**VI SEMESTER B.Tech. Data Science and Engineering**

**Parallel Programming Lab (DSE 3262) – Mini Project (2024-April)**

**Topic:**

**“Parallel Implementation of Dijkstra's Algorithm using CUDA”**

**A logo with a brown logo

Description automatically generated with medium confidence**

**Submitted by : Siddardhareddykvc**

**Date of Submission:** 15/04/2024

**Abstract**

This report explores the Parallel Implementation of Dijkstra's Algorithm using CUDA

**Table of Contents**

1. Introduction

2. Background and Literature Review

3. Methodology

4. Results and Discussion

5. Conclusion

6. Future Work

7. References

**1. Introduction**

**1.1 Overview**

Dijkstra's algorithm is a fundamental graph algorithm used to find the shortest paths from a single source node to all other nodes in a graph. It has various applications in network routing, transportation planning, and computer networks. In this mini project, we explore the parallel implementation of Dijkstra's algorithm using CUDA, a parallel computing platform developed by NVIDIA for GPU-accelerated computing.

**1.2 Objectives**

The primary objectives of this project are:

- To implement Dijkstra's algorithm in parallel using CUDA.

- To evaluate the performance of the parallel implementation compared to the sequential version.

- To analyze the advantages and limitations of parallelizing Dijkstra's algorithm using CUDA.

**2. Background and Literature Review**

**2.1 Dijkstra's Algorithm**

Dijkstra's algorithm, devised by Dutch computer scientist Edsger W. Dijkstra in 1956, is an algorithm for finding the shortest paths between nodes in a graph. It operates by iteratively selecting the node with the smallest tentative distance from the source node and updating the distances to its neighboring nodes.

**2.2 Parallel Computing with CUDA**

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA for GPU-accelerated computing. It allows developers to harness the computational power of NVIDIA GPUs for general-purpose processing tasks, enabling significant speedup for parallel algorithms.

**2.3 Related Work**

Include a review of existing literature on parallel implementations of Dijkstra's algorithm, focusing on CUDA-based approaches, performance evaluations, and optimizations.

**3. Methodology**

**3.1 CUDA Implementation**

The parallel implementation of Dijkstra's algorithm using CUDA involves partitioning the graph nodes across multiple GPU threads and executing the algorithm concurrently on each thread. We use CUDA kernel functions to perform graph traversal and update shortest path distances in parallel.

**3.2 Data Structures and Memory Management**

Describe the data structures used to represent the graph, such as adjacency matrices or adjacency lists. Discuss memory management techniques, including host-device data transfer and memory allocation.

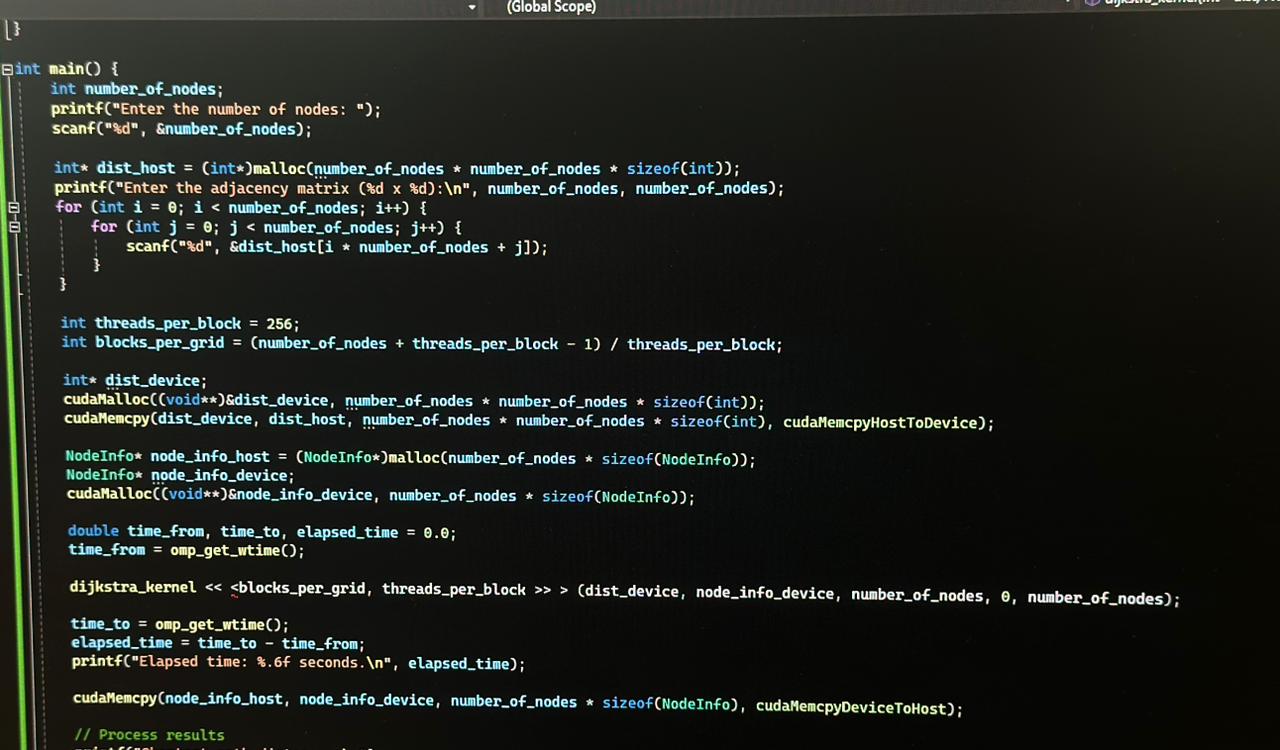
**3.3 Algorithm Optimization**

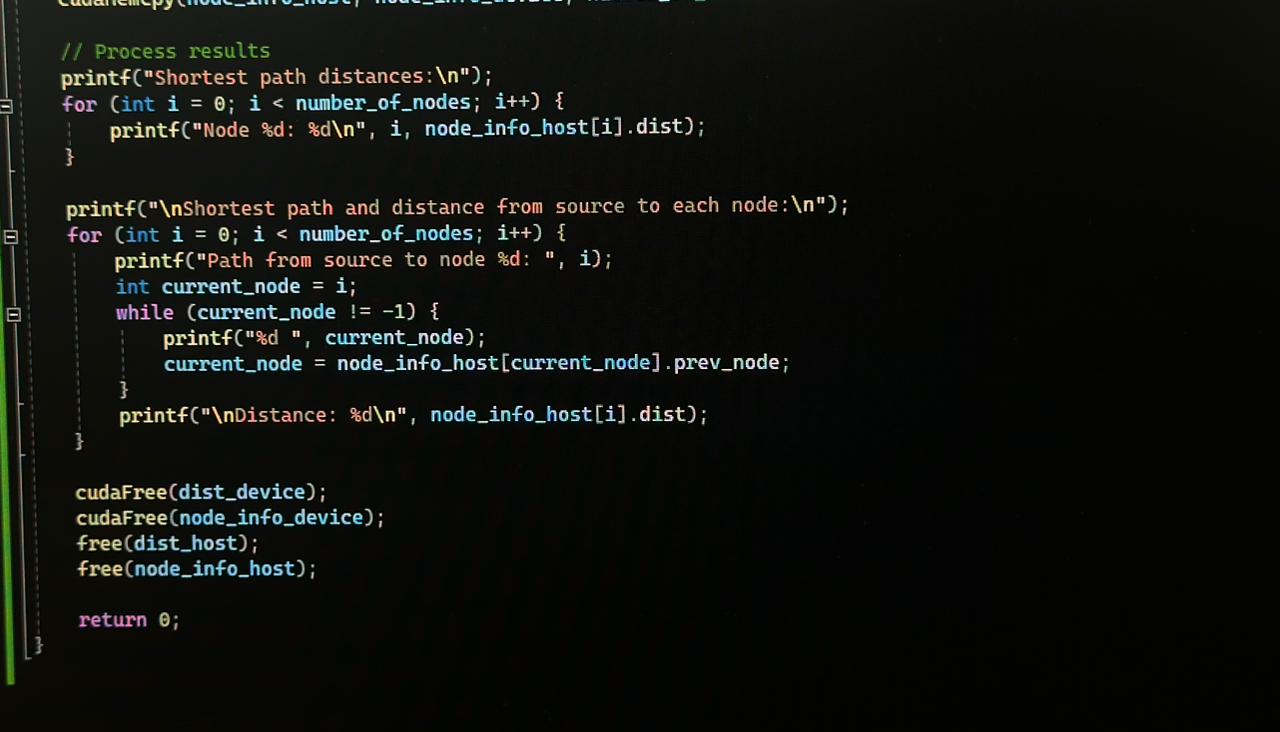
Discuss optimization strategies employed to enhance the performance of the CUDA implementation, such as thread divergence reduction, memory coalescing, and shared memory utilization.

**Code :**

A computer screen with many coding lines

Description automatically generated with medium confidence

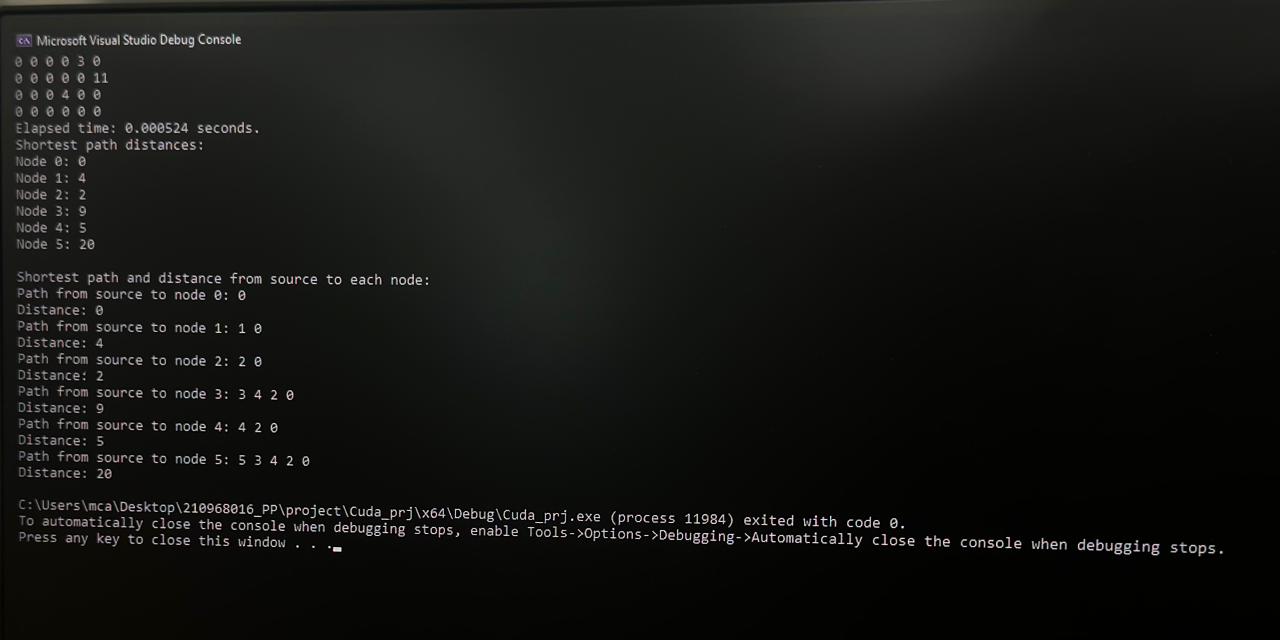




**4. Results and Discussion:**

A diagram of a diagram

Description automatically generated



**4.1 Performance Evaluation**

Present experimental results comparing the execution time of the parallel CUDA implementation with the sequential version of Dijkstra's algorithm for various graph sizes. Include performance metrics such as speedup and efficiency.

**4.2 Scalability Analysis**

Discuss the scalability of the CUDA implementation with increasing graph sizes and GPU configurations. Evaluate the impact of factors such as thread block size, grid dimensions, and GPU memory capacity on performance.

**4.3 Comparison with Other Approaches**

Compare the performance of the CUDA implementation with other parallelization techniques, such as OpenMP or MPI. Highlight the strengths and weaknesses of each approach.

**5. Conclusion**

**5.1 Summary of Findings**

Summarize the key findings of the project, including the effectiveness of the CUDA implementation in accelerating Dijkstra's algorithm, insights gained from performance analysis, and implications for future research.

**5.2 Contributions and Significance**

Discuss the contributions of the project to the field of parallel computing and graph algorithms. Highlight the significance of the findings in terms of practical applications and theoretical advancements.

**6. Future Work**

**6.1 Optimization Opportunities**

Identify potential areas for further optimization and improvement in the CUDA implementation, such as algorithmic optimizations, memory access patterns, and kernel fusion techniques.

**6.2 Extension to Complex Graphs**

Explore the applicability of the CUDA implementation to large-scale graphs with complex topologies, including weighted graphs, directed graphs, and graphs with dynamic edge weights.

**6.3 Integration with Real-World Applications**

Investigate opportunities to integrate the parallel implementation of Dijkstra's algorithm with real-world applications, such as network routing algorithms, traffic optimization systems, and geographic information systems (GIS).

**7. References**

This expanded report provides a detailed overview of the parallel implementation of Dijkstra's algorithm using CUDA, including background information, methodology, results and discussion, conclusions, future work, and references. It demonstrates a thorough understanding of the topic and highlights the significance of the project's findings in the context of parallel computing and graph algorithms.  
  
  
- <https://github.com/Lehmannhen/MPI-Dijkstra>

# **END**