CMPT 431: Distributed Systems (Fall 2023)

Assignment 2 - Report

Name	Manav Meghpara
SFU ID	301401593

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: In rmat, all threads take around similar time to complete the task. In LJ, every threads takes different time to complete same number of vertex. Because the work in countTriangle and number of edges is different in both graphs resulting in workload imbalance between threads.

   Triangle Counting on LJ: Total time = 59.04833 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken

0	1211893	42920143	339204247	59.047814
1	1211893	15515691	213398829	12.475607
2	1211893	7141445	84872369	3.923223
3	1211892	3416494	45558441	1.161025

Triangle Counting on RMAT: Total time = 4.65546 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	6250000	12650751	7	4.654930
1	6250000	12546670	5	4.614338
2	6250000	12399925	6	4.545296
3	6249999	12402654	9	4.555721

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:
Despite having approximately same number of edges, there is time difference among threads. They are Not correlated. Because most of the work is done in countTriangle which depends on indegree and outdegree of its neighboring vertex. This accounts for uneven workload .

Triangle Counting on LJ: Partitioning time = 0.026537 seconds. Total time = 25.49327 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	0	17248451	136034350	25.466372
1	0	17248478	144912938	19.896218
2	0	17248452	219730487	14.209533

3	0	17248392	182356111	8.033358

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 = 20.36070 seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1208376	17207213	169281711	20.360002
1	1206994	17245828	170907416	20.359967
2	1209122	17207552	170959253	20.359862
3	1223079	17333180	171885506	20.359816

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:
Clearly there is a significant difference in num_edges for each thread, while the time remains same across threads. There is uneven workload distribution. Because, the amount of work depends on the number of edges, which is clearly different among threads.

PageRank on LJ: Total time = 60.952352 seconds.

thread_id	num_vertices	num_edges	time_taken
0	24237860	858402860	60.951546
1	24237860	310313820	60.950832
2	24237860	142828900	60.951441
3	24237840	68329880	60.937767

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:
   Both Barrier1 time and num_edges are significantly different for each threads. They are correlated to each other. As the number of edges increases, workload increases and so it waits less on barrier while threads with lesser edges will have to wait more on barrier.

   PageRank on LJ: Total time = 60.952352 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24237860	858402860	0.000036	0.001854	0	60.951546
1	24237860	310313820	1.641108	0.000224	0	60.950832
2	24237860	142828900	2.277595	0.000034	0	60.951441
3	24237840	68329880	2.664601	0.000025	0	60.937767

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:
   Barrier2 times for both strategy is significantly different for first 3 threads while almost similar for last thread. This is because work load in second loop before barrier2 is dependent on num_vertices and it significantly differs for last thread with other 3 threads.

   PageRank on LJ: Total time = 26.263030 seconds. Partitioning time = 0.027892 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	6510820	344969020	0.077385	0.161222	0	26.234454
1	10210920	344969560	0.072486	0.150125	0	26.234429
2	18080820	344969040	0.013464	0.127179	0	26.234403
3	62148860	344967840	0.000039	0.000034	0	26.221953

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:
   Both Barrier times reduced significantly as compared to Q6 and are even among threads since the workload is evenly distributed nobody needs to wait at barrier. Total time for threads is much more than Q6 because of added work of waiting to get next vertex each time it finishes vertex.

   PageRank on LJ: Total time = 86.467335 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24233109	345065788	0.001053	0.000769	9.910531	86.466496
1	24250808	344758039	0.001000	0.000765	9.904200	86.466454
2	24237845	345001133	0.000973	0.000803	9.901119	86.466363
3	24229658	345050500	0.000962	0.000792	9.918706	86.451239

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:

The time taken by **getNextVertexToBeProcessed()** is similar for each thread but too high overall. Because this function is called many times by each threads causing a flood. Also this is an atomic function which will create an extra overhead making the program slow.

PageRank on LJ: Total time = 86.467335 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24233109	345065788	0.001053	0.000769	9.910531	86.466496
1	24250808	344758039	0.001000	0.000765	9.904200	86.466454
2	24237845	345001133	0.000973	0.000803	9.901119	86.466363
3	24229658	345050500	0.000962	0.000792	9.918706	86.451239

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 taken by **getNextVertexToBeProcessed()** is very low for all threads compared to Q8. There is now fewer calls to this function, as every thread will call this function only after they complete 2000 vertices of work. This reduced the collision of this function calls by threads.

PageRank on LJ: Granularity = 2000. Total time = 27.748899 seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24230284	345872877	0.009656	0.000916	0.005390	27.748169
1	24255855	345040391	0.010159	0.001257	0.005349	27.748116
2	24188997	344313363	0.001631	0.001106	0.005378	27.748084
3	24276284	344648829	0.009548	0.001035	0.005281	27.732947

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:

Because in parallel solution, there is increased overhead from two barrier waits, getNextWork and from CAS operation. One way to increase performance is to partition subgraph such that there is least interactions(edges) with other subgraphs improving shared memory access.