Connected Components of an Undirected Graph (MPI)

Parallel and Distributed Systems (Homework 2)

Apostolos Pantazis **AM:** 10910

December 2025

Abstract

This report presents a hybrid parallel implementation of the Label Propagation algorithm designed to identify connected components in massive, billion-scale graphs. By integrating **MPI** for distributed-memory communication and **Cilk** for shared-memory task parallelism, the system utilizes the aggregate memory of high-performance computing clusters. A primary technical challenge addressed is the 32-bit MPI limit; a chunked broadcasting mechanism was developed to successfully process the `com-Friendster` dataset, which contains over 3.6 billion edges. Experimental results demonstrate that while the hybrid model enables processing of massive graphs, performance is frequently constrained by memory bandwidth saturation and the communication overhead of global synchronization.

1 Introduction

The identification of Connected Components (CC) is a fundamental problem in graph theory, essential for understanding the structure of massive networks such as social graphs and web link structures. In an undirected graph, a connected component is a maximal subgraph in which any two vertices are connected by paths. As the scale of these graphs reaches billions of edges, a single machine's memory and computational throughput are no longer sufficient to maintain the graph in-core or process it within a reasonable timeframe.

This project implements a hybrid parallel algorithm for finding connected components. The implementation centers on solving the "memory wall" by partitioning the graph across several nodes. By integrating MPI (Message Passing Interface) for distributed-memory communication and Cilk for intra-node multi-threaded task parallelism, the implementation can efficiently utilize a high-performance cluster. This report explores the algorithm's design, the technical challenges associated with 64-bit data scales, and an analysis of performance across the `com-Friendster` and `mawi_2015` datasets.

2 Algorithm Description

2.1 The Label Propagation Model

The core algorithm is based on Label Propagation, often referred to as ”Coloring.” The logic is highly suitable for parallelization because each vertex can be updated independently. Each vertex v is assigned a unique label $L(v) = ID(v)$. In each iteration, every vertex v updates its current label by taking the minimum value of its own label and the labels of all its neighbors $u \in N(v)$. The process is repeated until no vertex in the entire graph updates its label. At convergence, all vertices in a component share the same minimum vertex ID.

2.2 The Hybrid MPI-Cilk Approach

To handle billions of edges, a two-tier parallel approach is used. At the MPI level, vertices are partitioned into contiguous blocks, allowing the memory footprint to be distributed across multiple nodes. At the Cilk level, the update loop for the local partition is parallelized using `cilk_for`. Cilk’s work-stealing scheduler prevents ”straggler” threads from slowing down the rank by redistributing vertex updates dynamically across available cores.

3 Technical Challenges: The 32-bit MPI Limit

A significant hurdle in processing `com-Friendster` is its size. With 3.6 billion edges, the dataset exceeds the maximum value of a 32-bit signed integer ($\approx$ 2.14 billion). Standard MPI functions, such as `MPI_Bcast`, use an `int` for the element count, which leads to integer overflow. I implemented a **Chunked Broadcast** routine to bypass this. The edge list is partitioned into segments of 500 million elements, which are broadcast sequentially using 64-bit offsets to track positions.

```c
void BroadcastGraph(Graph** g_ptr, int rank) {
    long long offset = 0;
    int chunk = 500000000;
    while (offset < num_edges) {
        int to_send = (num_edges - offset > chunk) ? chunk : (int)(
            num_edges - offset);
        MPI_Bcast(&((*g_ptr)->edges[offset]), to_send, MPI_INT, 0,
            MPI_COMM_WORLD);
        offset += to_send;
    }
}
```

Listing 1: Chunked Broadcast for Billion-Scale Data

4 Implementation Details

4.1 Graph Storage (CSR Format)

The graph is stored in Compressed Sparse Row (CSR) format. This format is highly efficient for Label Propagation as it allows for sequential access to a vertex's neighbors, maximizing cache line utilization. Given the dataset size, the `num_edges` and `offsets` variables are stored as `long long` to prevent overflow.

4.2 Synchronization and Global Convergence

After each local Cilk-parallelized update phase, the ranks synchronize using `MPI_Allgatherv`, where ranks broadcast their updated labels to all other nodes. Global convergence is determined via `MPI_Allreduce`; an `Allreduce` with the `MPI_MAX` operator checks if any rank performed an update. If the result is 0, the cluster terminates the loop.

5 Experimental Results

The following tables detail the runtime performance on the cluster. Note that the execution time captures the algorithm duration, excluding the MTX file loading time.

Nodes	Cilk Workers per Rank	Dataset	Time (s)
1	8	`com-Friendster`	35.7717
1	32	`com-Friendster`	38.0841
1	64	`com-Friendster`	38.8151
2	64	`com-Friendster`	38.9292
4	64	`com-Friendster`	37.0405

Table 1: Friendster Results (65.6M Nodes, 3.6B Edges)

Nodes	Cilk Workers per Rank	Dataset	Time (s)
1	8	`mawi_2015`	1.8962
1	16	`mawi_2015`	1.9951
1	32	`mawi_2015`	2.0487
1	64	`mawi_2015`	2.0604
2	64	`mawi_2015`	2.2431

Table 2: Mawi Results (226M Nodes, 480M Edges)

6 Performance Analysis

6.1 Communication Bottlenecks

For `mawi_2015`, increasing resources led to an increase in execution time. In this graph with 226 million nodes, the label array is nearly 1 GB. Because the algorithm converges very quickly on this graph structure, the time spent on `MPI_Allgatherv` to synchronize that 1 GB array across the network exceeds the actual computation time. This suggests that synchronization overhead can dominate execution for sparse graphs.

6.2 Memory Bandwidth Limitations

On `com-Friendster`, performance was consistent regardless of the number of Cilk workers. This suggests the algorithm is memory-bound. Once the memory bus is saturated, adding more Cilk workers does not reduce execution time; workers simply wait for data from RAM. Irregular memory access patterns common in graph algorithms further exacerbate this by causing high cache miss rates.

7 Conclusion

This report demonstrated a hybrid parallel implementation for finding connected components. The chunked broadcast mechanism allowed for the processing of 3.6 billion edges, bypassing standard MPI limitations. Results indicate that performance is primarily dictated by memory bandwidth and network interconnect speeds rather than raw CPU core counts.

You can find all the necessary files for this project on this GitHub repository: `https://github.com/pantazisa/Homework`