CMPT 431: Distributed Systems (Fall 2023)

Assignment 2 - Report

Name	Lingjie Li
SFU ID	301417109

Instructions:

- This report is worth 65 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 using the **slow slurm partition**. Answers based on fast nodes and/or different numbers of workers will result in 0 points.
- All your answers for PageRank must be based on the experiments using **20 iterations**. Answers based on different configurations will result in 0 points.
- All the times are in seconds.

---

1. [8 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:
   The time per thread remains consistent for RMAT graphs, as edges and triangles are evenly distributed across threads. In contrast, LJ graphs exhibit significant variation in thread-specific time due to the massive discrepancies in edge and triangle counts, despite having an even distribution of vertices.

   Triangle Counting on LJ: Total time = 60.025310 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1211892	42920131	339204160	60.024747
1	1211892	15515692	213398914	13.371380
2	1211892	7141449	84872361	4.230088
3	1211895	3416501	45558451	1.243782

Triangle Counting on RMAT: Total time = 4.475416seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	6249999	12650749	7	4.474904
1	6249999	12546666	5	4.425324
2	6249999	12399926	6	4.363544
3	6250002	12402659	8	4.359183

2. [7 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:
   Adjusting task distribution according to edge counts helps to equalize the workload and thread execution time, albeit not perfectly evenly. **Yes**, the time taken by each thread is directly related to the triangle counts, there is indeed a correlation between thread time, edges, and triangle count for each thread.

   Triangle Counting on LJ: Partitioning time = 0.000121 seconds. Total time = 28.839210 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	0	17248417	136034241	28.799007
1	0	17248515	144913049	23.361203

2	0	17248571	219731388	17.254227
3	0	17248270	182355208	10.135588

3. [2 point] Run Triangle Counting with `--strategy=3` on LJ graph. Update the thread statistics in the table below.

Triangle Counting on LJ: Total time = 20.888351 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1210215	17262906	170463088	20.888078
1	1210997	17172425	170295267	20.888062
2	1209641	17251578	171154304	20.888008
3	1216718	17306864	171121227	20.887936

4. [7 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:
Thread execution times seem similar because of the added barrier, but the number of edges per thread varies. **No**, the barrier within each thread ensures apparent consistency in thread time, with finished threads waiting for unfinished ones until all tasks are complete.

PageRank on LJ: Total time = 41.912827 seconds.

thread_id	num_vertices	num_edges	time_taken
0	24237840	858402620	41.912194
1	24237840	310313840	41.912151
2	24237840	142828980	41.912127

3	24237900	68330020	41.894121

5. [7 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:
   The thread's barrier1 time is inversely proportional to its edge count: more edges, shorter barrier1 time. **Yes**, it is correlated. Thread's barrier1 time is inversely proportional to the number of edges it
   receives. Fewer edges result in higher Barrier1 time due to waiting for other threads.

   PageRank on LJ: Total time = 41.912827 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24237840	858402620	0.000779	0.011085	0	41.912194
1	24237840	310313840	21.848762	0.004772	0	41.912151
2	24237840	142828980	31.287773	0.003513	0	41.912127
3	24237900	68330020	36.866989	0.000567	0	41.894121

6. [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:

Strategy 2 leads to increased Barrier2 time compared to Strategy 1 because the second loop depends on each thread's vertices. In Strategy 1, uniform distribution of vertices among threads results in similar second loop times, reducing the wait time for Barrier2.

PageRank on LJ: Total time = 25.943940seconds. Partitioning time = 0.000115 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	6510800	344968340	1.530026	3.358619	0	25.909338
1	10210960	344970300	1.454052	3.133338	0	25.856383
2	18080960	344971420	0.221310	2.656787	0	25.760627
3	62148700	344965400	0.001580	0.000546	0	25.760553

7. [7 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? What is your observation on the time taken by each thread compared to time taken by each thread in question 6? Why are they same/different?

   Answer: Compared to the prior question, Barrier1 and Barrier2 times decrease, but individual thread execution times extend due to frequent calls to getNextVertexToBeProcessed. These calls entail shared variable updates, thereby increasing the time spent in getNextVertex.

   PageRank on LJ: Total time = 74.692427 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24252940	345135962	0.000970	0.000927	9.010587	74.692011
1	24206163	344684585	0.001043	0.000917	9.007252	74.692003
2	24255717	344956387	0.000990	0.000943	9.016585	74.691965
3	24236600	345098526	0.001019	0.000942	9.012471	74.677298

8. [6 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: getNextVertex_time takes significantly longer than the previous strategy's barrier time due to frequent calls getNextVertexToBeProcessed() in each iteration, which equals the number of vertices, and the repeated updates of shared variables.

   PageRank on LJ: Total time = 74.692427 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24252940	345135962	0.000970	0.000927	9.010587	74.692011
1	24206163	344684585	0.001043	0.000917	9.007252	74.692003
2	24255717	344956387	0.000990	0.000943	9.016585	74.691965
3	24236600	345098526	0.001019	0.000942	9.012471	74.677298

9. [6 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 time required for getNextVertexToBeProcessed() decreased substantially, signifying a significant reduction in function calls, leading to shorter execution time. This decrease can be attributed to the reduced frequency of getNextVertexToBeProcessed() calls.

   PageRank on LJ: Granularity = 2000. Total time = 30.707499 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24318139	345399059	0.001703	0.001511	0.005506	30.707008
1	24198713	344729639	0.001703	0.001511	0.005506	30.707008
2	24324997	345036872	0.002559	0.000988	0.005636	30.706998

3	24109571	344709890	0.002172	0.001606	0.005603	30.706302

10. [9 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:
    Performance is hampered by communication overhead and thread synchronization. Enhancing it involves implementing more refined task partitioning, workload balancing strategies, or task decomposition to optimize thread utilization and minimize overhead.