Parallel Graph Algorithms using MPI and CUDA with Full-Stack Visualization

Group Chupakabara

May 7, 2025

1 Introduction

This project explores parallel implementations of three classical graph algorithms — Dijkstra, Floyd-Warshall, and Bellman-Ford — using MPI (Message Passing Interface) for distributed computing and CUDA (Compute Unified Device Architecture) for GPU acceleration. To enhance accessibility and analysis, we developed a full-stack web interface with a React TypeScript frontend and a Django backend that visualizes the algorithms' performance through interactive charts.

2 Problem Description and Motivation

Graph algorithms are fundamental in fields like routing, logistics, and network optimization. However, their time complexity becomes a bottleneck for large-scale data. This project seeks to accelerate these algorithms by applying parallelism and allow real-time performance visualization via a web application.

3 Related Work

Previous research highlights the benefits of parallelizing graph algorithms. MPI is commonly used for distributed systems, and CUDA efficiently utilizes GPUs for highly parallelizable computations. While similar efforts exist in high-performance computing, our project distinguishes itself with a full-stack platform for visualization and comparison.

4 System Design

4.1 Parallel Algorithm Implementation

- **Languages**: Python (with mpi4py), CUDA C++
- **Algorithms**:
  - Dijkstra (MPI & CUDA)
  - Floyd-Warshall (MPI & CUDA)
  - Bellman-Ford (MPI & CUDA)

- **Graph Representation**: 2D adjacency matrices with infinity for unreachable nodes
- **Testing Scale**: Graphs with 100 to 1000 nodes

4.2 Full-Stack Application

- **Frontend**: React with TypeScript, Chart.js and Recharts
- **Backend**: Django with REST APIs
- **Web Pages**:
  - Landing Page
  - Dijkstra MPI Page
  - Dijkstra CUDA Page
  - Floyd-Warshall MPI Page
  - Floyd-Warshall CUDA Page
  - Bellman-Ford MPI Page
  - Bellman-Ford CUDA Page

5 Results and Performance Analysis

5.1 Metrics

- Execution Time (ms)
- Speedup (T_1/T_n)
- Scalability (time vs number of nodes)

5.2 Summary Table

Algorithm	Implementation	Speedup	Notes
Dijkstra	MPI	~3.2x	Sparse graphs perform well
Dijkstra	CUDA	~12x	Excellent on dense graphs
Floyd-Warshall	MPI	~2.8x	Good for smaller matrices
Floyd-Warshall	CUDA	~15x	GPU excels in matrix ops
Bellman-Ford	MPI	~2.5x	Slower due to communication
Bellman-Ford	CUDA	~10x	Strong GPU performance

6 Discussion

CUDA implementations clearly outperformed MPI in large datasets due to GPU parallelism. MPI showed better CPU-core utilization for modestly sized graphs. Visualizing these differences helped us gain a deeper understanding of the performance trade-offs.

7 Conclusion

We demonstrated that parallel graph algorithms significantly reduce computation time using MPI and CUDA. The frontend adds value by making the data visual and interactive, making our project not just computational but educational.

8 Future Work

- Dynamic graph input via frontend
- Real-time graph visualization
- Extend to more graph algorithms like A*, Kruskal's, and BFS

9 Group Members

- Mohammad Noman (2022304): MPI Code + Report Writing
- Ubaid ur Rehman (2022599): CUDA Implementation + Backend
- Shaheer (2022424): Frontend