CSCI-4974/6971: Assignment 5 (100 pts)

Distributed Triangle Counting

Due Date: Thursday 21 Nov. 2016, 16:00

For this assignment, we're going implement a distributed version of the triangle counting code we've been using as part of our hands-on work over the past couple weeks. We'll also implement a sampling approach utilizing this algorithm as a baseline. We'll be using the CCI Amos system and examining the effects of graph structure and sampling on a couple graphs.

1. Implement a distributed triangle counting algorithm
2. Create an approximate algorithm using uniform sampling
3. Examine how different graph structures affect the execution of the above

For background material and reference, use the lecture slides and our hands-on code (random along with the CCI user wiki:

- `https://secure.cci.rpi.edu/wiki/index.php/CCI_User_Wiki`
- `https://secure.cci.rpi.edu/wiki/index.php?title=Blue_Gene/Q`

Submit the code file to `slotag@rpi.edu` by the due day and time listed above. Put your responses to the short answer questions in comments in the code file.

1 Triangle Counting Implementation

Triangle counting and enumeration is a basic but widely-used graph analytic method. We've used serial triangle counting previously in class to calculate clustering coefficients of various graphs. Here, we want to implement a parallel version of that code. See the `cpp` file for implementation details. We'll be implementing both an exact version as well as a version that using only a small sample of vertices to output an approximate count.

You'll be running on the California road network (roadnet-ca) and Google web crawl (google) from the SNAP database. In the assignment tarball, you'll find these binary edge list graph files and partition assignments for 64 and 512 tasks. Run all four possible graph/part count combinations with a 10% sample. To run the code on 512 tasks with the sample-based algorithm using 10% of the vertices, you can call:

- `srun -o test.google.512 --partition debug --time 10 --ntasks 512 ./a.out google.ebin google.512 0.1`

You can verify the correctness of your code using the triangle count listed for roadnet-ca in SNAP (`http://snap.stanford.edu/data/index.html`). Note that the triangle count for google will be different than that listed in SNAP, since we create a naive undirected representation from the directed graph file (i.e. we end up creating a bunch of multi-edges).

Also Note: when testing locally, you probably will need to change the `BLUE_GENE` flag at the top of the file. This alters how the binary data file is handled, since the Blue Gene defaults to big-endian while your system is very likely little-endian.

2 Response Questions

In a single file, submit your code as well as the responses to the following questions:

1. When running using a 10% sample of vertices, how does the approximation error change with increasing task count for each graph? Can you explain your observations?
2. What do you see in terms of relative performance (execution time) for each test case? How does each graph scale? Give an explanation.
3. Take a look at the communication volume (amount of data that needs communicated) for each test case. Which test case results in the highest communication volume? Again, give a probable explanation.
4. The distributed algorithm described in the `cpp` file is just one approach. Describe another possible distributed algorithm.

This homework will be evaluated on successful implementation of the exact and approximate algorithms (30 pts each) and answers to response questions (10 pts each).