encouraged to discuss the problem and possible solutions with other students (as well as the TA/instructor), however, **all code that you turn in must be completely your own work**. **Disseminating your code to other students is strictly prohibited**. Further, downloading code from the web or other sources other than examples provided in class is also prohibited.