CSCI-4974/6971: Homework 4
<v1.0> updated April 10, 2024

Distributed Graph Processing

Due Date: Friday 24 April 2024, 11:59pm via Submitty
But accepted until 1 May.

For this assignment, we are going to modify some of the prior code we have worked with to allow for distributed parallel graph processing. To do this, we will be using MPI within Python via `mpi4py`. We will focus specifically on Breadth-first Search (BFS). Since the goal here is to observe computational differences with parallelization, we will be using one of our 'larger' datasets:

- p2p-Gnutella: `http://cs.rpi.edu/~slotag/classes/SP24m/hw/p2p-Gnutella31.data` (for testing)
- Congress Tweets : `http://cs.rpi.edu/~slotag/classes/SP24m/hw/congress.data` (for debugging)

We will be using Submitty for collecting homeworks. Upload a single *.py file that outputs responses for all of the below. There will be separate gradeables listed for the 4974/6971 sections. Pay careful attention to output formatting. Your code should be runable as a script on the command line (via `bash$ mpirun -n # python3 hw04.py`). **You can use any NetworkX/NumPy/SciPy functionality you wish, but do not use any other external libraries unless otherwise specified**. For this assignment, you will also need to install MPI4Py.

1. We will be implementing distributed BFS. The basic program flow is as follows. Please see the template file `hw04.py` for more detailed instructions. We will also be going through this homework in class in considerable detail.

    - **Phase 1**: Read in the dataset and select a root. We will read in our datasets in the same way we have in the past. Though, we will not be using the full graph for our processing - more details are in Phass 3 and 4.
    - **Phase 2:** Determine our parallel environment details. Specifically, each rank (i.e., process) needs to determine the total number of processors for communication and its specific process ID within the communicator.

- **Phase 3:** Next, we want to determine the "local" vertices for each rank. Simply cast `G.nodes()` as a list, and partition the list in `int(G.order() / nprocs` chunks, where `nprocs` is the total number of processors as determined in Phase 2. Note that `nprocs` might not evenly divide `G.nodes()`, so simply assign all the 'remainder' to the highest number rank.
- **Phase 4:** Here, we will create the local graph. This graph will include all of the locally-owned vertices and all of their incident edges. Note that this will include additional vertices in the graph which are not local. These are called the "ghost" vertices. For ghost vertices, we also need to know their rank for communications. The `get_rank()` function is included for this purpose.
- **Phase 5:** This will be where we run our distributed BFS. The primary modification will be to add communication functions. Each rank will have a local queue that will only contain local vertices. When a ghost vertex is discovered as a neighbor of a vertex in the queue, we want to communicate that vertex to the owning rank. We will do this by creating a list for each rank, appending the vertices to send to that list, and then eventually calling MPI send() and MPI recv() to do the communications.

Note in the template file where modifications are expected. Observe whether the timing results scale with an increasing number of ranks. In what circumstances would one expect scaling vs not expect it?