Warp-Centric CUDA Programming for SSSP Algorithms

15-418 Project Checkpoint Report

Yiwen(Victor) Song (yiwenson), Xingran Du (xingrand)

April 25, 2021

1 Work Completed So Far

We planned to explore the benefits of a warp-centric GPU programming paradigm in optimizing CUDA implementations of two SSSP (Singel Source Shortest Path) algorithms: Dijkstra's and Bellman-ford.

So far, we have implemented the sequential CPU version, the naive, and the warp-based GPU version of both Dijkstra's and Bellman-ford SSSP.

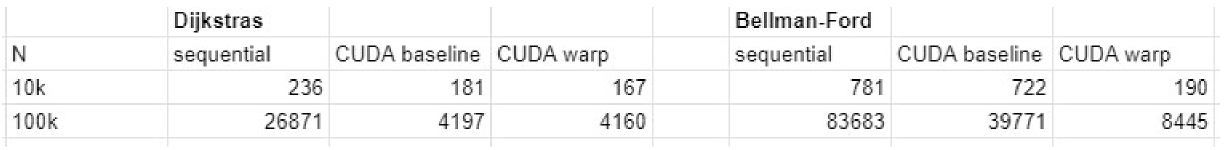
We found that the warp-based optimization is not very applicable for Dijkstra's algorithm, so we only applied a SIMD copy from global memory to shared memory in an effort to coalesce irregular memory accesses. This turns out to have limited effect on the runtime. So we went on to implement the Bellman-Ford algorithm and successfully applied the warp-based optimization there.

We have written scripts to generate random graph inputs following the Barabási–Albert (BA) model, which uses preferential attachment to generate a random scale-free graph (degrees follow the power law). We can choose the number of nodes and edges in the graph, and benchmark each of our SSSP implementations on the generated input.

With respect to the project plan, we are ahead of schedule for algorithm implementations, but have not yet explored the performance in detail. We have some preliminary results in the following section, based on a limited amount of input sizes, and we have only one "natural" implementation for each of the naive and the warp-based GPU versions; a lot of design decisions should be explored and carefully considered in the following weeks, and the performance benchmarking should cover more diverse inputs, and be more fine-grained than whole program runtime.

2 Preliminary Results

We evaluate with fixed M = 3 and multiple N values. This means that the graphs have about the same amount of sparsity (with numEdges = 3 × numNodes) and varying number of nodes N.



	Dijkstras				Bellman-Ford		
N	sequential	CUDA baseline	CUDA warp		sequential	CUDA baseline	CUDA warp
10k	236	181	167		781	722	190
100k	26871	4197	4160		83683	39771	8445

3 Adjusted Goals

- Implement the delta-stepping algorithm (sequential CUDA), which is a modified version of Dijkstra's/Bellman-Ford. Use the same warp-centric idea to optimize the implementation.
- Evaluate with different graph regularity and node degree densities. Add timer code to explore the fine-grained memory access time, work balance, bandwidth, etc.
- Add at least one real-world graph example, and compare our results with results in a paper.
- Compare and discuss the workload difference of these three algorithms
- If time permits, implement and benchmark additional optimizations to these algorithms

4 Adjusted Schedule

- Week Apr 26 - part I: read papers on the delta-stepping algorithm
- Week Apr 26 - part II: coding (implement the thing above)
- Week May 3 - part I: continue coding, plan benchmarks
- Week May 3 - part II: start evaluating speedup, adjust the implementation or apply further optimizations, iterate
- Week May 10: Work on project report. If time permits, work on the stretch goals.

5 Final Deliverables

We will present a detailed report on the effectiveness of the warp-centric optimization. We will show charts and graphs for the benchmark results on the three SSSP algorithms and different inputs, comparing no optimization, warp-based optimization, and other optimizations we see fit, to evaluate the power as well as the limits of the warp-centric GPU programming paradigm in SSSP graph applications.

6 Remaining Issues

Our concern right now is whether we can successfully apply the warp-centric programming paradigm to the delta-stepping algorithm and get a good amount of speedup. It turned out to be hard for Dijkstra's, but at first glance the delta-stepping algorithm should benefit from the warp-centric programming paradigm.