BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2025
http://excel.fit.vutbr.cz

GPU Acceleration of Dijkstra Algorithm

Author: Jan Fiala, xfiala63@stud.fit.vutbr.cz
Supervisor: doc. RNDr. Milan Češka Ph.D.

Abstract
The Single-Source Shortest Path (SSSP) is a fundamental graph problem concerned with determining the shortest paths from a designated source node to all other nodes in a graph. While Dijkstra's algorithm provides an optimal solution for this problem in a sequential setting, its practical applicability diminishes for large-scale, densely connected graphs comprising millions of nodes. In such cases, repeated computations can result in execution times extending to several minutes or even hours. To address this limitation, parallelization techniques are increasingly employed, offering a substantial reduction in computation time, potentially bringing it down to just a few seconds.

1. Introduction

The Single-Source Shortest Path [1] (SSSP) is a fundamental graph problem with the goal of identifying the shortest paths to all nodes from a specified source node in a graph. Dijkstra's algorithm is optimal for solving the SSSP problem for non-negative weighted graphs with a time complexity of $O(N \cdot \log N + E)$ for the sequential variant optimized through the usage of a Fibonacci heap [2]. The **Boost library** [3] sequential implementation utilizes this structure to accelerate the computation, primarily for dense graphs. However, sequential solutions are not always practical, as we need to repeatedly compute graphs composed of millions of nodes. This work introduces a Novel approach to parallelizing the SSSP problem, aiming to further accelerate computation and reduce execution time significantly. By rethinking the parallelization strategy, this method achieves improved performance over state of the art parallel implementations for both sparse and dense graph scenarios.

2. State of the Art

The GPU-based implementations by **Martín** [4] and **Crauser** [5] serve as baselines for performance comparison. Both approaches parallelize the settlement phase of Dijkstra's algorithm by identifying nodes with the lowest tentative distances across all unsettled nodes. However, this strategy often limits the number of nodes processed in each iteration, especially under wide edge weight distributions.

Crauser introduces a bucket-based system to prioritize nodes with minimal distances, while **Martín** uses a global minimum threshold to settle nodes in parallel. Martín's method incurs minimal synchronization overhead and performs well under narrow weight ranges, where bucket separation in Crauser's method becomes costly. Conversely, Crauser's bucketed approach scales better with broader weight distributions by enabling the settlement of more nodes per iteration.

Figure 1 illustrates the parallel relaxation step during node settlement in Dijkstra's algorithm.

Figure 3 showcases a simplified behavior of the two state of the art approaches. They can safely settle only the lowest tentative distance node, resulting in exploring only the top portion of the graph at the given iteration.

3. Novel Approach

The approach behaves as if the graph was **unweighted**, maximizing the GPU's potential by using all of its resources effectively. Such approach allows us to **explore beyond the costly edges** of a graph earlier, even though they are not necessarily part of the **shortest paths** in the finale. This can be seen in Figure 4.

Current state of the art GPU implementations of

Dijkstra's algorithm struggle with graphs exhibiting a wide range of edge weights. These approaches typically settle only a few, or even a single, node per iteration, leading to limited subgraph exploration, as shown in Figure 3.

The approach mitigates this bottleneck by exploring all reachable nodes, assigning tentative distances that may not initially reflect the shortest paths. Through careful management of concurrent updates, the algorithm ensures that **at least one shortest path**, referred to here as the **Dijkstra path**, is discovered and propagated. As the exploration progresses, incorrect distances are gradually corrected, converging toward the true shortest paths. ddddddddd While the algorithm has a higher theoretical complexity of $O(N^2)$, it achieves significant speedups in practice by exploiting parallelism, accepting the trade-off of redundant computations due to re-settlement of nodes upon updates.

4. Experiments

The implementations were compared on hundreds of real world graphs, as well as on thousands of synthetic ones. Both strengths and weaknesses of the individual implementations were observed along with the behavior of the Novel approach, with the Novel coming on top in most of the evaluations. Boost Library's [3] Dijkstra optimized using a Fibonacci heap served as the sequential baseline for the evaluation.

Figure 5 shows the issue with the State of The Art implementations, especially with the spike for the graph with 22 963 Nodes. If the graph's topology and weight range is not optimal for them, they perform significantly worse in terms of computation time for these graphs, while the Novel mitigates the problem.

Figure 6 demonstrates the capabilities and setbacks of the GPU parallel implementations. These graphs were chosen as a clear example of what the issue is and how the **Novel approach** clearly solves it. The sequential implementation takes over a hundred seconds for these graphs, while the Novel takes about 2 seconds. Additionally, the Novel algorithm has the least overhead, which allows it to perform well even in a sparse graph environment.

Figure 7 presents how the State of The Art implementations fall short of even the optimized Boost sequential Dijkstra for sparse graphs, with the Novel coming out on top yet again due to its almost negligible overhead, being the most versatile and effective approach for most of the graphs when compared to the **Martín** and **Crauser** variants.

4.1 Are the re-implementations competitive?

In the following figures 1 and 2, we can see the comparison between the original implementations and the re-implementations along with the Novel approach and seq. BOOST vs. seq. Dijkstra with a priority queue:

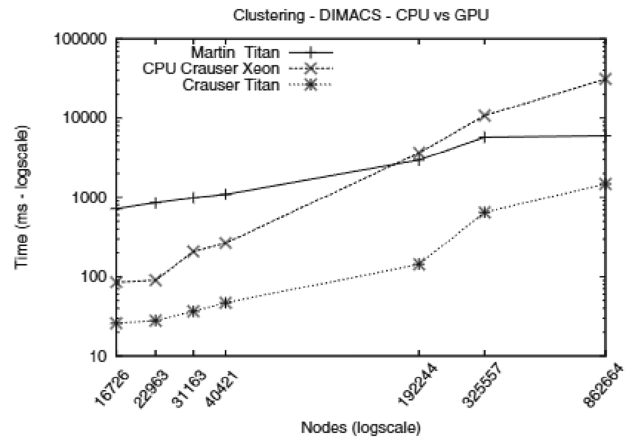

Figure 1. Baseline results of the original state of the art implementations

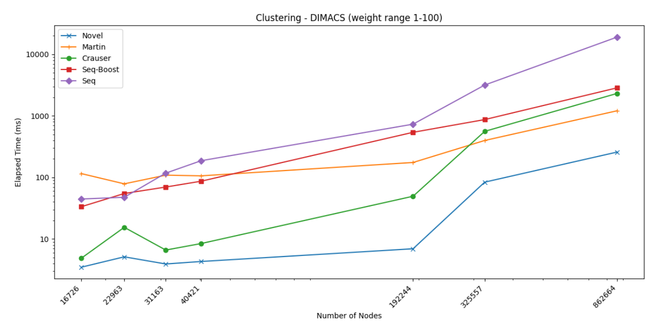

Figure 2. Results of the re-implementations, seq. BOOST Dijkstra + Novel approach

The testing environment's hardware is very similar in both the original papers and our evaluation case. Thanks to that we are able to state that the re-implementations of the State of The Art implementations are at least comparable, and that the Novel approach steadily outperforms them in most of the cases. All experiments were evaluated within the same environment. The setup can be seen within the poster itself in a small box within the "Experiments" section.

Acknowledgements

I would like to thank my supervisor doc. RNDr. Milan Češka Ph.D. for the help and thoughts about the existing bottlenecks that helped me come up with the Novel approach to the SSSP problem, as well as his feedback, time and patience.

References

[1] Andrew Davidson, Sean Baxter, Michael Garland, and John D Owens. Work-efficient parallel gpu methods for single-source shortest paths. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 349–359. IEEE, 2014.

[2] Robert E. Tarjan. Fibonacci heaps and their applications, 2000. Accessed: 2024-09-13.

[3] Jeremy Siek. Dijkstra's shortest paths. online, 2001. Accessed: 2024-09-10.

[4] Hector Ortega-Arranz, Yuri Torres, Diego R Llanos, and Arturo Gonzalez-Escribano. A new gpu-based approach to the shortest path problem. In *2013 International Conference on High Performance Computing & Simulation (HPCS)*, pages 505–511. IEEE, 2013.

[5] Hector Ortega-Arranz, Yuri Torres, Arturo Gonzalez-Escribano, and Diego R Llanos. Comprehensive evaluation of a new gpu-based approach to the shortest path problem. *International Journal of Parallel Programming*, 43(5):918–938, 2015.