CMPT 431: Distributed Systems (Fall 2021)

Assignment 4 - Report

Name	Yu Ke
SFU ID	keyuk

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`
- 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. Unless otherwise stated, use default values for parameters.

---

1. [3 point] Run Triangle Counting with dynamic task mapping on LJ graph. Update the thread statistics in the table below.

Triangle Counting on LJ: Total time = _18.13313__ seconds.

thread_id	num_vertices	num_edges	triangle_count	time_taken
0	1212220	17244740	170038571	18.119327
1	1210911	17282611	170840370	18.132305
2	1210540	17234195	170887578	18.132227
3	1213900	17232227	171267367	18.132259

2. [10 points] Run PageRank with dynamic task mapping 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 5 from Assignment 3 report? What is your observation on the time taken by each thread compared to time taken by each thread in question 5 from Assignment 3 report? Why are they same/different?

Answer:

Compared toA3Q5, the barrier time in A4Q2 is much shorter in both barrier_1 and barrier_2.

Compared to A3Q5, the time taken is much longer in A4Q2 by each thread.

The reason the time costs much more is that each thread's granularity is 1, which means there are a large number of "getNextVertexToBeProcessed()" are called and this wastes a lot of time. Besides, the number of vertices for each thread are different from A3Q5, and this mapping makes time more.

PageRank on LJ: Total time = _74.503930__ seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	time_taken
0	24260759	345080830	0.001247	0.001172	74.495576
1	24173673	344569312	0.001257	0.001190	74.495535
2	24270007	345135734	0.001244	0.001177	74.495623
3	24246981	345089584	0.001261	0.001170	74.480555

3. [10 points] Run PageRank with dynamic task mapping 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 around 7 seconds per thread.

This is relatively high. Because this function is called every time a thread gets a new vertices, which means it would be called around 2x10^7 each iteration for each thread, and surely it wastes a lot of time.

PageRank on LJ: Total time = _74.503930__ seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken

0	24260759	345080830	0.001247	0.001172	7.254827	74.495576
1	24173673	344569312	0.001257	0.001190	7.243289	74.495535
2	24270007	345135734	0.001244	0.001177	7.243377	74.495623
3	24246981	345089584	0.001261	0.001170	7.245880	74.480555

4. [10 points] Run PageRank with dynamic task mapping 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 much shorter than it is in Q3.

   The reason it is low is because this function is not called every time there is a new vertec for each thread, but is called every granularity number of vertices, which is much smaller than it is by Q3.

   PageRank on LJ: Granularity = 2000. Total time = _35.431706__ seconds.

thread_id	num_vertices	num_edges	barrier1_time	barrier2_time	getNextVertex_time	time_taken
0	24234284	345429303	0.002835	0.000768	0.004646	35.423762
1	24239855	344785055	0.002315	0.000826	0.004652	35.423727
2	24242284	345032389	0.002602	0.000898	0.004780	35.423690
3	24234997	344628713	0.002424	0.000779	0.004565	35.404618

5. [12 points] While dynamic task mapping 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 the cost of the parallel is more than the cost of workload, which makes a not quite good balance in this system. Also communication and synchronization between multiple sub-tasks and processes is difficult to achieve.

I think we could improve the performance by finding an optimal granularity between the two extremes of fine-grained and coarse-grained parallelism.