Problem Analysis Document

Parallelization of Graph Algorithms: PageRank

Course: CS-478 — Design and Analysis of Algorithms

Submitted By:

Abdullah Noor - 2022029
Ali Vijdaan - 2022560
Hamza Amin - 2022378
Maaz Hamid - 2022655

December 24, 2025

Contents

1 Problem Definition

1.1 Introduction

The rapid expansion of the World Wide Web and Social Networks has generated graph datasets of unprecedented size. Classical algorithms designed for sequential execution on single-core processors fail to scale with these datasets. This project focuses on the **PageRank** algorithm, a link analysis method used to measure the relative importance of nodes in a graph.

1.2 The Core Problem

Calculating PageRank involves iterative matrix-vector multiplication. For a graph with millions of nodes, a sequential approach is inefficient due to:

1. **Computational Latency:** The time complexity grows linearly with the number of edges.
2. **Memory Access Latency:** Graph traversal involves random memory access (pointer chasing), which causes high cache miss rates (the "Memory Wall").

The objective is to design a parallel algorithm using **OpenMP** that reduces execution time while maintaining numerical correctness.

2 Mathematical Modeling

2.1 Graph Representation

We model the web as a directed graph $G = (V, E)$, where:

- V is the set of web pages (Nodes).
- E is the set of hyperlinks (Edges), where $(u, v) \in E$ denotes a link from u to v.

2.2 The Random Surfer Model

PageRank assumes a "random surfer" who starts at a random page and either:

1. Clicks a link on the current page with probability d (damping factor, typically 0.85).
2. Jumps to a random page in the graph with probability $1 - d$.

2.3 The Equation

The rank of a page u, denoted $PR(u)$, is defined recursively:

$$PR(u) = \frac{1-d}{N} + d \sum_{v \in In(u)} \frac{PR(v)}{OutDegree(v)} \tag{1}$$

Where:

- $N = |V|$ is the total number of nodes.
- $In(u)$ is the set of pages pointing to u.
- $OutDegree(v)$ is the number of outgoing links from v.

This system of equations can be viewed as the stationary distribution of a Markov Chain. We solve it using the **Power Iteration Method** until the error converges below a threshold ϵ.

3 Dataset Analysis: web-Stanford

We have selected the 'web-Stanford' dataset from the SNAP repository. Understanding its topology is crucial for parallel design.

3.1 Statistical Profile

- **Nodes:** 281,903
- **Edges:** 2,312,497
- **Average Degree:** ≈ 8.2
- **Type:** Directed, Unweighted.

3.2 The Sparsity Challenge

A naive approach would represent the graph as an $N \times N$ Adjacency Matrix.

$$\text{Memory} = N^2 \times 4 \text{ bytes} \approx (281,903)^2 \times 4 \text{ B} \approx 317 \text{ GB}$$

This is infeasible for standard RAM. However, the graph is **Sparse**. The density is:

$$\text{Density} = \frac{|E|}{|V|^2} \approx \frac{2.3 \times 10^6}{7.9 \times 10^{10}} \approx 0.000029$$

Over 99.99% of the matrix entries are zero. This necessitates the use of **Adjacency Lists** or **Compressed Sparse Row (CSR)** formats.

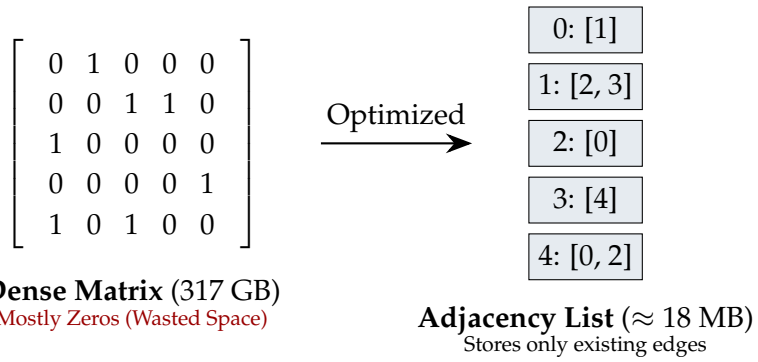

Figure 1: **Memory Model Comparison.** We must reject the Matrix approach due to the high sparsity of the web-Stanford dataset.

3.3 Power-Law Distribution

Real-world web graphs are scale-free. A small number of nodes (Hubs) have massive degrees, while the majority have very few.

- **Implication for Parallelism:** If we divide work simply by node count (e.g., Thread 1 gets nodes 0-100, Thread 2 gets 101-200), the thread processing the "Hub" node will take much longer.
- **Requirement:** The algorithm requires **Dynamic Load Balancing**.

4 Sequential Baseline Analysis

4.1 Sequential Algorithm Logic

```
For iter = 1 to k:
    For each node u in V:
        sum = 0
        For each neighbor v pointing to u:
            sum += Rank[v] / OutDegree[v]
        NewRank[u] = (1-d)/N + d * sum
    Rank = NewRank
```

4.2 Complexity Analysis

- **Outer Loop:** Runs k times (iterations).
- **Inner Loop:** Iterates over all vertices $|V|$.
- **Deepest Loop:** Iterates over incoming edges. Across all vertices, this touches every edge exactly once per iteration.

$$T_{seq} = \mathcal{O}(k \cdot (|V| + |E|))$$

For 'web-Stanford' ($|E| \approx 2.3M$), assuming $k = 50$:

$$\text{Operations} \approx 50 \times 2,300,000 \approx 1.15 \times 10^8 \text{ Ops}$$

While this seems manageable, the bottleneck is not the arithmetic operations, but the **Memory Bandwidth** required to fetch neighbor data randomly from RAM.

5 Proposed Parallel Strategy

To address the limitations of the sequential model, we propose a shared-memory parallel approach.

5.1 Approaches Considered

1. **MPI (Distributed):** Good for graphs larger than RAM. *Verdict: Overkill for 281k nodes.*
2. **CUDA (GPU):** High throughput, but limited memory on standard GPUs. *Verdict: Future scope.*
3. **OpenMP (Shared Memory):** Ideal for multi-core CPUs where the graph fits in RAM. **Selected Approach.**

5.2 Anticipated Challenges

- **Race Conditions:** If multiple threads update the rank of a neighbor simultaneously (Push model).
- **Solution:** Use a **Pull-based** model where a thread reads neighbors and writes exclusively to its own node.
- **False Sharing:** If adjacent nodes in the array are updated by different threads, cache lines may be invalidated.
- **Load Imbalance:** Due to the Power-Law distribution described in Section 3.3.