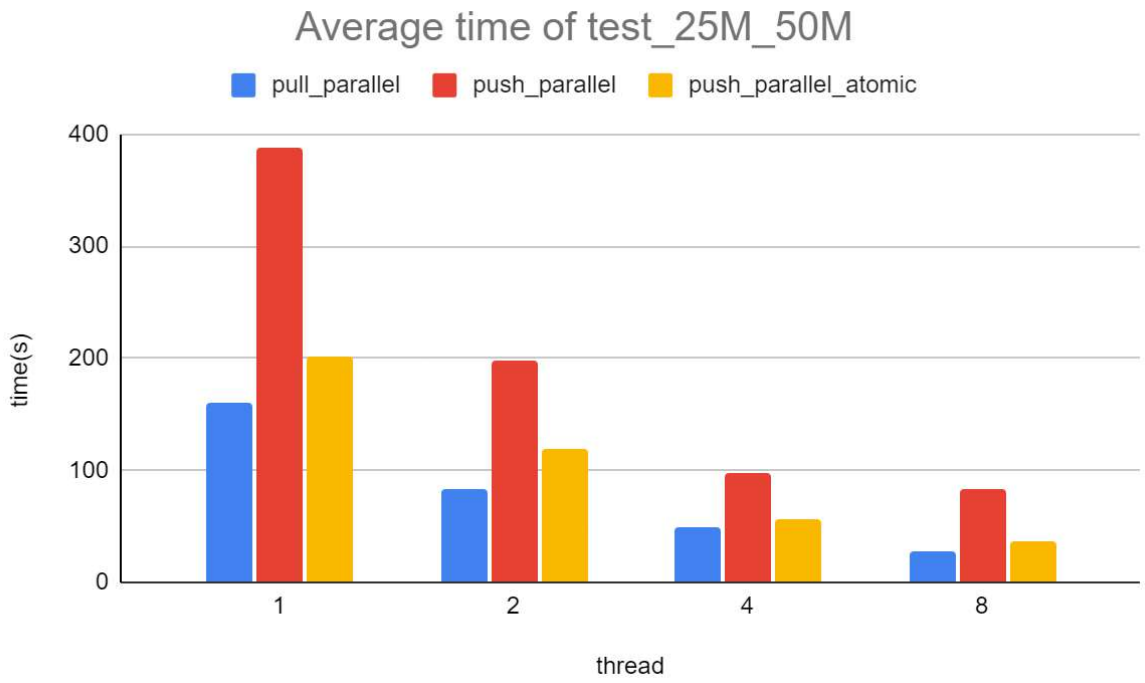
CMPT431 Assignment 3 Report
Siu Yu Chau 301604828 2024-10-3

Q1. Run each of your three parallel programs in parts 1-3 above on the test_25M_50M data set with 1, 2, 4, and 8 threads. Each of your parallel programs should run 3 times, each including 20 iterations on the slow cluster using the default floating point data type. {Total number of runs is 3 (parallel versions) x 4 (different thread counts) x 3 (number of runs for each version/thread combination) = 36 runs]

Plot a graph with average execution time on the y-axis, thread count on the x-axis where each thread configuration has three bars, one for the average runtime of each of your parallel versions. Your graph should be something like this (obviously with different values):



Q2. Based on data from the graph in Q1, which of the three algorithms is more scalable to larger numbers of threads? And why?

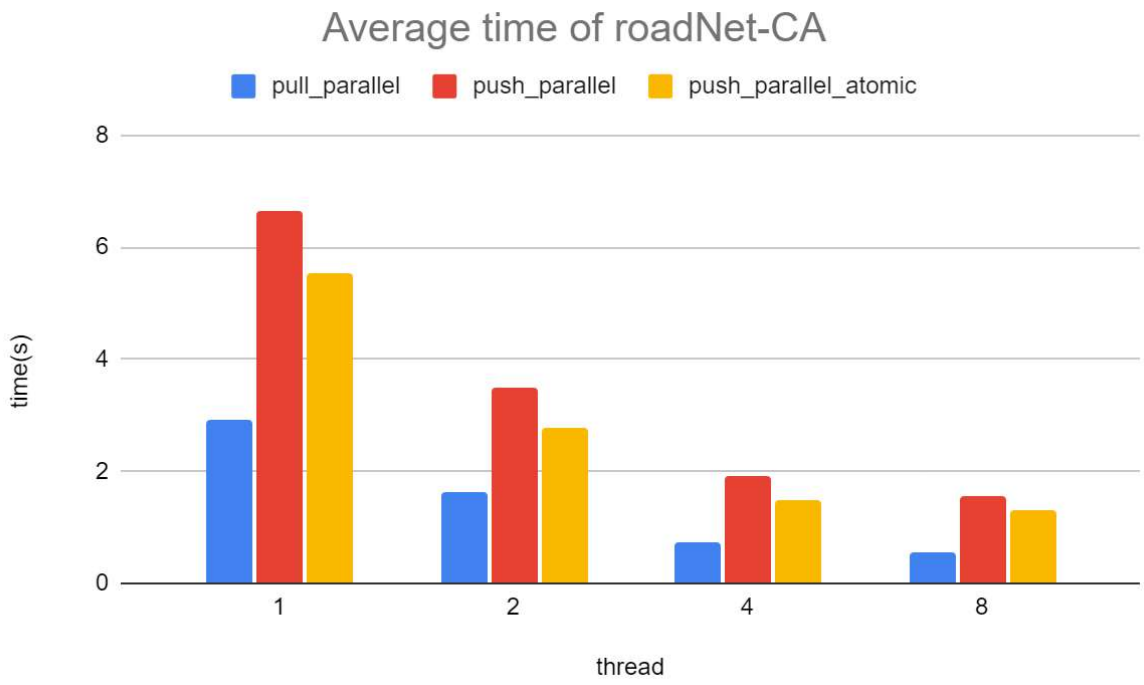
Push_parallel_atomic and Pull_parallel are more scalable than push_parallel, they are kind of similar from this graph.

Pull is scalable one because it only accesses incoming neighbors when inspecting a node, and writes to that node only. So no synchronization is needed to pull parallel.

For push, since we need to write to the node that may belong to other threads, we need synchronization to ensure correctness. The synchronization overhead is slowing down the execution.

Push_parallel_atomic is faster than push_parallel because it doesn't use a lock for synchronization. A mutex requires system call while atomic is low level instruction so it is faster.

Q3. Repeat Q1 for the roadNet-CA dataset and plot a similar graph.



Q4. Based on data from the graph in Q3, which of the three algorithms is more scalable to larger numbers of threads? And why?

The graph is similar to Q1, but since the graph size is smaller, pull_parallel seems better. It could be due to the overhead of using atomic. Pull only uses a barrier which is lightweight.