GPU-Accelerated Single Source Shortest Path (SSSP) Algorithms

URL

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

Summary

We aim to implement and benchmark GPU-accelerated versions of Single Source Shortest Path (SSSP) algorithms, specifically delta-stepping and parallel Bellman-Ford. The project will focus on optimizing these algorithms for GPU architectures to enhance performance on large-scale graphs. Additionally, we will explore connections and compare our implementations with the latest SSSP algorithms proposed in recent literature.

Background

The Single Source Shortest Path (SSSP) problem is a fundamental graph algorithm with applications in various domains, including transportation, networking, and social network analysis. The problem involves finding the shortest paths from a single source vertex to all other vertices in a weighted graph. Classical algorithms like Dijkstra's has been widely used, but their sequential nature of using priority queues limits their scalability on large, irregular graphs.

To explore more parallelism, a number of algorithms has been proposed to trade strict ordering for approximate ordering or extra work. For example, Bellman-Ford relaxes all edges repeatedly, allowing for more parallelism at the cost of extra work. Delta-stepping is another algorithm that lies between Dijkstra's and Bellman-Ford, which groups vertices into "buckets" based on their tentative distances and processes these buckets in order, allowing for more parallelism while still maintaining some ordering. These algorithms forms the basis for many parallel SSSP implementations.

GPU offers massive fine-grained parallelism and high memory bandwidth, making it a promising platform for accelerating graph algorithms. However, real-world graphs are often sparse and highly irregular, vertex degrees vary widely, and frontiers can change size dramatically between iterations. This leads to uncoalesced memory accesses, branch divergence within warps, and high contention on shared data structures such as frontier queues or distance arrays. Designing efficient GPU algorithms for SSSP requires addressing these challenges through careful data structure design, memory access optimization, and load balancing techniques.

Recently, Duan et al. proposed a new SSSP algorithm on directed graphs with non-negative real edge weights that breaks the long standing "sorting" barrier of Dijkstra's algorithm on sparse graphs [1]. Their approach carefully combines Dijkstra-style frontier maintenance with Bellman–Ford–like bounded multi-source relaxations and recursive frontier shrinking, showing that one can compute all distances asymptotically faster than sorting vertices by distance, even in directed graphs. This result not only settles a major open question in the theory of SSSP but also reinforces the importance of reducing reliance on global priority queues and expensive ordering operations.

For this project, we would like to implement GPU-accelerated versions of delta-stepping and parallel Bellman-Ford algorithms, and explore optimizations specific to GPU architectures. We will benchmark our implementations on large-scale graphs from various domains (social networks, road networks, etc.) and compare their performance with Dijkstra's Algorithm as a reference. Additionally, we will investigate potential connections between our implementations and the newly proposed SSSP algorithm by Duan et al. to identify scenarios where GPU acceleration provides significant benefits.

The Challenge

This project will have many challenges. First, both delta stepping and parallel Bellman Ford are specific and complex graph algorithms, therefore correctly and efficiently translating them to the GPU requires careful design of several data structures to reduce things like random memory accesses (maximize coalesced reads). Designing the buckets in delta stepping in a way that supports strong parallelism without a large overhead from contention or synchronization will be quite difficult, especially for different types of graphs.

Next, load balancing is another challenge since real world graphs can be very skewed, meaning some threads may have much more work than others. To mitigate this, specific designs are required for working with things like the frontier (active nodes in graph), a distributed work queue for the frontier, and optimized kernel launches. GPU memory limits and transfer overheads are other factors that need to be considered, as large graphs might require batches and methods for partitioning the graph.

Finally, comparing against a strong sequential version and analyzing the implementation with recent SSSP papers will require benchmarking and detailed results (same graphs/consistent metrics) and enough validation to make sure that all speedups are due to algorithmic and implementation choices rather than other noise/bugs.

Resources

For this project, we would reference previous works on GPU-accelerated SSSP algorithms from literature, including the paper that proposed delta-stepping as an algorithm [2], as well as various implementations of delta-stepping on the GPUs for various acceleration methods [3][4]. We would also look into various parallel implementations of Bellman-Ford on GPUs [4][5][6][7] to understand the challenges and optimizations involved in implementing these algorithms on GPU architectures. We would utilize publicly available graph datasets from sources such as the Stanford Large Network Dataset Collection [8] and the 9th DIMACS Implementation Challenge [9] for benchmarking and evaluation purposes.

Goals and Deliverables

To allow for us to keep track of our progress, we would set up the following goals:

50% Milestone (Minimum Viable Project)

- Implement the single-threaded Dijkstra's algorithm as a baseline for comparison.
- Implement a basic version of the parallel Bellman-Ford algorithm on GPU for non-negative edge weights.
- Verify correctness of the implementations on manual test graphs and at least one small graph from public datasets.
- Collect preliminary performance data comparing the GPU Bellman-Ford implementation with the single-threaded Dijkstra's algorithm.

75% Milestone

- Implement a multi-threaded version of Dijkstra's algorithm on the CPU for better baseline comparison. (Potentially using existing lock-free priority queue implementations or a coarse-grained approach, aiming at some speedup over single-threaded Dijkstra's for a better baseline.)
- Implementing a basic version of the delta-stepping algorithm on GPU.
- Verify correctness of both GPU implementations on small and medium-sized graphs from public datasets and robustness across different graph types (sparse, dense, etc.).
- Run preliminary benchmarks on medium-sized graphs, comparing the performance of GPU Bellman-Ford and delta-stepping against CPU Dijkstra's implementations.

100% Milestone (Planned Project Scope)

- Finalize and optimize the GPU implementations of both delta-stepping and parallel Bellman-Ford algorithms.
- Ensure that both the single-threaded and multi-threaded Dijkstra's implementations are fully functional and optimized for fair comparison.
- Potentially explore parallel CPU implementation of Bellman-Ford or delta-stepping for additional baselines.
- Conduct comprehensive benchmarking from multiple graph datasets of varying sizes and characteristics, with different source vertices.
- Analyze and document the performance results, highlighting scenarios where GPU acceleration provides significant benefits, and notice any initialization or overhead costs.
- Prepare a final report detailing the implementation, optimizations, benchmarking results, and insights gained from the project.

125% Milestone (Additional Features)

- Explore potential optimizations specific to GPU architectures, such as memory access patterns, load balancing techniques, and warp-level parallelism.
- Use GPU profiling tools to identify bottlenecks and optimize kernel performance.
- Extend benchmarks to include additional graph datasets and larger graphs to evaluate scalability and sensitivity to graph structure.
- Investigate potential connections between our GPU implementations and the newly proposed SSSP algorithm by Duan et al., identifying scenarios where GPU acceleration provides significant benefits.

150% Milestone (Stretch Goals)

- Implement additional SSSP-related algorithm variants on GPU, such as direction-optimized algorithms or multi-source shortest path algorithms.
- Explore hybrid CPU-GPU approaches for SSSP, where certain parts of the algorithm are executed on the CPU while others are offloaded to the GPU for improved performance.
- Provide additional analysis on the impact of graph characteristics (e.g., density, degree distribution) on the performance of GPU-accelerated SSSP algorithms, identifying best practices for different graph types.

- Potentially produce an open-sourced benchmarking suite for GPU-accelerated SSSP algorithms, allowing other researchers to evaluate and compare their implementations.

Platform Choice

Our code would be implemented using NVIDIA's CUDA platform, which provides a robust framework for developing parallel applications on NVIDIA GPUs. CUDA offers a rich set of libraries and tools that facilitate efficient memory management, kernel execution, and performance profiling, making it well-suited for implementing graph algorithms like SSSP. Additionally, we would leverage existing graph processing libraries such as Gunrock or cuGraph to build upon established data structures and algorithms, allowing us to focus on optimizing our specific implementations of delta-stepping and parallel Bellman-Ford. We would execute the code on the existing GHC machines equipped with NVIDIA GPUs to ensure access to high-performance hardware for benchmarking and testing our implementations.

Schedule

Week 1 (Nov 17 - Nov 23):

Both team members:

- Literature review on SSSP algorithms, focusing on delta-stepping and parallel Bellman-Ford.
- Familiarization with CUDA programming and GPU architecture.

Daniel:

- Creates the project plan.
- Implement the delta-stepping algorithm on GPU.
- Testing on small-scale manual graphs for correctness.

Eric:

- Setup the project directory and version control.
- Implement the parallel Bellman-Ford algorithm on GPU.
- Testing on small-scale manual graphs for correctness.

Week 2 (Nov 24 - Nov 30):

Both team members:

- Determine baseline implementation of whether go for single-threaded or multi-threaded Dijkstra's.

Daniel:

- Continue optimizing delta-stepping implementation, include some basic optimizations for GPU.
- Begin benchmarking on small to medium-sized graphs from public datasets.
- Start creating benchmarking framework and scripts.

Eric:

- Continue optimizing parallel Bellman-Ford implementation, include some basic optimizations for GPU.
- Begin benchmarking on small to medium-sized graphs from public datasets.
- Start writing the baseline Dijkstra's implementation.

Week 3 (Dec 1 - Dec 7):

Both team members:

- Create a milestone report documenting progress, challenges, and next steps.
- Start creating the final report details.

Daniel:

- Finalize delta-stepping implementation and optimizations.
- Conduct comprehensive benchmarking against baseline Dijkstra's implementation.
- Analyze performance results and identify bottlenecks.

Eric:

- Finalize parallel Bellman-Ford implementation and optimizations.
- Conduct comprehensive benchmarking against baseline Dijkstra's implementation.
- Analyze performance results and identify bottlenecks.

References

[1] R. Duan, J. Mao, X. Mao, X. Shu, and L. Yin, “Breaking the Sorting Barrier for Directed Single-Source Shortest Paths,” arXiv preprint arXiv:2504.17033, 2025.
[2] U. Meyer and P. Sanders, “Δ-stepping: a parallelizable shortest path algorithm,” Journal of Algorithms, vol. 49, no. 1, pp. 114–152, 2003, doi: https://doi.org/10.1016/S0196-6774(03)00076-2.
[3] K. Wang, D. Fussell, and C. Lin, “A fast work-efficient sssp algorithm for gpus,” in Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2021, pp. 133–146.
[4] A. Davidson, S. Baxter, M. Garland, and J. D. Owens, “Work-Efficient Parallel GPU Methods for Single-Source Shortest Paths,” in 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 349–359. doi: 10.1109/IPDPS.2014.45.
[5] P. Agarwal and M. Dutta, “New approach of Bellman Ford algorithm on GPU using compute unified design architecture (CUDA),” International Journal of Computer Applications, vol. 110, no. 13, 2015.
[6] G. Hajela and M. Pandey, “Parallel implementations for solving shortest path problem using Bellman-Ford,” International Journal of Computer Applications, vol. 95, no. 15, pp. 1–6, 2014.
[7] S. Kumar, A. Misra, and R. S. Tomar, “A modified parallel approach to Single Source Shortest Path Problem for massively dense graphs using CUDA,” in 2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011), 2011, pp. 635–639. doi: 10.1109/ICCCT.2011.6075214.
[8] “Stanford Large Network Dataset Collection.” Accessed: Nov. 17, 2025. [Online]. Available: https://snap.stanford.edu/data/
[9] “9th DIMACS Implementation Challenge: Shortest Paths.” Accessed: Nov. 17, 2025. [Online]. Available: https://www.diag.uniroma1.it/challenge9/download.shtml