Milestone Report: Parallelizing SSSP using Delta-Stepping

Daniel Li & Enrico Green

Website: https://github.com/EnricoGreenStudent/15618-Project/

Progress

We started off with building a testing framework for the SSSP problem. We wrote a script that generates lists of edges which form valid graphs, as well as a program which can parse the outputs of the aforementioned script in order to build an adjacency list of the described graph. The program includes correctness testing and timing code so that we can compare various different solutions to the SSSP problem.
In addition to the above testing code, we've also implemented and benchmarked several SSSP algorithms. In particular, we've implemented Dijkstra's algorithm (which is inherently sequential), Bellman-Ford (parallelized using OpenMP), and Delta-Stepping (also parallelized using OpenMP).

Preliminary Results

Test Name	Dijkstra's Runtime	Bellman-Ford Runtime	Delta-Stepping runtime
random-1k	0.3404	0.0538	0.0067
random-20k	39.0228	3.2303	0.0272
sparse-1k	0.0310	0.0076	0.0018
complete-250	0.0400	0.0278	0.0101
cycle-100k	0.0288	37.1803	0.2883
tree-100k	0.0598	121.0124	0.2014

As we can see, the runtimes (and subsequently, speedups between algorithms) varied drastically between tests. Despite being sequential, Dijkstra's algorithm seemed to perform reasonably well in all tests aside from random-20k (a randomly-generated graph with 20,000 vertices and 50,000 edges). Bellman-Ford was much faster than Dijkstra's in tests with few vertices, but still took multiple seconds on random-20k and performed quite poorly in the cycle and tree tests. The delta-stepping algorithm performed fairly well across the board, but was outperformed by the sequential Dijkstra's algorithm in the cycle and tree tasks.

Notably, despite Delta-Stepping being a parallelizable algorithm, we've had some difficulty introducing a significant amount of parallelism using OpenMP. Of the two seemingly parallelizable subroutines, we've only succeeded in parallelizing one so far, and it uses several mutexes which may cause unwanted synchronization. Disabling the OpenMP pragmas and running Delta-stepping sequentially only has a minor effect on many of our test cases. While graph algorithms are typically difficult to parallelize, we think we should still be able to improve our OpenMP implementation by a significant amount.

The original paper specifies a shared memory buffer design using a technique they referred to as dart throwing with retries. Our goal is to experiment with simple queues with locks first, then look into other lock-free techniques such as the one mentioned in the paper if high contention becomes an issue.

Future Schedule & Deliverables

Our goals have not changed – we still plan on implementing both an OpenMP and a CUDA version of the Delta-Stepping algorithm. However, since we have found additional design areas to consider for parallelism with OpenMP, we plan to place a bit more emphasis on the OpenMP improvement task over the next week.

As a result of our struggles parallelizing using OpenMP, as well as other coursework and exams, we may not be able to spend as much time optimizing our CUDA implementation as we would like. Since SSSP algorithms (including Delta-Stepping) are pretty sequential in nature, we think that prioritizing on improving the CPU-based OpenMP implementation over the GPU-based CUDA implementation makes a lot of sense. At the very least, we still plan on submitting a correct CUDA implementation, even if it does not achieve significant speedups.

For our poster session, we plan on discussing the Delta-stepping algorithm and the challenges we encountered while implementing it in both OpenMP and CUDA. We don't plan on having any interactive demos, but we will have several graphs/charts comparing performance across a variety of tests. If we have time, we would like to include some benchmarks from real-world graphs as well.

Date	Objective	Assigned to
12/03 - 12/06	Design data structures for CUDA delta-stepping, passing data to device memory	Daniel
12/03 - 12/06	Improve parallelization in OpenMP delta-stepping implementation	Enrico
12/07 - 12/10	Implement CUDA Delta-Stepping algorithm	Both
12/07 - 12/10	Meet and finalize report/demonstration contents	Both
12/11 - 12/14	Write final report	Both

12/11 - 12/14	Debug and improve performance of CUDA algorithm	Enrico
12/11 - 12/14	Find several real-life graphs and benchmark our various algorithms on them	Daniel