Prim’s Algorithm Parallelization

ROHAN SURESH, PARTHIV YAGNESH

Overview

- Prim's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, weighted graph.
- Prim's algorithm guarantees to find the MST for any connected, weighted graph with non-negative edge weights.
- Time complexity: O(V^2) for adjacency matrix representation, or O(E log V) for adjacency list representation using a priority queue.

Algorithm

1. Read input graph details including the number of vertices, vertex degrees, and edge weights.
2. Initialize data structures including an adjacency list to represent the graph, a priority queue to store edges, and a boolean array to track visited vertices.
3. Process each vertex in the graph, adding its adjacent edges to the priority queue.
4. Pop edges from the priority queue, adding new edges to the MST and updating the priority queue with adjacent edges.
5. Repeat until the MST contains all vertices except the source.

Parallelization Workflow

Parallel Vertex Selection:

Assign each thread a subset of vertices to process independently.

Parallelize the processing of vertices and their adjacent edges to identify minimum-weight edges.

Atomic Operations and Synchronization:

Ensure atomicity for operations such as updating the priority queue and marking vertices as visited to prevent race conditions.

Use synchronization mechanisms like locks or critical sections when accessing shared data structures.

Parallelization Workflow

Edge Relaxation and Update:

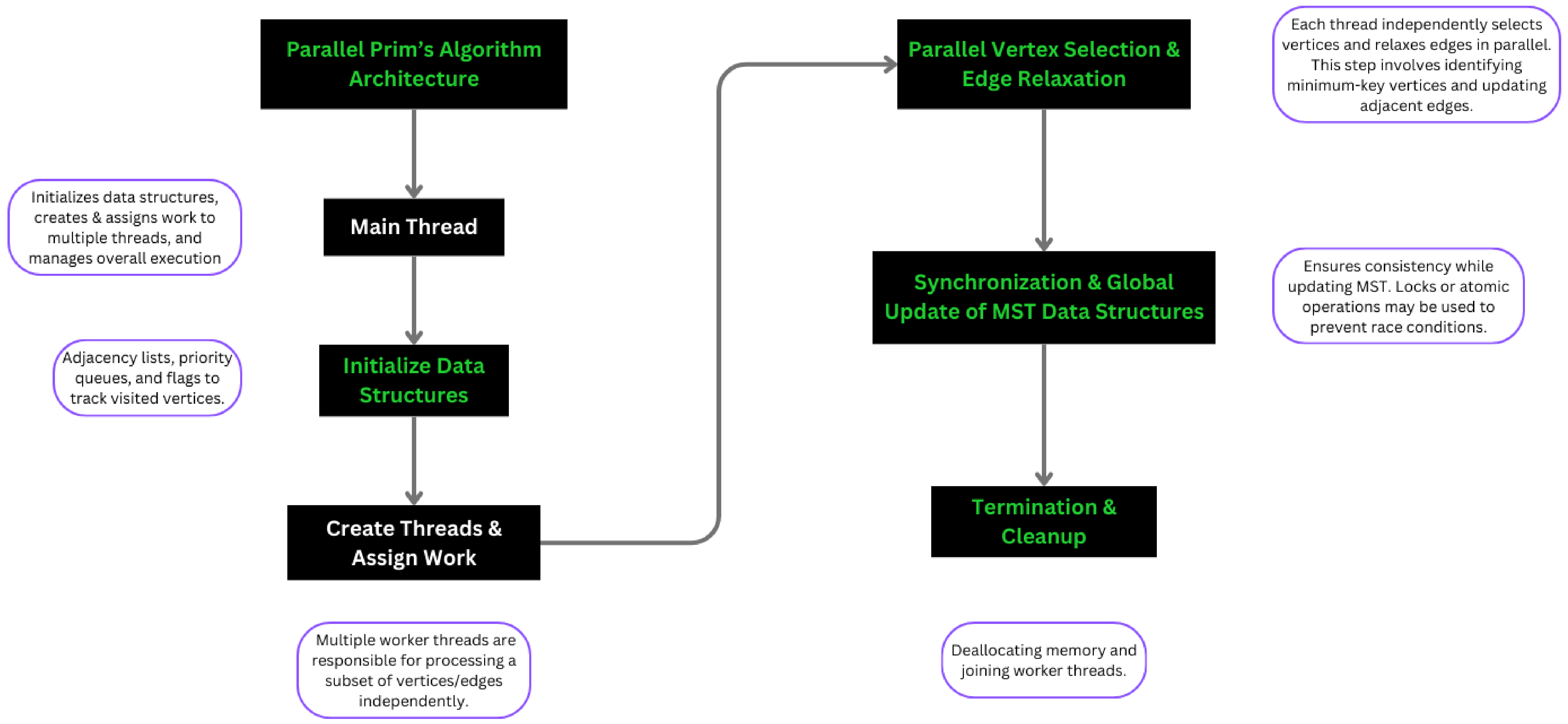
Parallelize the process of relaxing edges and updating the priority queue with adjacent edges.

Ensure proper synchronization to maintain consistency when updating shared data structures.

Optimizations:

Implement optimizations such as early termination to stop the algorithm once the MST is complete.

Explore opportunities for workload balancing to evenly distribute computation among threads.

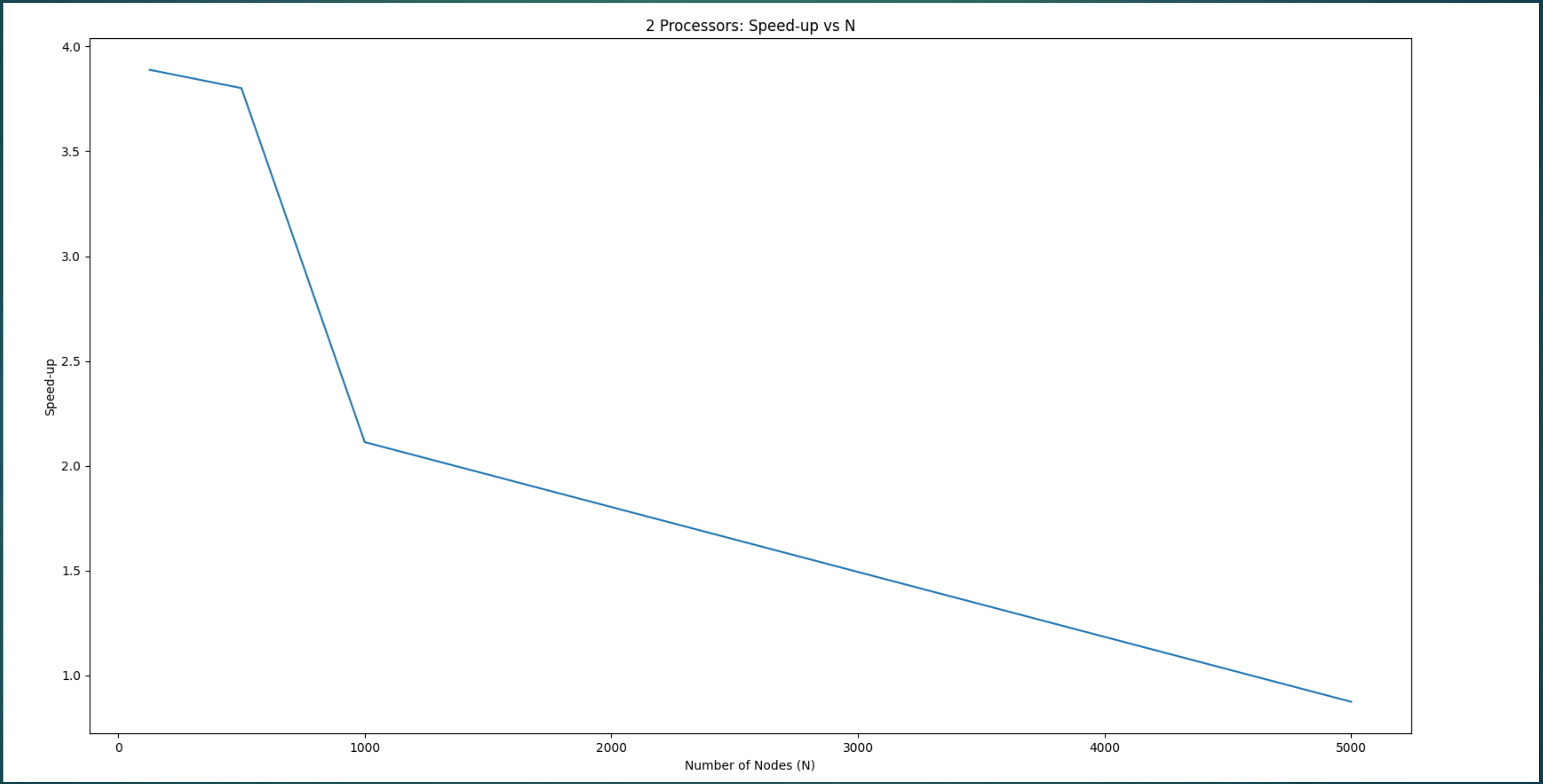

Parallel Prim's Algorithm Architecture
Initializes data structures, creates & assigns work to multiple threads, and manages overall execution
Main Thread
Adjacency lists, priority queues, and flags to track visited vertices.
Initialize Data Structures
Create Threads & Assign Work
Multiple worker threads are responsible for processing a subset of vertices/edges independently.
Parallel Vertex Selection & Edge Relaxation
Each thread independently selects vertices and relaxes edges in parallel. This step involves identifying minimum-key vertices and updating adjacent edges.
Synchronization & Global Update of MST Data Structures
Ensures consistency while updating MST. Locks or atomic operations may be used to prevent race conditions.
Termination & Cleanup
Deallocating memory and joining worker threads.

Challenges

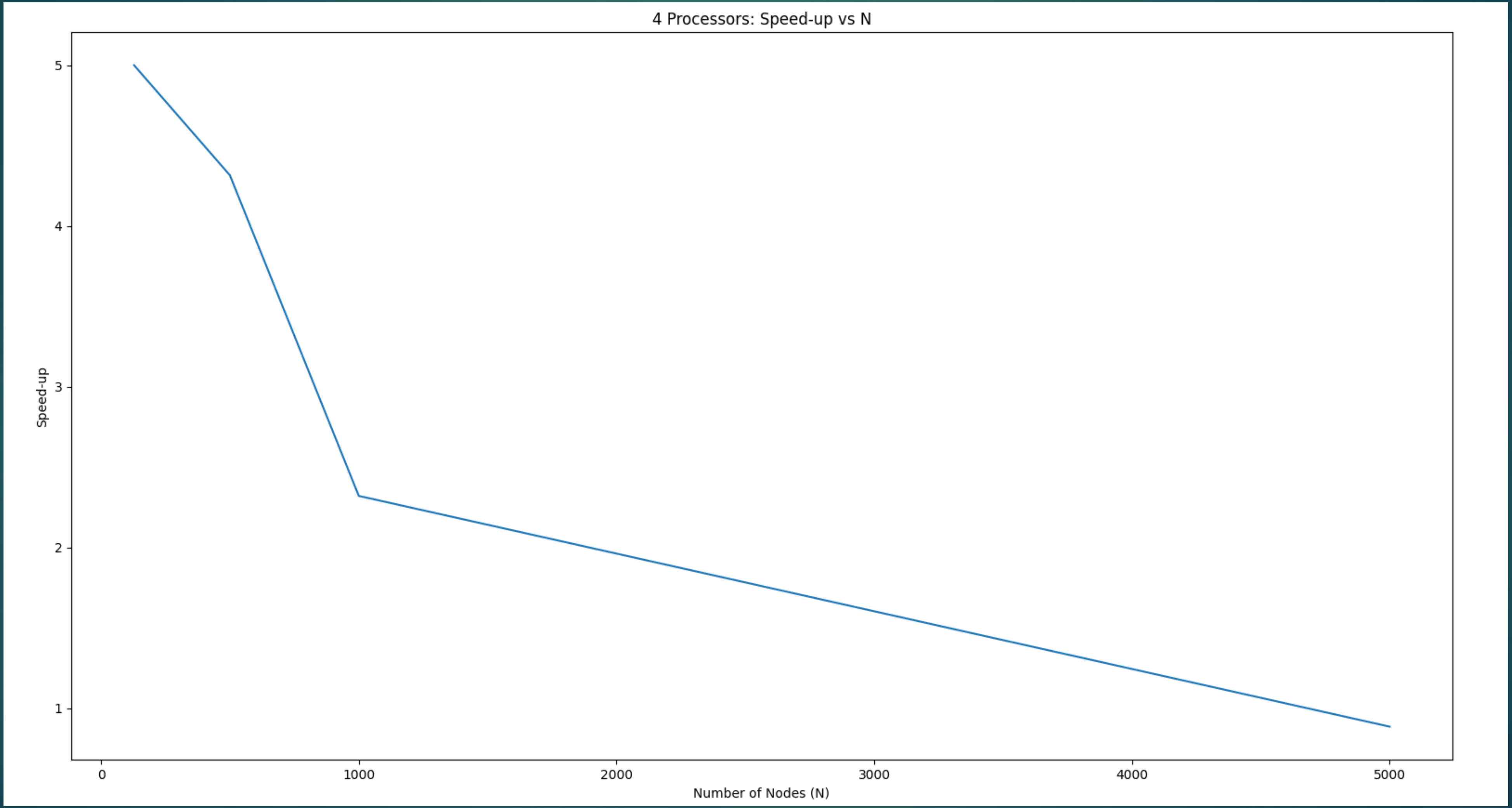
- **Synchronization Overhead:** Parallel execution requires proper synchronization mechanisms to maintain data consistency and avoid race conditions. Managing synchronization overhead is essential to ensure efficient parallelization.

- **Load Balancing:** Distributing workload evenly among threads is crucial for optimal performance. Load balancing considerations involve partitioning the graph efficiently and minimizing idle time among threads to maximize utilization of resources.

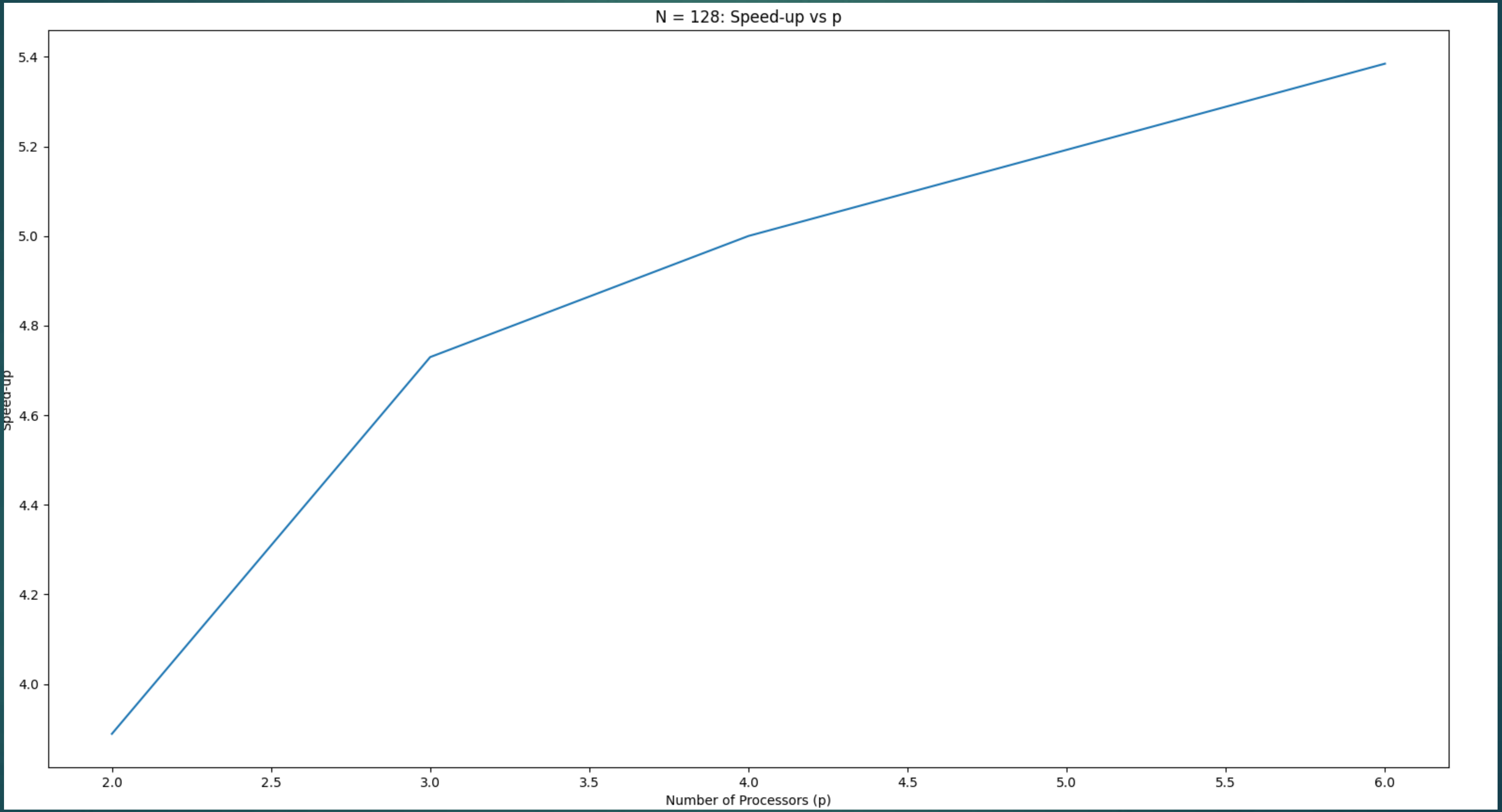
Performance Metrics



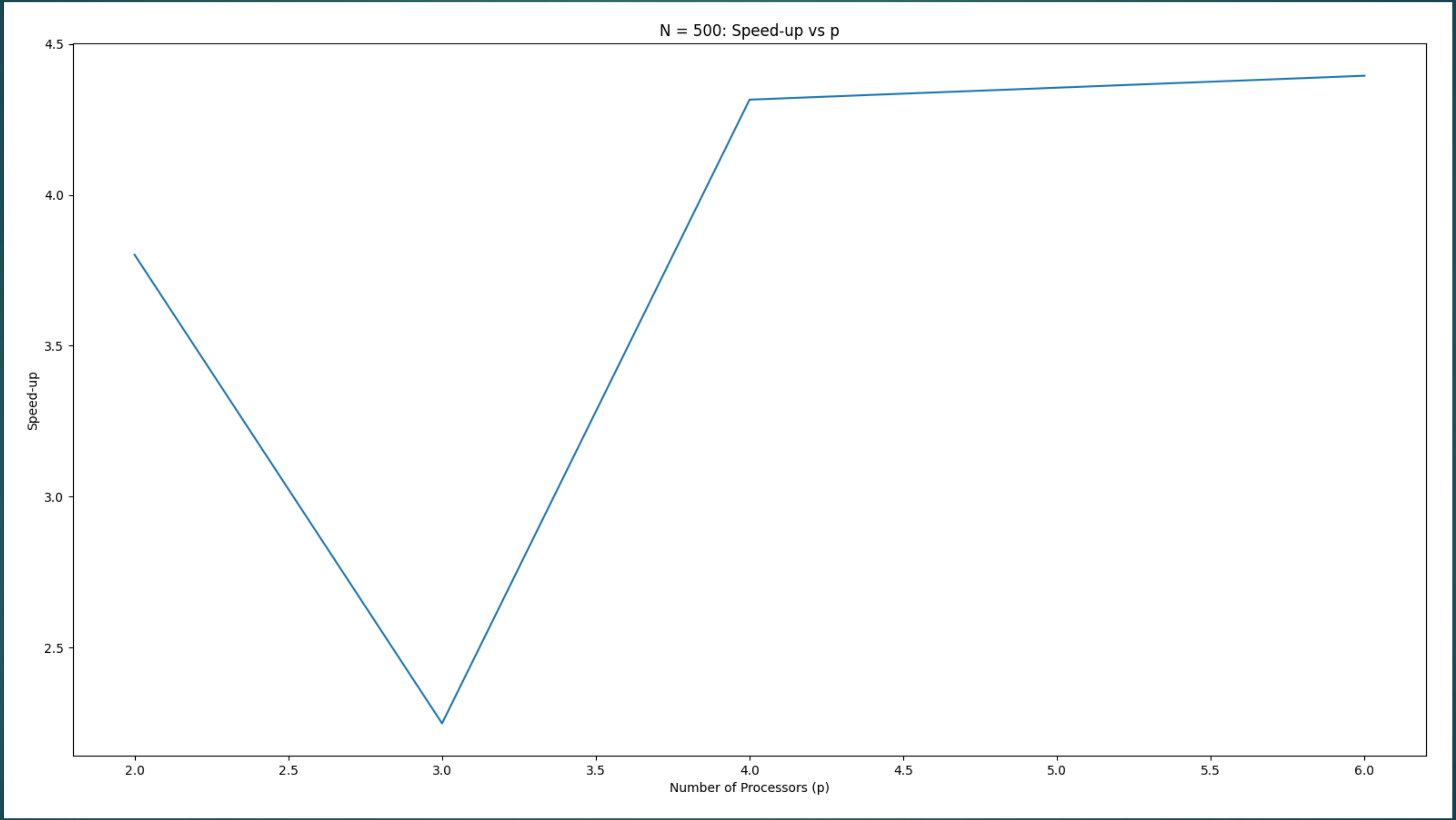
Performance Metrics



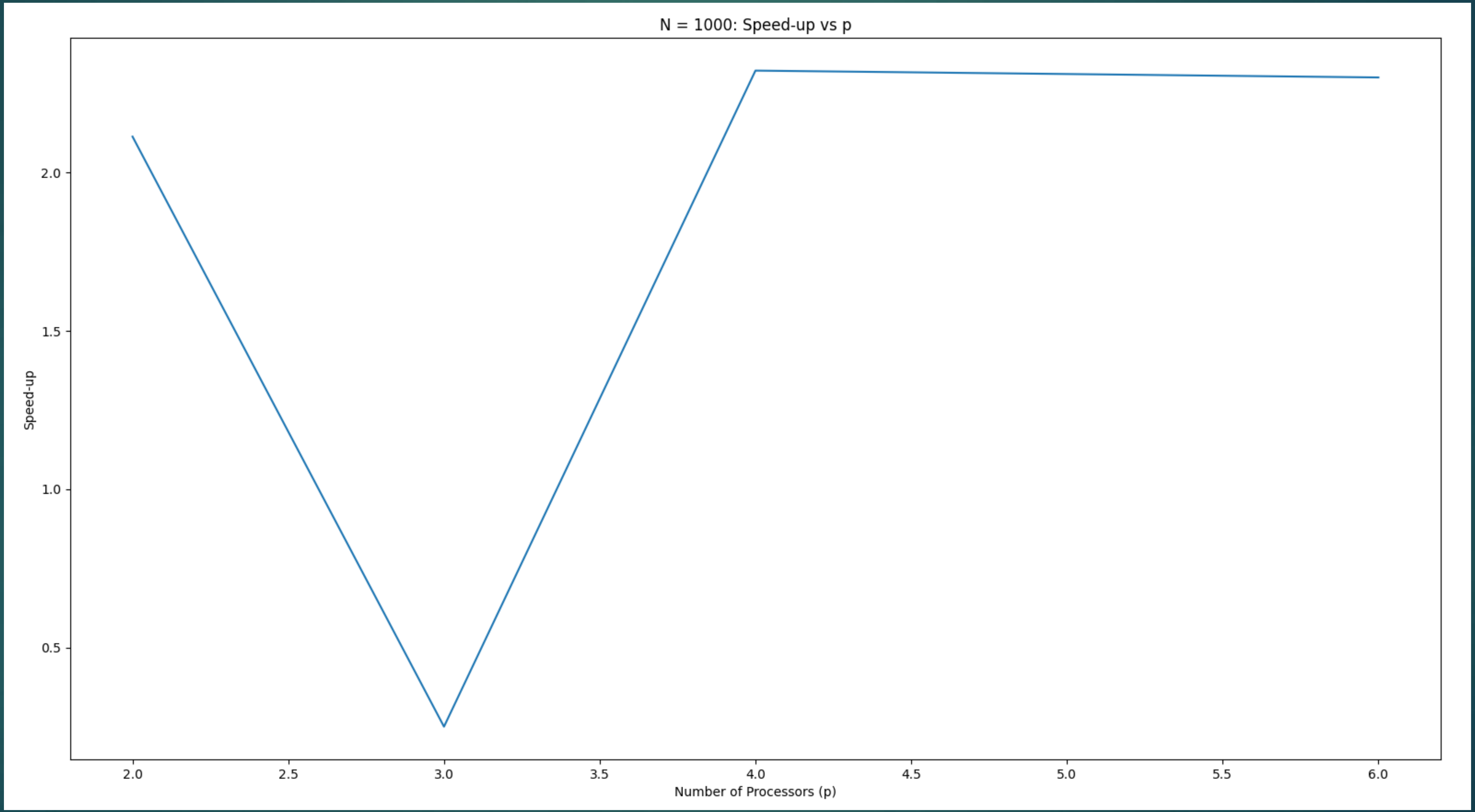
Performance Metrics



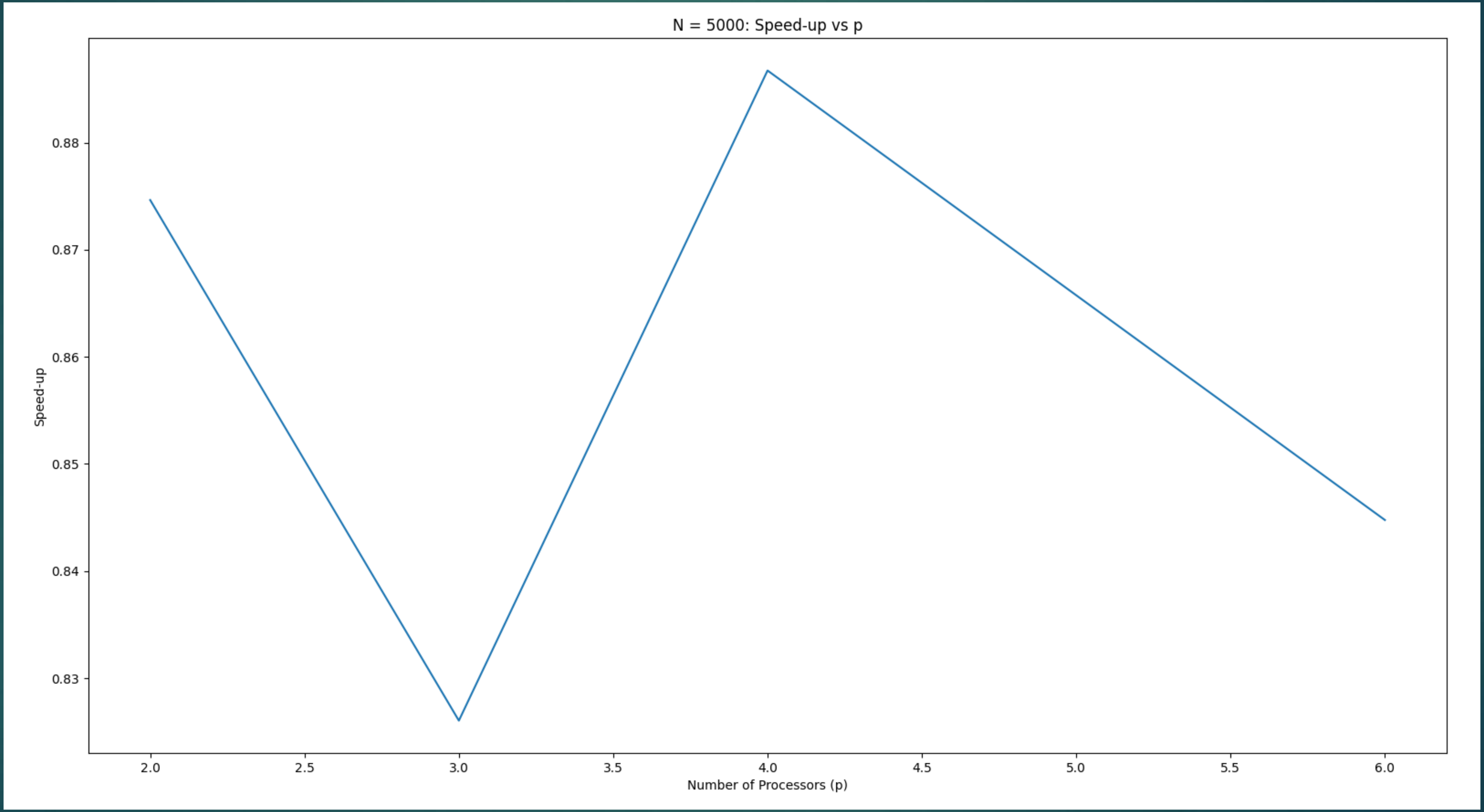
Performance Metrics



Performance Metrics



Performance Metrics



Advantages of using HPC Architectures

- Parallelism: Harnesses multiple processors for simultaneous computation.
- Scalability: Scales efficiently with problem size, accommodating larger graphs.
- Performance: Utilizes optimized hardware for faster execution.

Advantages of using HPC Architectures

- Resource Efficiency: Maximizes resource utilization, minimizing idle time.
- Large-Scale Problem Solving: Handles massive graphs with ease.
- Parallel Libraries: Access to MPI, OpenMP, CUDA for enhanced performance.

References

- Parallelization of Minimum Spanning Tree Algorithms
- Parallel Algorithms for Minimum Spanning Trees
- Parallel Programming with Prim’s Algorithm
- PES University Parallel Prim's Algorithm