CS22B1082
Thallapally Nimisha

Parallel PageRank Algorithm Report

1. Introduction

This report investigates the parallelization of the **PageRank algorithm** using OpenMP. The goal is to evaluate the performance of the parallel implementation with increasing numbers of threads, using **execution time, speedup, and parallel fraction** as key performance metrics.

The algorithm was implemented using two different graph representations:

- **Adjacency Matrix**
- **Adjacency List**

The performance was analyzed for a large graph with **50,000 nodes and 1,000,000 edges**, focusing on how thread count affects execution time and efficiency.

2. Methodology

Parallel Implementation using OpenMP

Graph Representations:

- **Adjacency Matrix:** Uses an NxN matrix, leading to high memory usage.
- **Adjacency List:** Stores only edges, reducing memory and improving performance.

3. Performance Evaluation

The performance was evaluated based on:

- **Execution Time:** Time taken to compute PageRank with different thread counts.
- **Speedup**
- **Parallel Fraction**
- Tests were conducted with thread counts: **1, 2, 4, 6, 8, 10, 12, 16, 20, 32, and 64**.

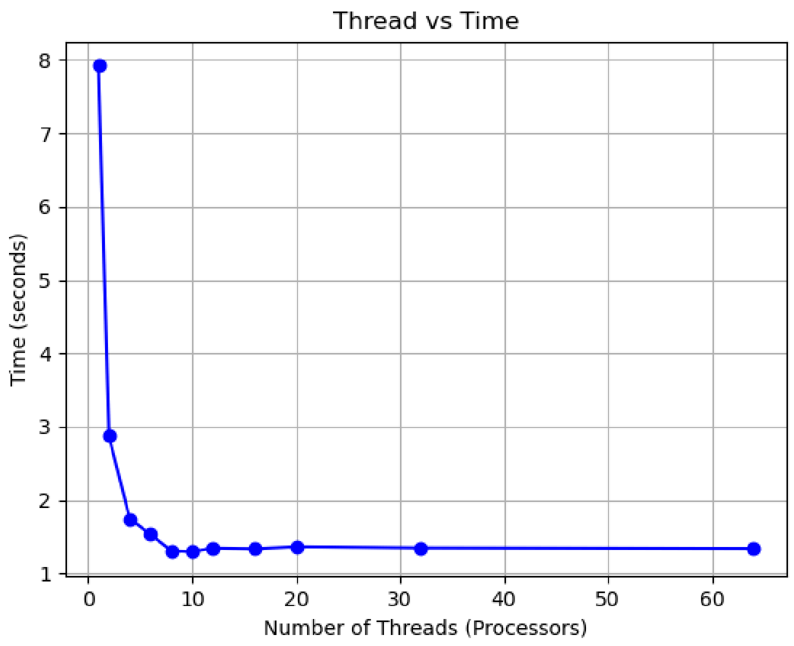
4. Results

4.1. Threads vs Time

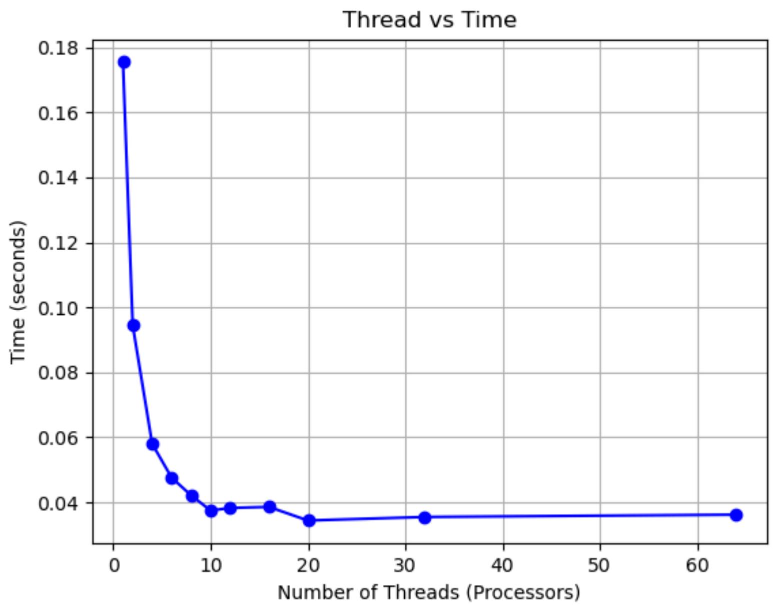
The execution times for different numbers of threads were recorded. The results show that execution time decreases as more threads are used, but the improvement slows down after a certain number of threads.

Graph: Threads vs Time

Using Adjacency List:



Using Adjacency Matrix:



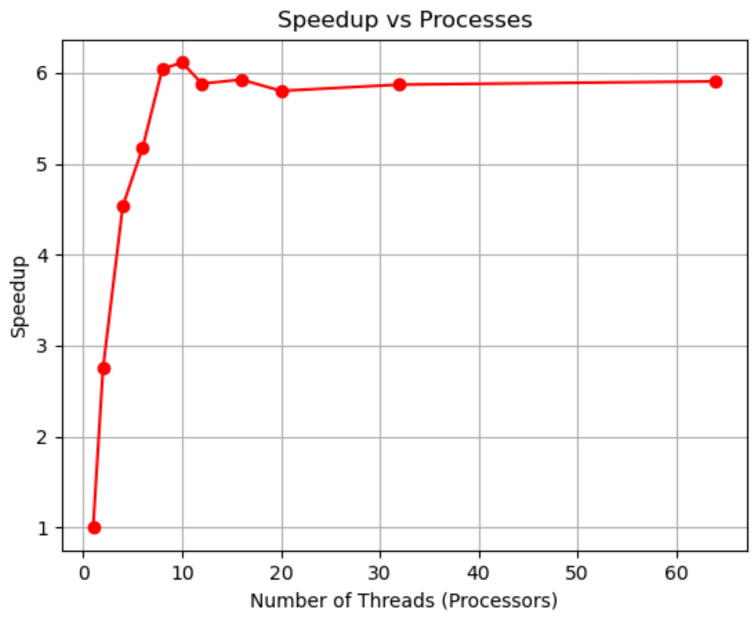
Observations:

- Execution time **decreases significantly** up to **8-10 threads**.
- Beyond **10 threads**, the reduction in execution time **slows down** due to **synchronization overhead**.
- **Adjacency List** performs significantly better than **Adjacency Matrix** in terms of time efficiency.

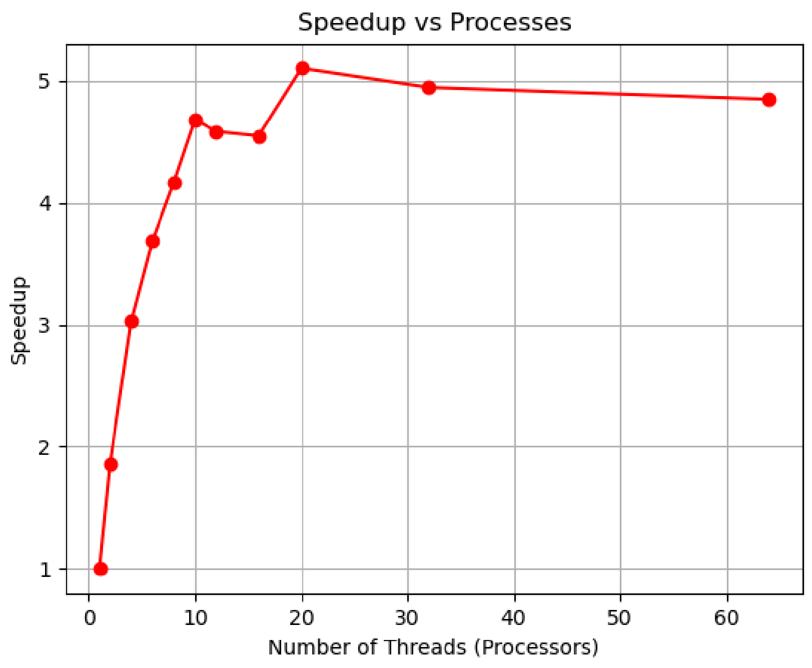
4.2. Speedup vs Processors

Speedup was calculated for each configuration, showing how parallelization benefits from increasing thread counts.

Graph: Speedup vs Processors
Using Adjacency Matrix:



Using Adjacency List:



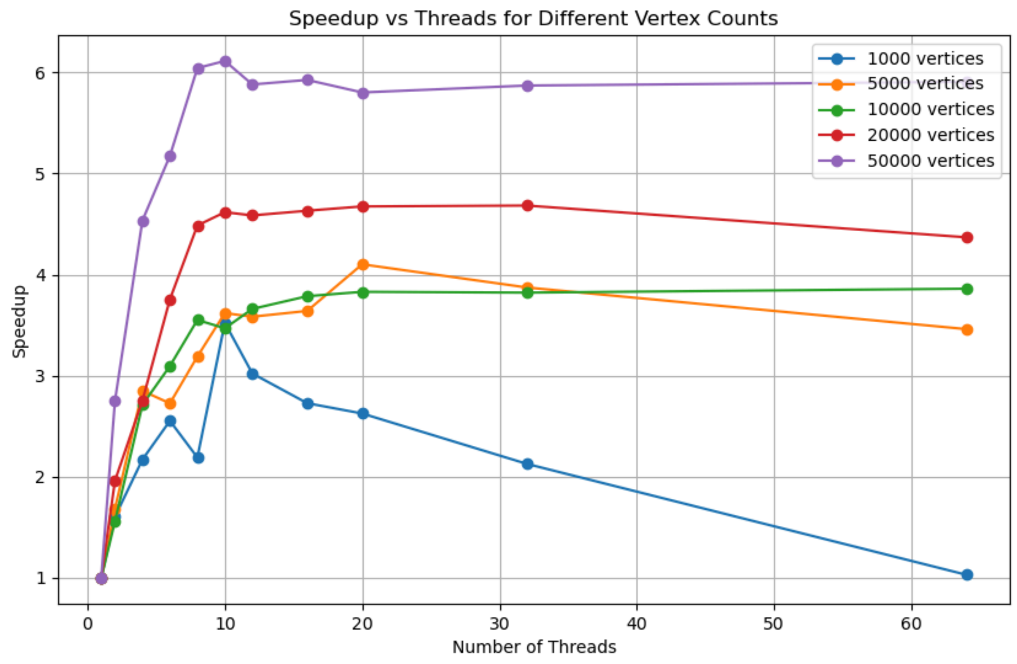
Observations:

- **Maximum Speedup:**
  - For **Adjacency Matrix**, the highest speedup was **6.04 at 8 threads**.
  - For **Adjacency List**, the highest speedup was **5.10 at 20 threads**.
- **Diminishing Returns:** Beyond **10-12 threads**, additional threads contribute **less** to performance improvement.

4.3 Plot for Varying Graph sizes

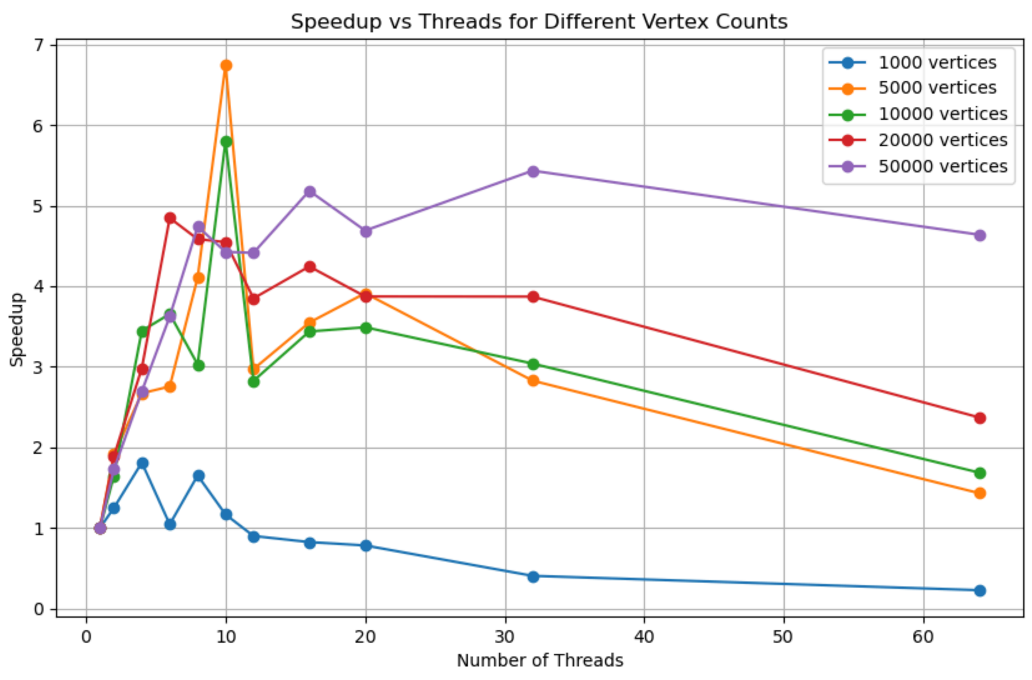
Using Adjacency Matrix:

CS22B1082
Thallapally Nimisha



CS22B1082
Thallapally Nimisha

Using Adjacency List:



5. Analysis

5.1. Speedup Observations

- Speedup **increases initially**, but slows down due to synchronization overhead.
- **Adjacency List scales better** than Adjacency Matrix due to lower memory usage.
- Beyond **10-12 threads**, adding more threads does not significantly improve speedup.

5.2. Observations on Parallel Fraction

- **Parallel fraction initially increases**, but plateaus beyond **8 threads**.
- Higher thread counts cause **synchronization overhead**, reducing the parallel fraction.
- For the **adjacency matrix**, the estimated parallel fraction **stabilizes around 0.85-0.95**, but performance degrades with more threads.
- For the **adjacency list**, the parallel fraction is slightly **higher (0.80-0.92)** and shows better **scalability**.

5.3. Amdahl's Law

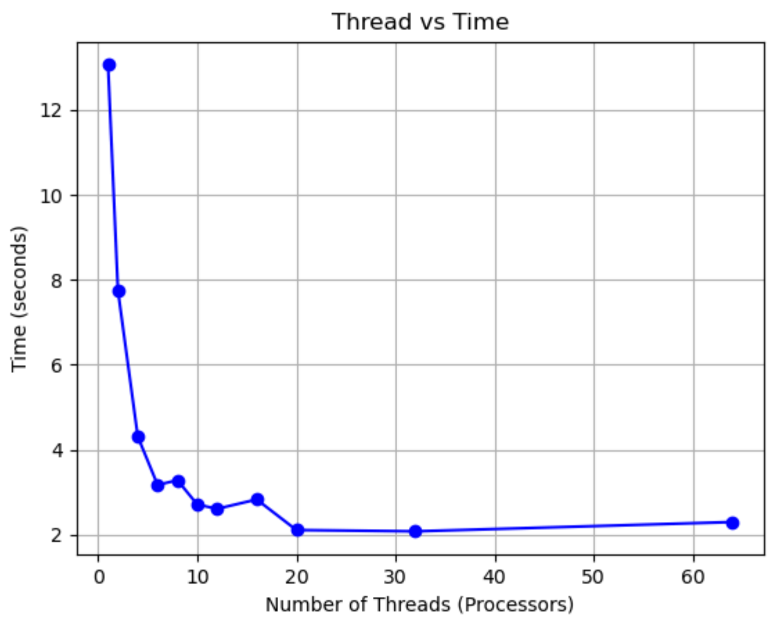
Both speedup and parallel fraction graphs follow **Amdahl's Law**

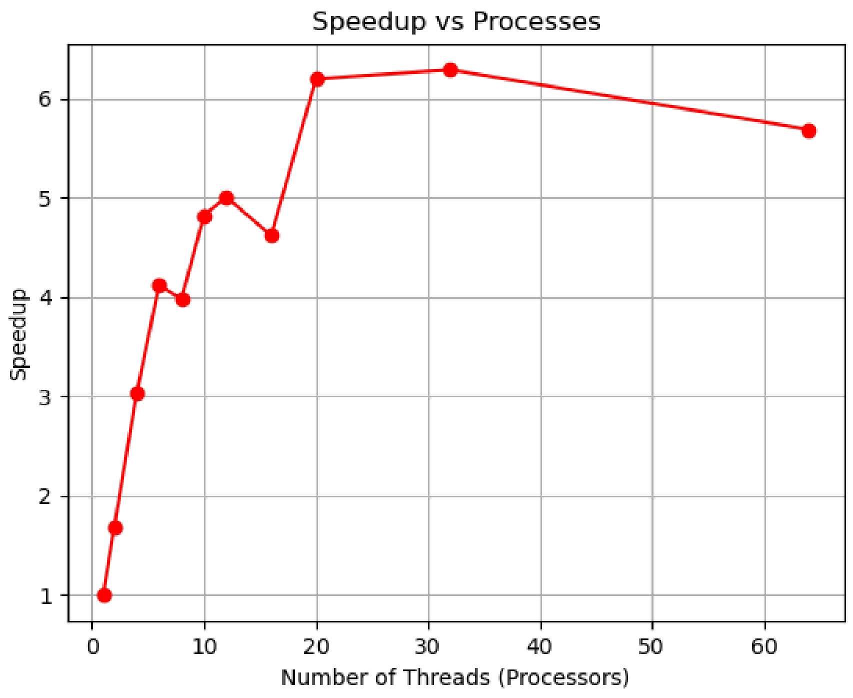
- **Adjacency List achieves better efficiency** due to lower memory usage and reduced overhead.
- **Adjacency Matrix reaches peak performance at ~8 threads**, after which additional threads contribute little to speedup.

For a Graph with 500000 Nodes and 5000000 edges:

This was only possible to implement using a Adjacency list

Using an adjacency matrix for PageRank can lead to high memory usage, especially for large graphs, since it requires $O(n^2)$ space. On the other hand, an adjacency list is much more efficient in terms of memory, especially for sparse graphs.





6. Conclusion

1. **Adjacency Matrix Approach:**
   - **High memory usage**, leading to process failures for large graphs.
   - Speedup **peaks at 6.04 with 8 threads**, but diminishes beyond this.
2. **Adjacency List Approach:**
   - **More memory-efficient** and scales better with increasing threads.
   - **Peak speedup: 5.10 at 20 threads**, showing better scalability than the matrix.
3. **Overall Performance Insights:**
   - **Parallelization improves efficiency**, but gains diminish beyond **10-12 threads**.
   - **Memory constraints** make adjacency matrices impractical for large graphs.
   - **Adjacency List is the recommended approach** for handling large-scale graphs.