CMPT 431: Distributed Systems (Fall 2020)
Assignment 3 - Report

Name	Adam Spilchen
SFU ID	301325612

Instructions:

- This report is worth 45 points.
- Answer in the space provided. Answers spanning beyond 3 lines (11pt font) will lose points.
- Input graphs used are available at the following location.
  - live-journal graph (LJ graph): `/scratch/input_graphs/lj`
  - RMAT graph: `/scratch/input_graphs/rmat`
- All your answers must be based on the experiments conducted with 4 workers on slow nodes. Answers based on fast nodes and/or different numbers of workers will result in 0 points.
- All the times are in seconds.

---

1 [12 points] Run Triangle Counting with `--strategy=1` on the LJ graph and the RMAT graph. Update the thread statistics in the tables below. What is your observation on the difference in time taken by each thread for RMAT and that for LJ? Why does this happen?

Answer:

RMAT might have many vertices with edges that connect to itself (loops) causing the countTriangles function to return 0 immidiately (u == v). Meaning very little work is done for each vertex. Wheras LJ might have many edges between different vertices requiring alot of work to test. The edges are also much more evenly distributed in RMAT.

Triangle Counting on LJ: Total time = 53.31739 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	121189	42920131	339204160	53.31731
1	1211892	15515692	213398914	10.785881
2	1211892	7141449	84872316	3.353711
3	1211895	3416501	45558451	0.987469

Triangle Counting on RMAT: Total time = 3.95950 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	6249999	12650749	7	3.918032
1	6249999	12546666	5	3.934324
2	6249999	12399926	6	3.959172
3	6250002	12402659	9	3.751142

2 [9 points] Run Triangle Counting with `--strategy=2` on LJ graph. Update the thread statistics in the table below. Partitioning time is the time spent on task decomposition as required by `--strategy=2`.

What is your observation on the difference in time taken by each thread, and the difference in num_edges for each thread? Are they correlated (yes/no)? Why?

Answer:

Yes they are correlated. The number of edges better represents the amount of work for each thread because the thread has to check every edge for each vertex it is allocated. If one thread is allocated vertices with more threads, it will have more neighbors to check and take longer than one with less edges and the same number of vertices.

Triangle Counting on LJ: Partitioning time = 0.02212 seconds. Total time = 25.53987 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	325541	17248451	136034350	25.517400
1	510546	17248478	144912938	19.725595
2	904041	17248452	219730487	14.038579
3	3107443	17248392	182356111	7.996110

3 [9 points] Run PageRank with `--strategy=1` on LJ graph. Update the thread statistics in the table below. What is your observation on the difference in time taken by each thread, and the difference in num_edges for each thread? Is the work uniformly distributed across threads (yes/no)? Why?

Answer:

At first glance the number of edges doesnt appear to affect time_taken, since the edges are very unevenly distributed. However because of the barriers, the time will be bounded by the slowest thread (aka. the thread with the most amount of work) which is thread 0 in this case.

PageRank on LJ: Total time = 54.749566 seconds.

thread_id	num_vertices	num_edges	time_taken
0	24237840	858402620	54.749315
1	24237840	310313840	54.749281
2	24237840	142828980	54.748959
3	24237900	68330020	54.748901

4 [9 points] Run PageRank with `--strategy=1` on LJ graph. Obtain the cumulative time spent by each thread on **barrier1** and **barrier2** (refer pagerank pseudocode for program 3 on assignment webpage) and update the table below. What is your observation on the difference in barrier1_time for each thread and the difference in num_edges for each thread? Are they correlated (yes/no)? Why?

Answer:

All threads must wait roughly the same time at barrier one, however there is a significant difference at barrier two. From the previous table, you can see thread 0 processes many more edges than the rest of the threads. They are correlated, and the correlation must be that thread 0 takes much more time to complete the task before reaching barrier 2, making the rest wait. Once thread 0 reaches the barrier all threads are released, which is why thread 0 spends very little time waiting compared to the rest.
PS. I did not feel it was neccesary to re run the same code on the same graph twice. Nor did I feel it neccesary to copy the same data between each table. Refer to the table in the previous question for the missing details.

PageRank on LJ: Total time = seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0			0.007016	0.000672	
1			0.002333	31.720628	
2			0.006105	43.412299	
3			0.005379	49.339952	

5 [6 points] Run PageRank with `--strategy=2` on the LJ graph. Update the thread statistics in the table below. Update the time taken for task decomposition as required by `--strategy=2`. What is your observation on barrier2_time compared to the barrier2_time in question 4 above? Why are they same/different?

Answer:

The wait times are much more consistant between threads. This must be because the edges are again a better representation of the work for each thread. More edges means more neighbors means more work checking each neighbor. You can even see that thread 3 has a longer wait time in barrier 2 and has approx. 2000 less edges than any other edge. So it must consistantly be the first to reach the barrier and has to wait longer for the others.

PageRank on LJ: Total time = 28.956516 seconds. Partitioning time = 0.023214 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	6510820	344969020	1.741575	0.000734	28.842748
1	10210920	344969560	1.617526	0.377163	28.849311
2	18080820	344969040	1.362072	0.440234	28.862263
3	62148860	344967840	0.000724	1.017192	28.932932