Assignment 3

- Qiaoyu Deng (qdeng@andrew.cmu.edu)
- Changkai Zhou (zchangka@andrew.cmu.edu)

Part 1

1.1 Pagerank

SCORES :	
grid1000x1000.graph	1.00000 / 1
soc-livejournal1_68m.graph	1.00000 / 1
com-orkut_117m.graph	2.00000 / 2
random_500m.graph	2.00000 / 2
rmat_200m.graph	2.00000 / 2
TOTAL	8.00000 / 8

1.2-1.4 BFS

SCORES:	**Top-Down**	**Bott-Up**	**Hybrid**
grid1000x1000.graph	2.00000 / 2	2.00000 / 2	2.00000 / 2
soc-livejournal1_68m.graph	2.00000 / 2	2.00000 / 2	2.00000 / 2
com-orkut_117m.graph	2.00000 / 2	2.00000 / 2	2.00000 / 2
random_500m.graph	4.00000 / 4	4.00000 / 4	4.00000 / 4
rmat_200m.graph	3.81906 / 4	4.00000 / 4	4.00000 / 4
TOTAL			41.81906 / 42

Part 2

2.1 Distributed Pagerank

for uniform_random we get :

> ref time: 0.15948
> student time: 0.05118
> Score: 20.00000

2.2 Distributed BFS

graph type	ref time	student time
uniform_random	2.10187	0.71535
grid	submit.sh error	submit.sh error
clustered	0.74040	0.31310

Discussion

Part 1-BFS(Q3)

- Synchronization happens in the process of updating new_frontier. Two ways to limit the overhead: 1. replace atomic operation with *compare&swap* method; 2. Allocate private space for each threads to store their private *new_frontier* and combine them at the end of loop in order to avoid atomic operations.
- Dynamically switch between the top-down and bottom-up BFS. We give a *threshold* of *frontier_count* and choose Bottom-Up if *frontier_count* > *threshold* or top-down if *frontier_count* < *threshold*. The reason is that if *frontier_count* is too large, the cost of vertices' iteration by top-town is too high, so we use bottom-up to balance it. The value of threashold is decided by best experiments result.
- Bottleneck of imperfect performance: communication/synchronization. Inevitably, we have to use atomic operation, even though we replace it with *compare&swap* methods.

Part 2-BFS(Q4)

We use the top down method as our basic implementation, because this method is naturally capabale to run in parallel, because when it knows the frontier in the local it can just check the frontier in local, do not need other information to perform computation. But bottom up method is not suitable for parallel, because every node subset needs to know all of frontier nodes in the graph. So, our implementation is divided into the following steps:

1. Start from root node, we let each machine to iterate its own local frontier(within the range between start_vertex and end_vertex), if it can find frontier node's adjacent node, then we go to step 2.
2. Check whether the new frontier nodes(which are found as adjacent nodes of last iteration's frontier nodes) comes from local or from remote. If it is in local, we can directly add them into new frontier, if it is from remote, we need to send those nodes to their "owner", this step we will sync with other machines.

3. If step 1 find no more new frontier, we need to tell other machines that it has no new frontier, if every machine in network confirms that, then we can end the computation, since there is no more change in the graph.

Q5

For BFS in part 2, we firstly tried hybrid method to implement but finally realize that the bottom up method require every machine in the network know the entire frontier set, which will introduce much more communication overhead, so actually hybrid method is not very good for running in parallel.