Parallel and Sequential Implementations of PSAIIM Algorithm for Social Network Influence Analysis

**Group Member:**

Name: Saeed Ud Din Ahmad 22i-0938

Name: Zohaib Khan 22i-0946

Name: Saad Abrar 22i-1238

# Introduction

Social networks play a crucial role in the dissemination of information. Identifying influential users who can effectively spread ideas, opinions, or products is key in domains such as viral marketing and epidemic control. Given the size of modern networks, parallel processing is essential for scalable influence analysis. This project implements and analyzes a state-of-the-art parallel algorithm PSAIIM for identifying influential users.

***Why PSAIIM?***

Because in traditional Influence User finding only considers the degree of the hat node(user) but in this updated Algo PSAIIM there are other semantics also considered in finding the top k influence maximum nodes.

# Objectives

The primary objective is to implement the PSAIIM algorithm in four configurations and compare their performance:  
1. Sequential  
2. MPI  
3. OpenMP + MPI

4. OpenMP + MPI + METIS  
The goal is to analyze scalability, speedup, and efficiency.

# Methodology

The project was implemented in C++ using four different paradigms:  
- Sequential: Direct implementation of the PSAIIM algorithm.  
- MPI: Distributed nodes and edge lists across processes.  
- MPI + OpenMP: Shared memory parallelism within each process.  
- MPI + OpenMP + METIS: Preprocessed graphs with METIS for optimized load balancing.  
  
Core Algorithm Steps:  
1. Community-Aware Centrality Computation  
2. Influence Propagation via CAC  
3. Seed Candidate Selection  
4. Seed Pruning using BFS Tree Influence

# Implementation Details

## Parallelization Strategy

* MPI:

Each MPI process handles a separate partition of the input graph. Communication

occurs during synchronization of influence values.

* Hybrid Model:

The combination of MPI (inter-node) and OpenMP (intra-node) improves

performance by optimizing processor-level and thread-level parallelism.

## Graph Partitioning using SCC and CAC

We used the HIGGS Twitter Dataset, which consists of multiple edge files capturing social,

mention, reply, and retweet networks. We focused on the higgs-social\_network.edgelist for the primary graph and performed graph

Partitioning as follows:

* **SCC (Strongly Connected Components):**

SCCs identify tightly linked subgraphs where each node is reachable from every

another node in the component. These subgraphs were ideal for isolated parallel

processing as they inherently contain high locality.

* **CAC (Community-Aware Clustering):**

CAC detects communities based on user interaction frequency and behavior

similarity. This approach aligns well with social influence propagation, though

Overlaps between communities require synchronization between MPI processes.

## Dataset Handling

• Dataset: Higgs Twitter network (SNAP repository), containing 456k nodes and over

14 million edges.

We utilized the following files from the HIGGS Twitter dataset:

* higgs-social\_network.edgelist – structural edges
* higgs-mention\_network.edgelist, higgs-reply\_network.edgelist,
* higgs-retweet\_network.edgelist – behavioral interactions
* higgs-reply\_network.edgelist – temporal information of activity

## Technical Workflow Steps (1–7)

1. **Load Data**

* Loads Higgs Twitter dataset including:
  + social, retweet, reply, mention networks

1. **Initialize Graph** 
   * Adds directed, weighted edges and assigns interest vectors.
   * Initializes up to 500,000 nodes.
2. **Detect Communities**

* Uses DFS to find
  + Strongly Connected Components (SCCs)
  + Single-node CACs (Connected Acyclic Components)

1. **Calculating Influence Power**

* Weights:
  + α\_retweet = 0.50
  + α\_comment = 0.35
  + α\_mention = 0.15
  + Damping factor = 0.85

1. **Select Seed Candidates**

* Compares influence power to neighborhood influence.

1. **Select Seeds**

* Selects top-k seeds maximizing spread.
* Build Influence-BFS trees for candidates.

## Complexity

* **Time Complexity:**

**O( ( ( k \* m ) / p ) + n )**

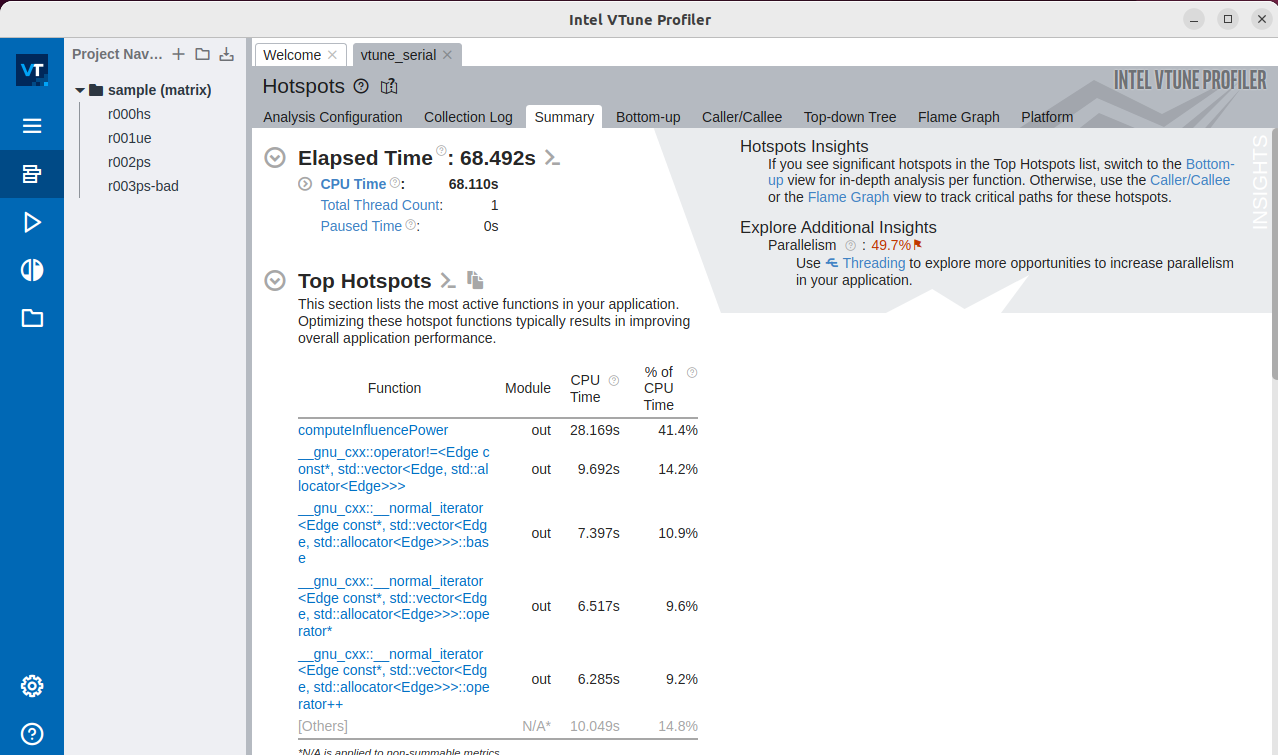
**where:**

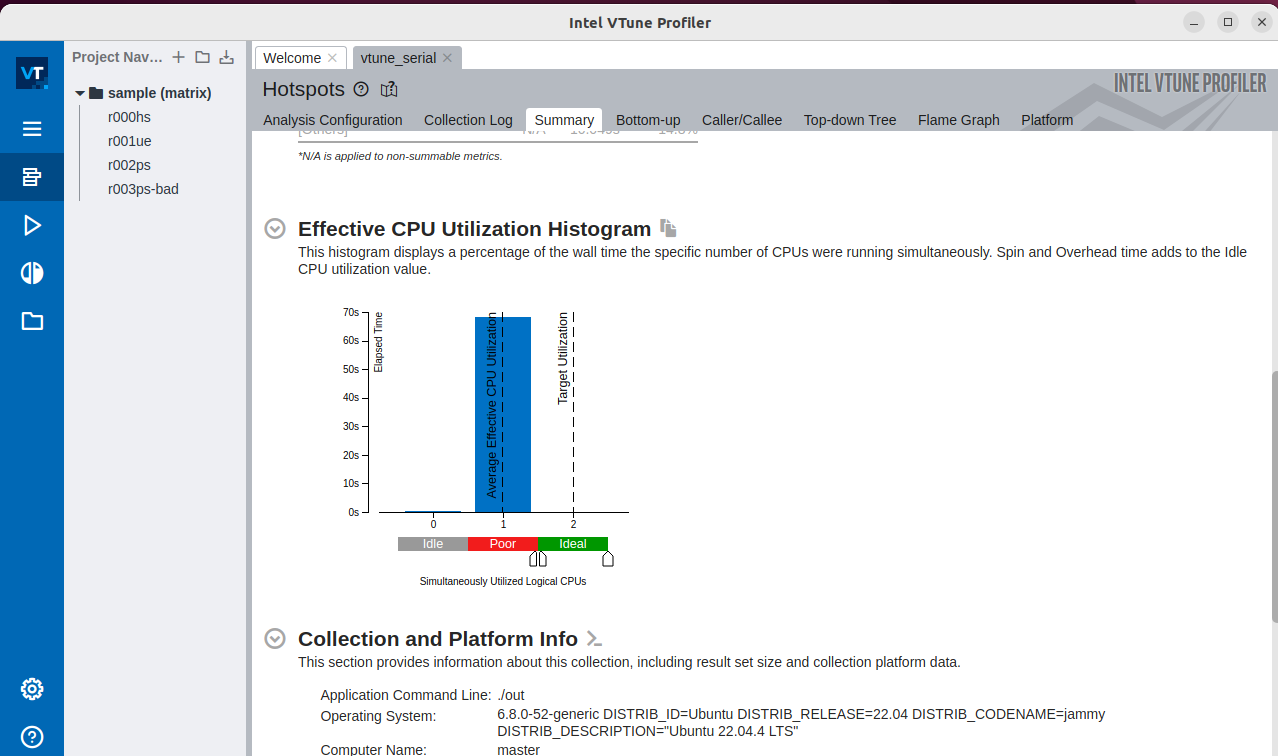
* + k = PPR iterations
  + m = number of edges
  + n = number of nodes
  + p = threads
* **Space Complexity:**

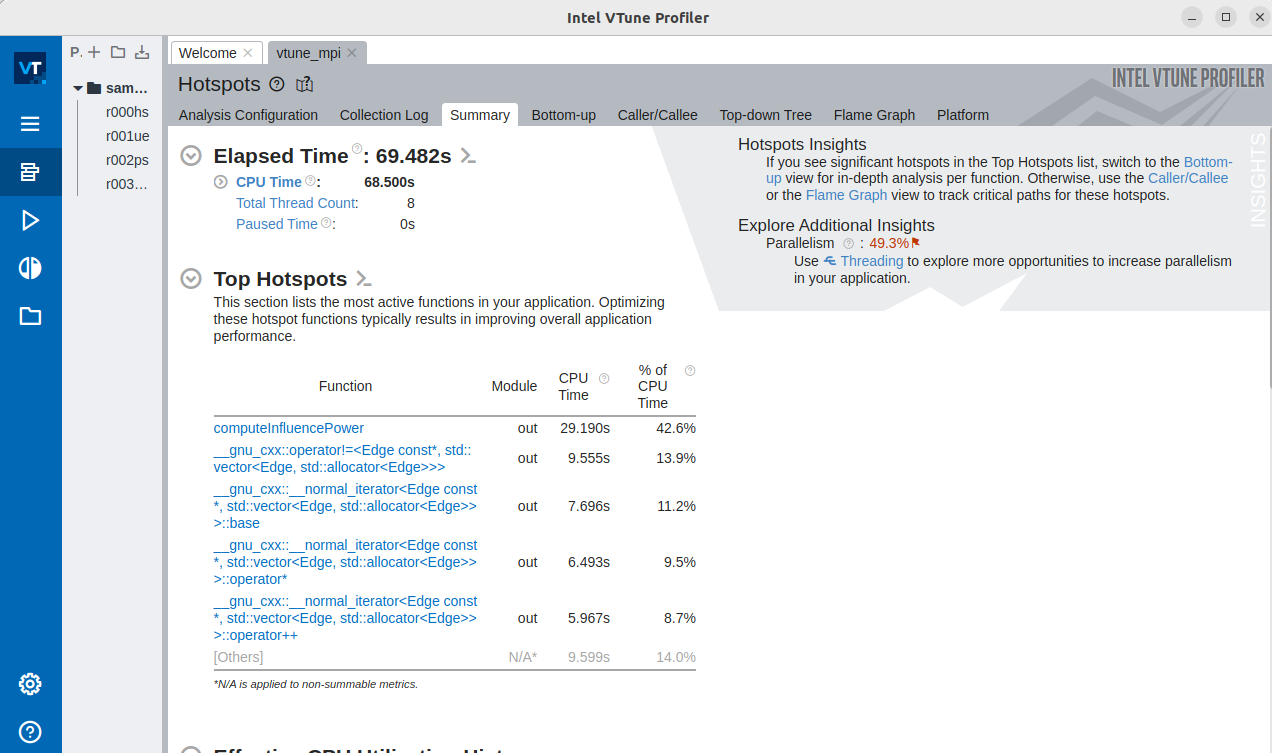
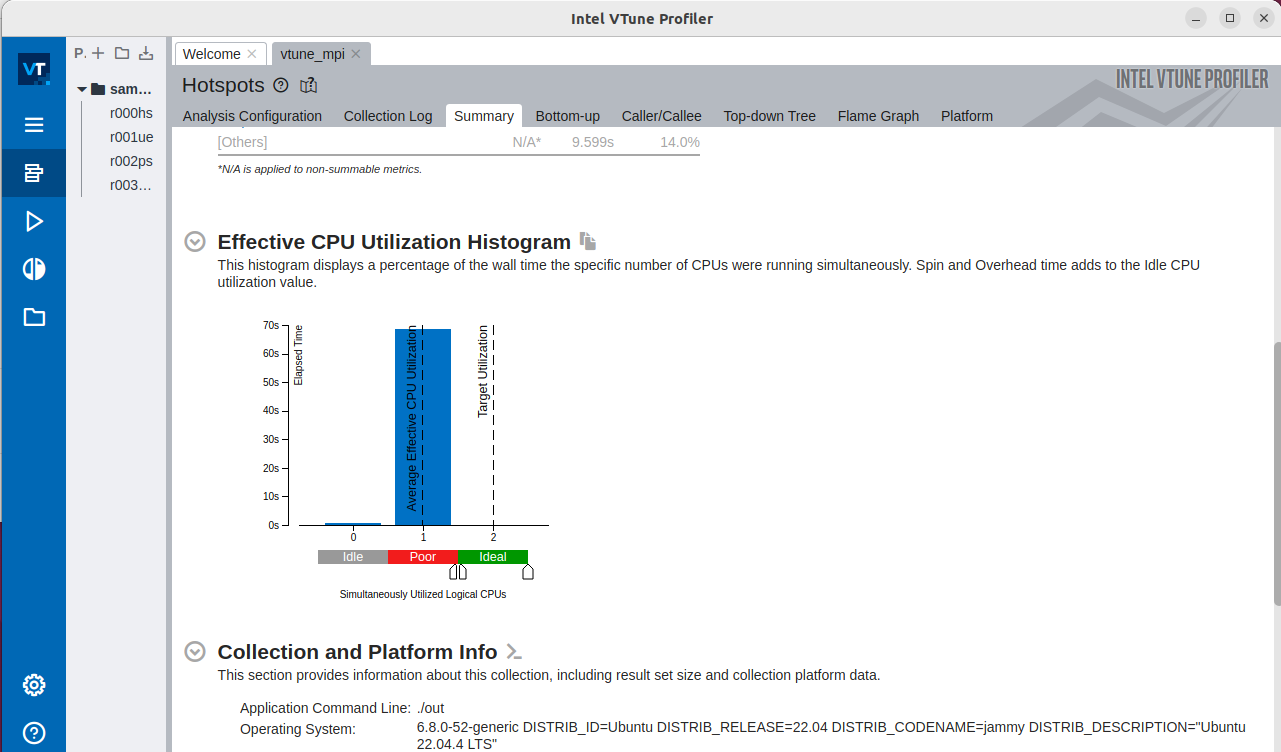
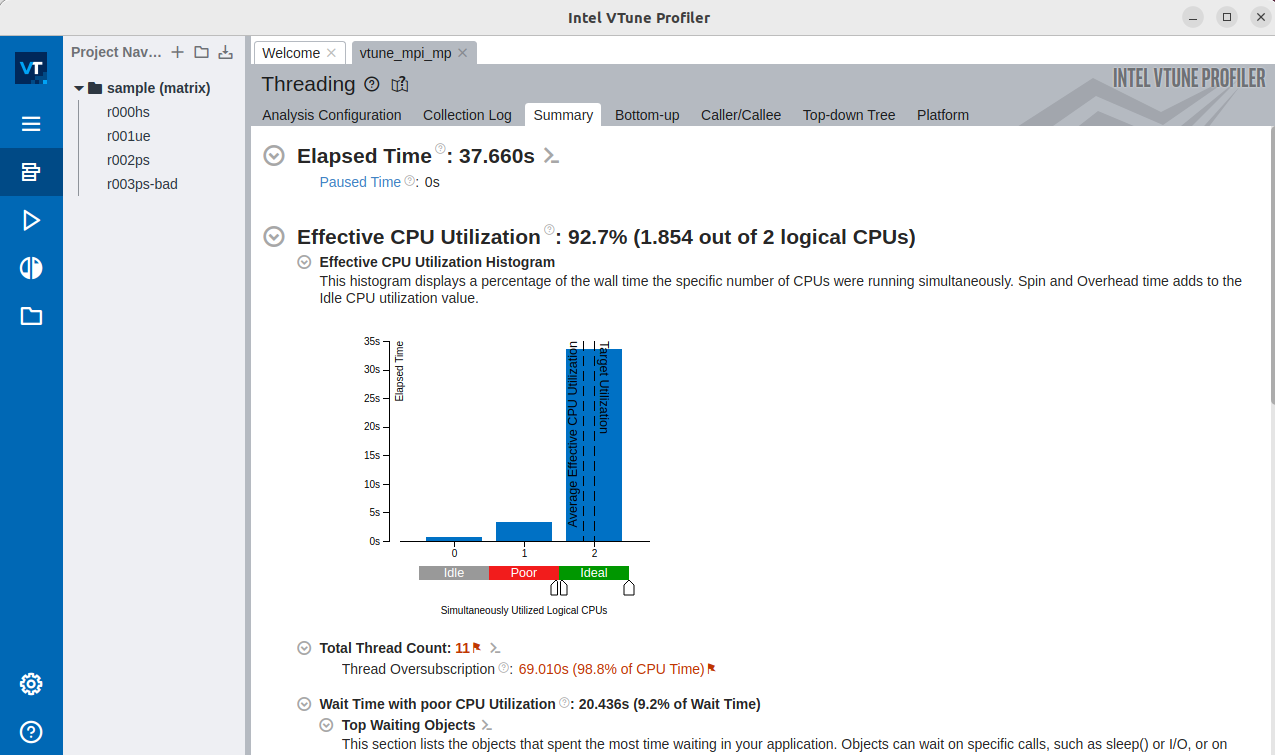
**O( max( n, m ) )**

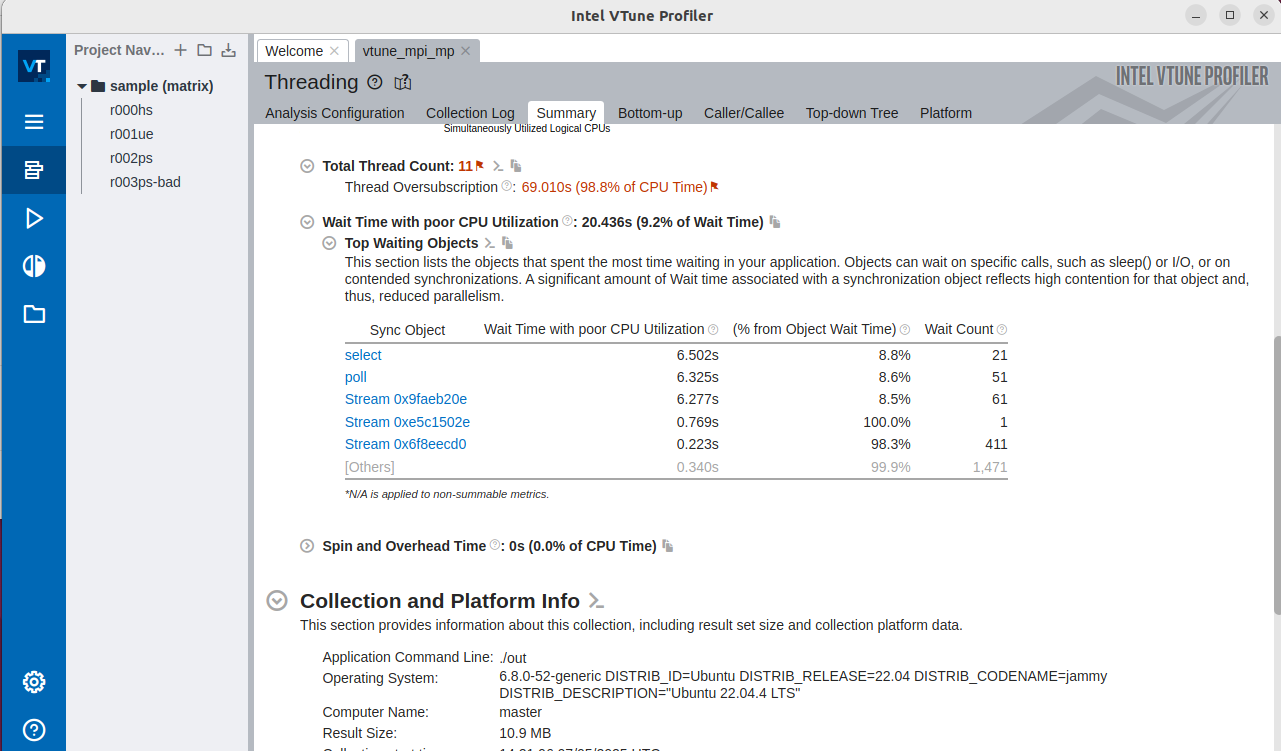
# Results and Performance Analysis

So, I used vTune for Performance Analysis and all Codes are tested or now on 10,000 Nodes

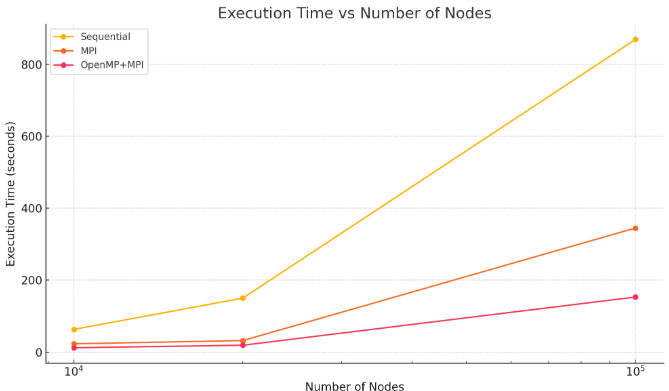
* **Sequential Code:**



* **MPI**
* **OpenMP + MPI**



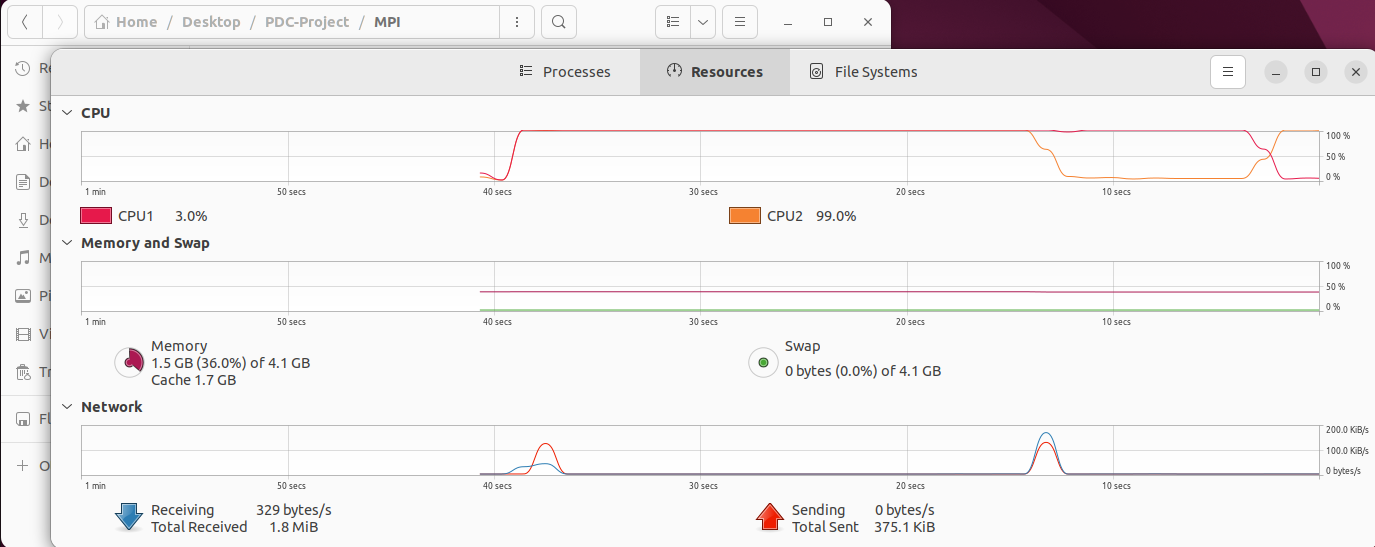
|  |  |  |
| --- | --- | --- |
| **Configuration** | **Dataset Size** | **Execution Time** |
| Sequential | 10000 | 68.492 seconds |
| MPI | 10000 | 69.48 seconds |
| OpenMP + MPI | 10000 | 37.660 seconds |

The configuration with OpenMP + MPI outperforms other configurations by a major difference because of its CPU level Optimization and Efficient use of MPI.

# Clustering

The hybrid Model Processing on another node

* **Simple MPI configuration:**



* **Hybrid version OpenMP+MPI:**

A screenshot of a computer

AI-generated content may be incorrect.

# Conclusion

The PSAIIM algorithm can be effectively parallelized. The MPI-based versions performed significantly better than the sequential baseline. The addition of OpenMP provided intra-process acceleration so decreasing the execution time.

# Future Work

* -Implement the GPU-based version using OpenCL.

# References

* Mnasri, W., et al. (2021). Parallel social behavior-based algorithm for identification of influential users in social networks. SN Applied Sciences.

[*Paper Link*](https://drive.google.com/file/d/1vp5he-8ogdPJRFPiT6BDGPVZoH9CEnAh/view)