Milestone Report for GPU-Accelerated Single Source Shortest Path (SSSP) Algorithms

URL

The project repository can be found at: https://github.com/Erice1221/GPU-Accelerated-SSSP-Algorithms

Work Completed So Far

Daniel: So far, I have implemented a single-threaded version of Dijkstra's algorithm in C++ as a baseline for performance comparison. I also created a basic CUDA implementation of Delta-Stepping SSSP algorithm. The CUDA implementation currently is able to preduce correct shortest path results on graphs, but it is not yet optimized for performance. I was expected to have some additional optimizations done by this milestone, but I ran into some unexpected challenges with memory management on the GPU, which slowed down my progress. I decided to focus on getting a correct implementation first before diving into optimizations, and would aim for a better optimized delta-stepping implementation before the final submission. In addition, I have also setup the program structure and testing framework to facilitate further development and benchmarking. Further work to help Eric to integrate his work on Bellman-Ford algorithms would also be required.

Eric: Currently, I have implemented a cpu sequential version of Bellman Ford in C++ as a baseline for the CUDA version. I also created the CUDA parallel version that essentially uses an edge parallel approach where each thread relaxes multiple edges in parallel. My first primary goal was to focus on correctness on both solutions the sequential and the parallel version. Then a focus on adjustments like early termination when no edges are relaxed in an iteration, shared memory for reductions within blocks to minimize atomic writes, and tuning the number of edges per thread depending on the graph size. Correctness has been verified across several types of random graphs such as large graphs with 100,000 to 10 million edges. In addition, I setup a testing script to compare speedup, perfomance, and compare results between the sequential and parallel versions.

Preliminary Results

Daniel: The single-threaded Dijkstra's algorithm performs as expected on small to medium-sized graphs, providing correct shortest path results. The CUDA Delta-Stepping implementation also produces correct results, but its performance is currently suboptimal compared to the CPU version due to lack of optimizations. Potential areas for improvement include better memory access patterns, reducing thread divergence, and optimizing the bucket management in the Delta-Stepping algorithm.

Eric: The single threaded sequential version worked as expected, correctly and optimally finding the shortest path. The parallel version also produces the same results (verified with a script that compares results). The sequential version outperforms the parallel version for very small graphs as expected due to overheads. However, on significantly larger graphs we have a 3x speedup for 100k edges up to 40x speedup for 10 million edges. Tuning the number of edges per thread was also important, as there was an optimal value depending on the graph size.

Concerns

Daniel: Currently, I don't have major concerns, but I am aware that optimizing GPU algorithms can be quite challenging, especially with memory management and ensuring efficient parallelism. I plan to allocate sufficient time for testing and optimization in the coming weeks, and would contact the course staff if I encounter any significant roadblocks.

Eric: No major concerns with the Bellman Ford implementation. A potential concern is graphs with edges that must be relaxed sequentially, as this would limit parallelism. The current plan is to continue testing with diverse graphs and finding additional optimizations such as potentially using a frontier based parallel approach or graph preprocessing.

Refined Schedule

Dec 1 - Dec 3:

- Daniel: Finalize an optimized implementation of Delta-Stepping SSSP algorithm on GPU.
- Eric: Finalize an optimized approach for parallel Bellman Ford on GPU.

Dec 4 - Dec 6:

- Daniel: Begin integrating Bellman-Ford algorithm implementation on GPU. Starts testing and benchmarking both algorithms against the CPU baseline.
- Eric: Anaylze, test, and benchmark optimized implementations and compare them to the sequential versions.

Dec 7:

- Both: Final testing, benchmarking, and documentation. Prepare for submission.