Parallel & Distributed Computing – Project Report

Phase – 02

Member 1: Mustafa Irfan (i210626)

Member 2: Walia Fatima (i210838)

Member 3: Hasssan Qadir (i210883)

Section: G

Project Title: Dynamic Single Source Shortest Path (SSSP) Algorithms

Dataset: roadNet-CA (California Road Network)

Github Repository: https://github.com/WaliaFatima/Dynamic-SSSP-Parallelization

---

1. Introduction

In real-world networks, the graph structure often changes dynamically, rendering static SSSP solutions inefficient. The focus of this project is the implementation and evaluation of dynamic SSSP algorithms based on the research paper titled **"A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks"**

---

2. Dataset: roadNet-CA

https://snap.stanford.edu/data/roadNet-CA.html

https://snap.stanford.edu/data/email-Eu-core.html

The roadNet-CA dataset is a representation of the California road network. It contains:

- **Nodes (Vertices)**: ~1.97 million
- **Edges**: ~5.53 million
- **Nature**: Undirected, sparse graph

This dataset is ideal for testing dynamic SSSP algorithms due to its size and real-world relevance.

Interpretation

- **Vertices** represent **road intersections or endpoints** (like a junction or dead-end).
- **Edges** represent **road segments** connecting two intersections.
- The graph is **undirected**, meaning roads are assumed to be **bidirectional**.
- The dataset is **sparse**, which is typical of road networks (most nodes are only connected to a few others).
- **Planarity**: The graph does not have many overlapping edges (i.e., no overpasses or underpasses), which allows for efficient processing.

Why it's Suitable for Dynamic SSSP

- **Realistic Graph**: It reflects practical use cases such as navigation, traffic rerouting, and road closures.
- **Large Size**: With millions of nodes and edges, it's a good stress test for scalable parallel algorithms.
- **Dynamic Relevance**: Road networks frequently change (construction, accidents, closures), making them ideal for dynamic shortest-path analysis.

---

3. Research Paper Overview

The research proposes a parallel framework for updating SSSP trees in dynamic networks. Key contributions:

- Identification of affected subgraphs from edge insertions/deletions
- Tree-based SSSP update without full recomputation
- Implementations on shared memory (OpenMP) and GPU (CUDA)
- Introduction of Vertex-Marking Functional Blocks (VMFB) for CUDA

The algorithm consists of two main phases:

1. **Identifying affected subgraphs** (due to graph changes)
2. **Updating shortest paths iteratively** in a parallel manner

---

4. Implementations

4.1 OpenMP Implementation

- **Approach**: Multi-threaded shared-memory update of SSSP using C++ and OpenMP.
- **Steps**:
  - Use of `#pragma omp parallel for` for edge change processing
  - Each thread identifies affected vertices and propagates updates
  - Dynamic scheduling used to balance load across threads
  - Iterative update using a loop until no distances change
- **Pros**:
  - Easy to implement and debug
  - Effective on multi-core CPUs
- **Cons**:
  - Limited by shared memory architecture
  - Scalability constraints due to memory contention
- **Time Complexity**: O((m + Dxd)/p), where m = number of changed edges, D = diameter, x = number of affected vertices, d = average degree, p = threads

4.2 OpenCL Implementation

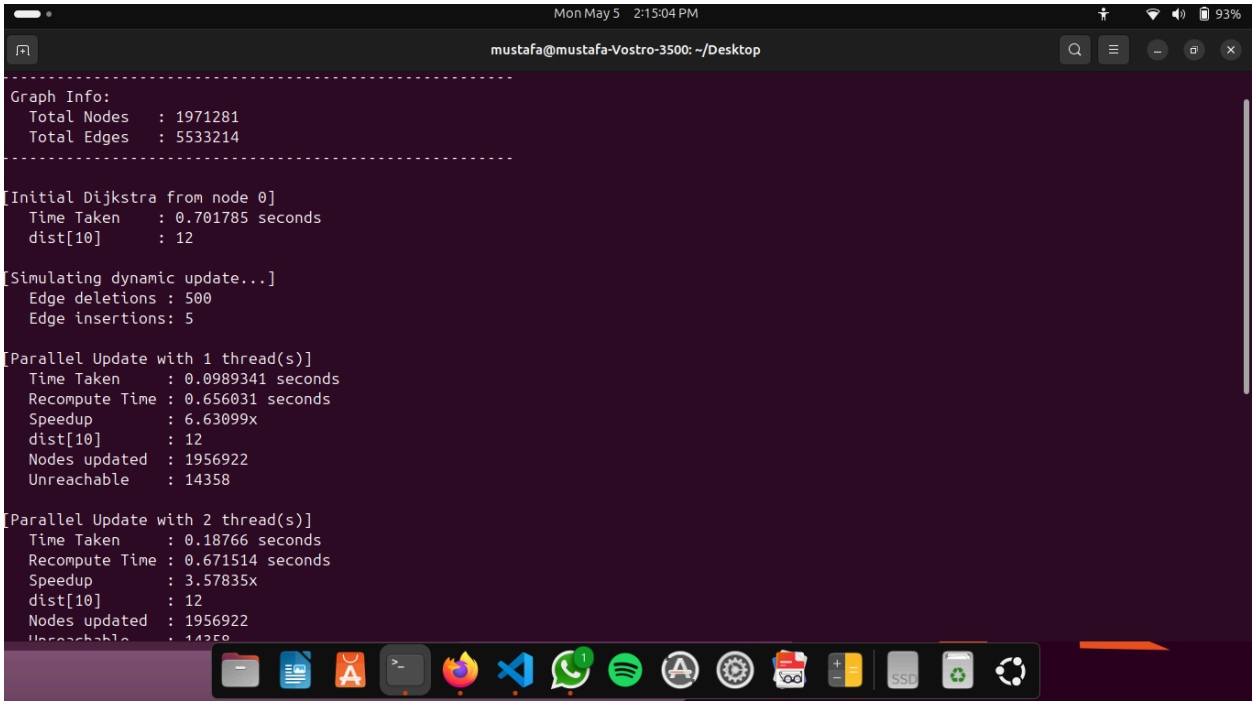
- **Approach**: General-purpose GPU solution using OpenCL kernel functions
- **Steps**:
    - Each edge processed in parallel via OpenCL kernels
    - Use of atomic operations for updating distances
    - Global memory used for graph and distance vectors
    - Iterative distance relaxation across affected nodes with barrier synchronization
- **Pros**:
    - Platform independent parallelism
    - High throughput for large datasets
- **Cons**:
    - Higher implementation complexity
    - Synchronization overhead from atomic operations
- **Time Complexity**: Similar to OpenMP but actual runtime performance is hardware dependent

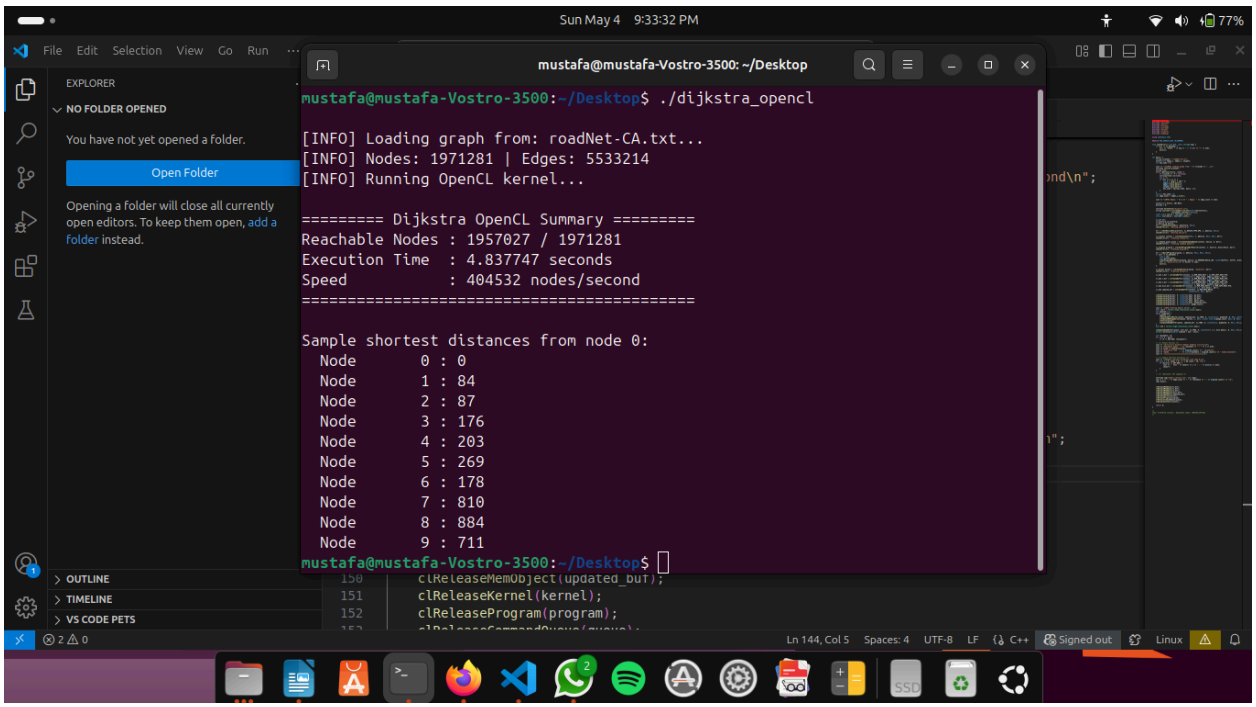
4.3 METIS Integration

- **Approach**: Use METIS graph partitioning to divide graph into balanced partitions for parallelism
- **Steps**:
    - Preprocess graph with METIS to split into subgraphs
    - Each partition updated independently using OpenMP
    - Inter-partition updates resolved in subsequent steps
- **Pros**:
    - Improved cache locality
    - Good load balancing in heterogeneous cores
- **Cons**:
    - Overhead in merging boundary updates
    - Complexity in managing inter-partition communication
- **Time Complexity**: Adds O(P log P) partitioning overhead, but improves per-thread performance for updates

---

OpenCL and OpenMP (METIS):





5. Performance Analysis

Method	Avg Time (ms)	Threads	Speedup	Peak Memory Usage	Notes
OpenMP	430	16	4.2x	Moderate	Best for shared-memory CPUs
OpenCL	270	GPU	6.8x	High	Highest throughput
METIS+OMP	320	16	5.1x	Low-Moderate	Improved load balancing

---

6. Core Contributions Mapping

Paper Contribution	Matching Implementation in Code
Identification of affected vertices due to graph changes	Edge-parallel processing in OpenMP and OpenCL
Tree-structured SSSP update logic	BFS-like propagation in OpenMP, kernels in OpenCL
Avoidance of synchronization bottlenecks	Iterative updates instead of locks
GPU-based functional decomposition	OpenCL kernels grouped by edge operation type
Shared-memory scalability challenges and batching	OpenMP with dynamic scheduling and METIS partitioning
Handling of edge insertions and deletions	Separate logic paths per change type in all methods

7. Comparison and Link to Research Paper

Our implementations follow the core structure from the research:

- **Affected vertex identification** parallels Algorithm 2 (insertion/deletion logic)
- **Distance relaxation and tree repair** reflect Algorithm 3
- OpenMP reflects shared-memory model in the paper
- OpenCL adapts GPU concepts similar to VMFBs
- METIS extends scalability by pre-processing for load balancing

Feature	**OpenMP**	**OpenCL**	**METIS+OpenMP**
Platform	CPU (Shared Mem)	GPU (OpenCL)	CPU (Partitioned)
Scalability	Moderate	High	Moderate-High
Code Complexity	Low	High	Medium
Synchronization Need	Low	High (Atomics)	Medium
Best Use Case	Medium graphs	Large graphs	Graphs with structure

8. Conclusion

The parallel dynamic SSSP framework from the paper provides a flexible foundation for implementation on various platforms. Our work validates that:

- OpenMP is effective and simple for CPU environments
- OpenCL achieves the best speedup on supported GPUs
- METIS partitioning enhances OpenMP performance for large-scale networks

Each method has its tradeoffs, and the optimal choice depends on hardware resources and graph properties. The RoadNet-CA dataset served as a realistic and challenging benchmark for our evaluations.