CMPT 431: Distributed Systems (Fall 2019)

Assignment 2 - Report

Name	Raghav Mittal
SFU ID	301307947 (rmittal)

Instructions:

- This report is worth 30 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/assignment2/input_graphs/lj`
  - RMAT graph: `/scratch/assignment2/input_graphs/rmat`
- All the experiments are conducted with 4 workers.
- All the times are in seconds.

---

1. [4 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:
   Time taken by each thread is about similar for RMAT as compared to LJ where it is different for all threads. The overall amount of task (work load) one thread performs depends on the number of edges that thread has to work on which is evenly distributed for RMAT and not for LJ.

   Triangle Counting on LJ: Total time = 56.67268 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1211893	42920143	357657374	56.672207
1	1211893	15515691	217447240	11.485793
2	1211893	7141445	86161675	3.539036
3	1211892	3416494	46058460	1.010525

Triangle Counting on RMAT: Total time = 3.18946 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	6250000	12650751	7	3.188929
1	6250000	12546670	5	3.144260
2	6250000	12399925	6	3.129221
3	6249999	12402654	9	3.118459

2. [3 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:

Num_edges for each thread is about the same but time_taken is different and they are **not** correlated. To compute the total number of triangles, a thread needs to look at out degree of starting vertex and in degree of ending vertex for each assigned edge and overall, it can be uneven among threads.

Triangle Counting on LJ: Partitioning time = 2.219700 seconds. Total time = 28.94663 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	0	17248444	144441858	26.726389
1	0	17248443	152103594	20.723394
2	0	17248443	225182660	14.837736
3	0	17248443	185596637	8.381360

3. [1 point] Run Triangle Counting with `--strategy=3` on LJ graph. Update the thread statistics in the table below.

Triangle Counting on LJ: Total time = 18.70998 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1210728	17239790	176983964	18.709268
1	1213200	17242215	176570808	18.709242
2	1210268	17286373	175663701	18.709219
3	1213375	17225395	178106276	18.703485

4. [3 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:

Time_taken is about the same for each thread whereas num_edges is not. No, the work is **not** evenly distributed as the overall load on one thread depends on the overall number of edges it is going to work on (summation of out_degree of all assigned vertices) which is different for each thread.

PageRank on LJ: Total time = 42.580079 seconds.

thread_id	num_vertices	num_edges	time_taken
0	24237860	858402860	42.579272
1	24237860	310313820	42.579216
2	24237860	142828900	42.579195
3	24237840	68329880	42.564700

5. [3 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 4 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:
   Both barrier1_time and num_edges are different for each thread. **Yes,** they are correlated: More edges -> Less barrier 1 time. The thread with lower number of edges will quickly finish its task and get stuck on barrier to wait for other threads to arrive and hence, increases its barrier time and vice-versa.

   PageRank on LJ: Total time = 42.580079 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	24237860	858402860	0.000685	0.035822	42.579272
1	24237860	310313820	22.230168	0.008968	42.579216
2	24237860	142828900	31.725027	0.030385	42.579195
3	24237840	68329880	37.303834	0.028775	42.564700

6. [3 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 5 above? Why are they same/different?

   Answer:
   Barrier_2 times are much lesser in strategy 1 (hence, **different**). Because of uneven number of vertices to each thread, the thread with lower number of vertices finishes its task (of 2nd ‘for’ loop) quickly and wait for others to arrive. This increases its barrier_2 time.

   PageRank on LJ: Total time = 26.881835 seconds. Partitioning time = 0.010645 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	6510800	344968340	1.352946	3.793021	26.870461
1	10210840	344968860	1.382880	3.537477	26.869807
2	18080860	344968640	0.191489	2.988543	26.870387

3	62148920	344969620	0.093763	0.000651	26.868646

7. [3 points] Run PageRank with `--strategy=3` on LJ graph. Update the thread statistics in the table below. What is your observation on barrier times compared to the barrier times in question 6 above? What is your observation on the time taken by each thread compared to time taken by each thread in question 6 above? Why are they same/different?

   Answer:

   Barrier times are lesser in Q7 vs Q6 since threads finish working on all the vertices together and they arrive on the barrier (almost) together. Time taken by each thread is more in Q7 vs Q6 since they are getting work on the fly. It increases wait time for each thread to get the next task (& hence, total time).

   PageRank on LJ: Total time = 67.589279 seconds. Partitioning time = 0.000000 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	24227804	344945654	0.000991	0.000817	67.588477
1	24240941	345129438	0.000991	0.000805	67.588434
2	24233572	344845339	0.000988	0.000794	67.588400
3	24249103	344955029	0.001011	0.000810	67.573214

8. [3 points] Run PageRank with `--strategy=3` on LJ graph. Obtain the total time spent by each thread in `getNextVertexToBeProcessed()` and update the table below. What is your observation on the time taken by `getNextVertexToBeProcessed()`? Why is it high/low?

   Answer:

   The times for getNextVertex seems to be pretty **high**. It is high since every thread is calling this function a lot of times (equal to the number of vertices it works on) and hence, this function becomes a hotspot and since this function is atomic (serial), it increases the getNextVertex time for each thread.

   PageRank on LJ: Total time = 67.589279 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24227804	344945654	0.000991	0.000817	7.564976	67.588477
1	24240941	345129438	0.000991	0.000805	7.565789	67.588434

2	24233572	344845339	0.000988	0.000794	7.565426	67.588400
3	24249103	344955029	0.001011	0.000810	7.578072	67.573214

9. [3 points] Run PageRank with `--strategy=3` on LJ graph with `--granularity=2000`. Update the thread statistics in the table below. What is your observation on the time taken by `getNextVertexToBeProcessed()`? Why is it high/low?

Answer:
The times for getNextVertex seems really **low** as the number of calls to this function by each thread has reduced by the factor of granularity. Each slave (thread) takes work of about 2000 vertices together in one call from this atomic function and hence, the contention is reduced on this function.

PageRank on LJ: Granularity = 2000. Total time = 26.690177 seconds. Partitioning time = 0.000000 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24239855	344178063	0.001982	0.001141	0.005288	26.689379
1	24232997	343851335	0.001857	0.000868	0.004861	26.689271
2	24237855	347941132	0.002183	0.001298	0.004932	26.689248
3	24240713	343904930	0.001971	0.001249	0.004845	26.674419

10. [4 points] While --strategy=3 with --granularity=2000 performs best across all of our parallel PageRank attempts, it doesn't give much performance benefits over our serial program (might give worse performance on certain inputs). Why is this the case? How can the parallel solution be improved further to gain more performance benefits over serial PageRank?

Answer:
Multiple threads trying to modify the same memory location introduces wait which makes this process slow as compared to collision free serial execution. In order to improve it, we can find minimum edge cut partition of the graph (increase locality) that would reduce the shared updates among threads.