Phase 2 Report:
Parallel Algorithm Implementation and Demonstration

PDC – Project | CS - B

i220821 Muhammad Tahir

i228223 Sameed Ahmed Siddiqui

I221239 Tauha Imran

Contents

1. Introduction

Overview of PSAIIM

The Parallel Socially-Aware Influence Identification Model (PSAIIM) is designed to detect influential nodes in large-scale social networks such as Twitter. It builds upon centrality and connectivity measures and models social behavior through influence spread using graph structures like retweet, reply, and mention interactions. PSAIIM integrates PageRank and community detection methods in a distributed manner.

Motivation for Parallelization

- **Scalability**:
  Modern social network datasets, such as the Higgs Twitter dataset (500K+ nodes), require parallel computation due to their size and complexity.
- **Performance**:
  Parallel computing shortens execution time significantly by distributing workloads across processes and threads.
- **Toolkit Utilization**:
  MPI enables communication between distributed tasks, OpenMP leverages shared memory for intra-node speedup, and METIS aids in partitioning graphs for optimal workload distribution.

---

2. Implementation Details

2.1 Parallelization Strategy

- **MPI**:
  Each MPI process handles a separate partition of the input graph. Communication occurs during synchronization of influence values.
- **OpenMP**:
  Inside each process, OpenMP threads are used to accelerate community detection and influence spread computations.
- **Hybrid Model**:
  The combination of MPI (inter-node) and OpenMP (intra-node) improves performance by optimizing processor-level and thread-level parallelism.

2.2 Graph Partitioning with METIS

- **Partitioning Type**:
  We used k-way edge-cut partitioning, which minimizes the number of edges crossing partitions, reducing inter-process communication.

- **Integration**:
  METIS was directly integrated in the graph loader module, where the edge list is parsed, node indices reindexed, and METIS output mapped back to the input graph.

2.3 Dataset Handling

- **Dataset**: Higgs Twitter network (SNAP repository), containing 500K nodes and over 14 million edges.
- **Preprocessing**:
  using files *generate_New_dataset.py* & *generate_Reduceddataset_from_Higgs.py*
  - Removed isolated nodes and duplicate edges.
  - Reindexed for 0-based METIS compatibility.
  - Stored graphs in METIS and edge-list formats.

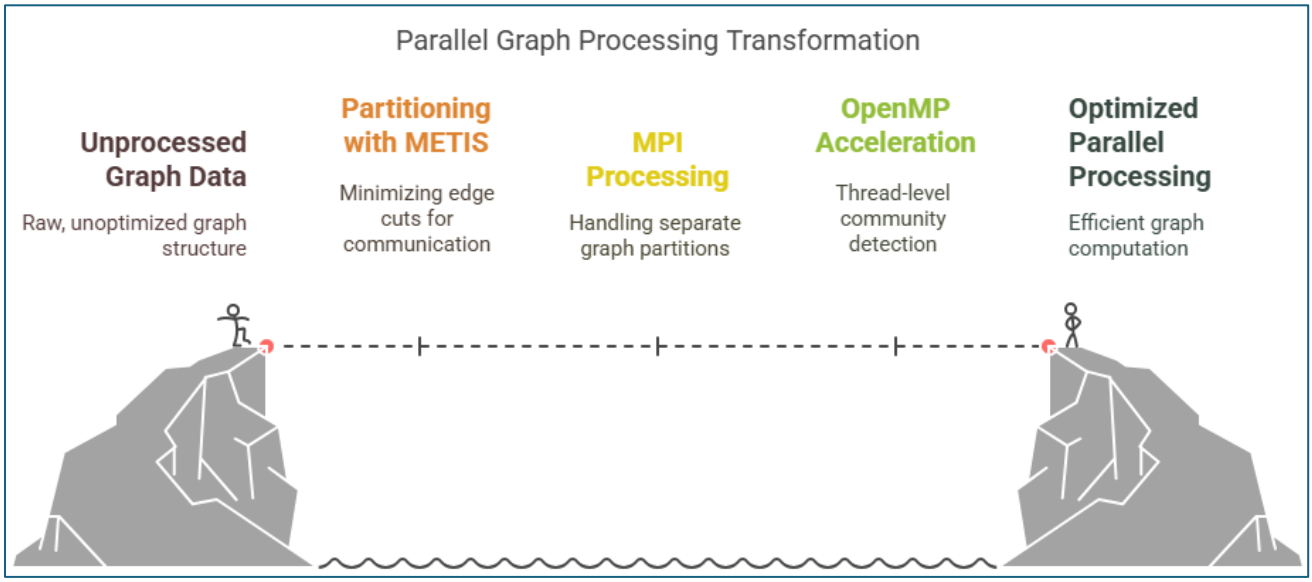

figure: illustration of Parallel Graph Processing Transformation

2.4 Technical Workflow Steps (1–7)

Step	Description
1. Load Data • social, retweet, reply, mention networks • interests per user	Loads Higgs Twitter dataset including:

2. Initialize Graph Adds directed, weighted edges and assigns interest vectors.	Initializes up to 500,000 nodes.
3. Detect Communities • Strongly Connected Components (SCCs) • Single-node CACs (Connected Acyclic Components)	Uses **DFS** to find:
4. Calculate Influence Power Weights: • α_retweet = 0.50 • α_comment = 0.35 • α_mention = 0.15 • Damping factor = 0.85	Computes influence using **parallel Personalized PageRank (PPR)**.
5. Select Seed Candidates Compares influence power to neighborhood influence.	Based on influence zone I(L) and threshold (0.015).
6. Select Seeds Selects top-k seeds maximizing spread.	Builds **Influence-BFS trees** for candidates.
7. Verify & Log Logs results to graph_analysis.log.	Validates community assignment, connectivity, and power.

figure: illustration of technical workflow steps

2.5 Complexities

- **Time Complexity**:

$$O\left(\frac{k \cdot m}{p} + n\right)$$

where:

- k = PPR iterations *(Personalized PageRank)*
- m = number of edges
- n = number of nodes
- p = threads

- **Space Complexity**:
  $O(\max(n, m))$

- **Parallelization**:
  Uses **OpenMP** for intra-partition threading (fine-grained parallelism).

---

3. Experimental Setup

3.1 Hardware/Environment

- **Cluster**:
  2-3 laptops connected over a local network running WSL2 Ubuntu 22.04. or 24.04
- **MPI Version**:
  MPICH 4.1.1, OpenMP GCC 11.4.0.
- **Compilation**:
  mpic++ -fopenmp -O3 main.cpp -o psaiim
- **Dependencies:**
  METIS 5.1.0, GTK for GUI (optional), matplotlib and PyQt for Python GUI version.

3.2 Execution Parameters

- **Processes/Threads**:
  - MPI: 1 to 3 processes depending on test.
  - OpenMP: 4 threads per process.
- **OpenMP Config**: Static scheduling, nested parallelism disabled.
- **METIS Config**: 3-way k-cut, 500 iterations max.

---

4. Results and Analysis

4.1 Performance Metrics

Runtimes:

Configuration	Dataset Size	Total Time (ms)
Serial	2000 nodes	4861

OpenMP	2000 nodes	3102
MPI + OpenMP	2000 nodes	2933

Table: showing execution time stats

Speedup:

Calculated using

Speedup= Parallel Execution Time / Serial Execution Time

Method	**Speedup (×)**
Serial	1.00 (baseline)
OpenMP	1.57×
MPI + OpenMP	1.66×

Table: showing speed up stats

Now for all cases

NUM_NODES	**Serial (ms)**	**OpenMP (ms)**	**MPI (ms)**
2000	4861	2933	3102
4000	10826	6520	6895
6000	20083	12096	12791
8000	33695	20298	21457

4.2Visualizations

- **Comparing Total Runtimes**
  The bar chart shows the relationship of the execution times between Serial (ms), OpenMP (ms), MPI+OpenMP (ms) implementations. Clearly showing a faster runtime for parallel implementations

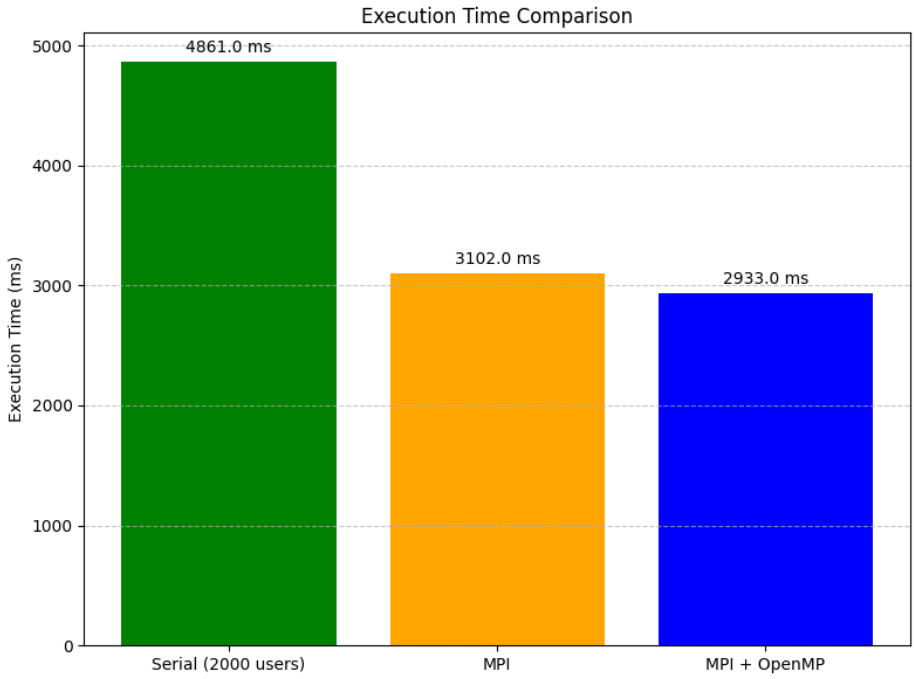

figure: bar chart of runtime comparisons

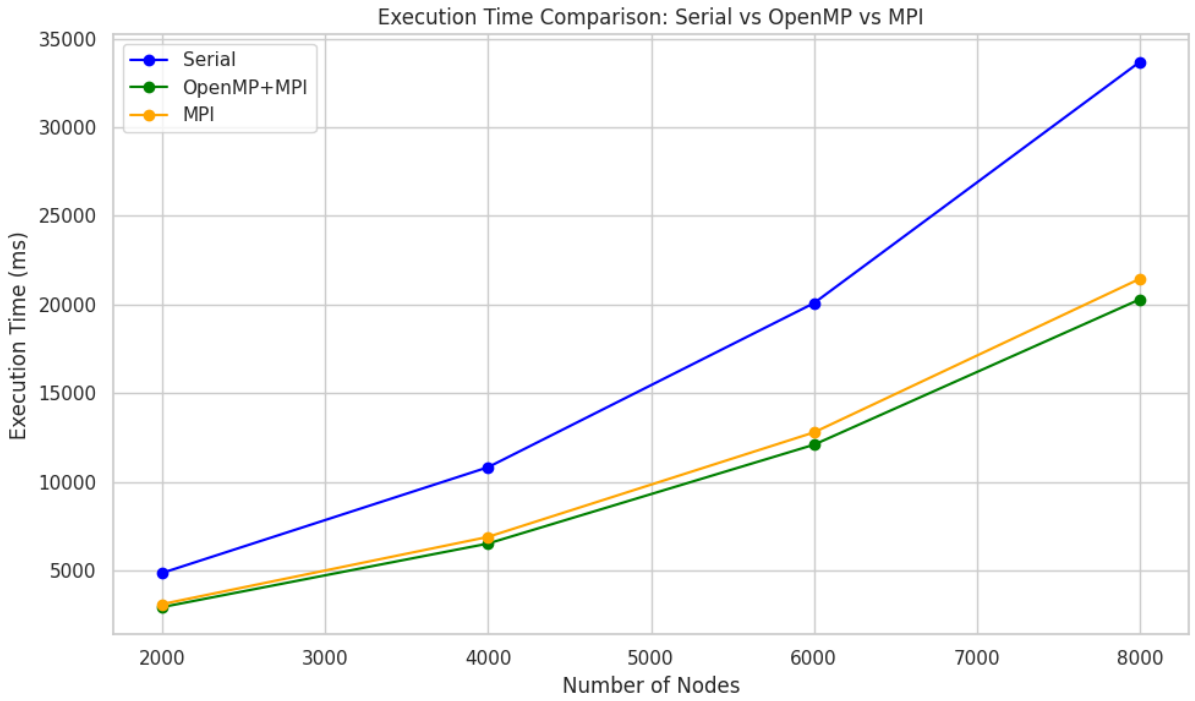

figure: line chart of runtime comparisons

- **Serial Scalability**:
  The figure below shows the rise in time taken for program execution for the scalar implementation. The time taken (y-axis) grows proportionally to the number of nodes processed (x-axis)

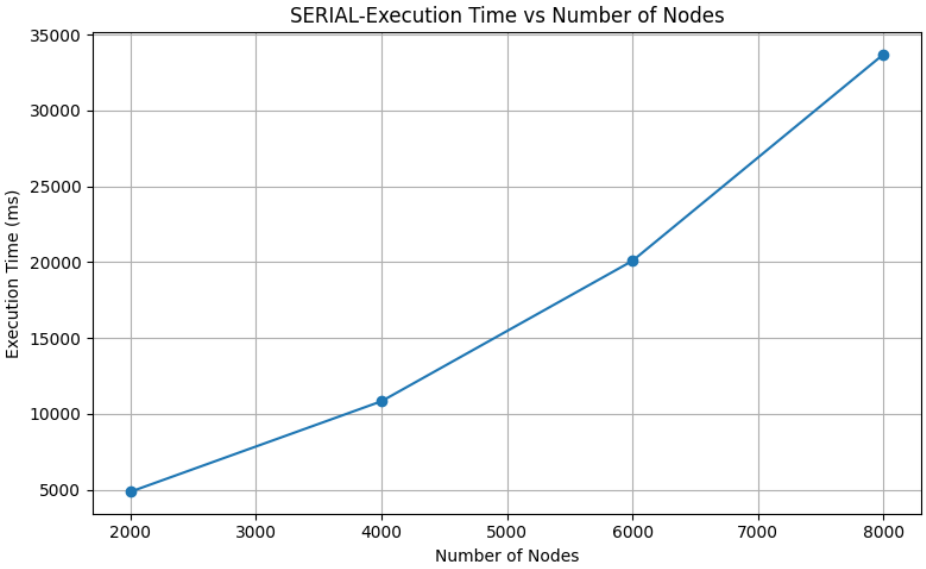

figure: line chart showing the relationship b/w time taken and number of nodes for the serial implementation

- **Basic MPI implementation analytics:**
  With a speedup of 1.57× , the MPI based parallelization implementation , showed improvement. Initialization time became an added factor as shown in bar chart below. The chart below shows the initialization and execution time for a data set of 2000 nodes.

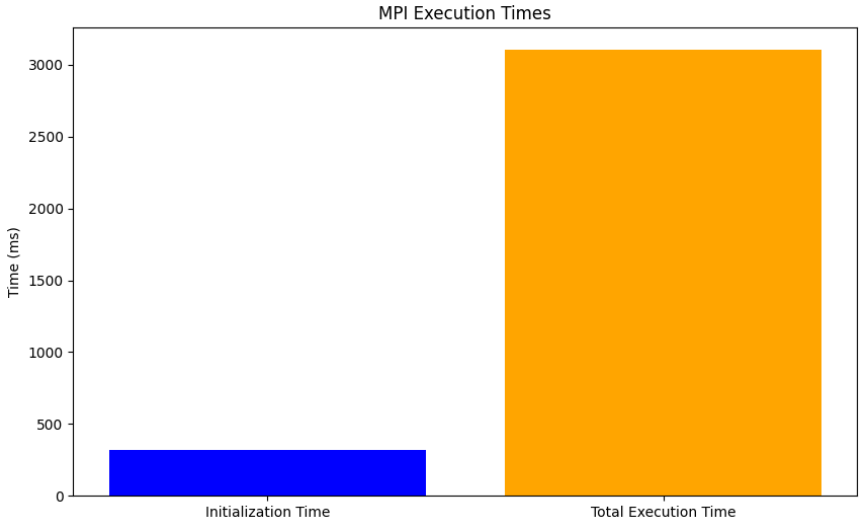

figure: bar chart for initialization and execution time mpi

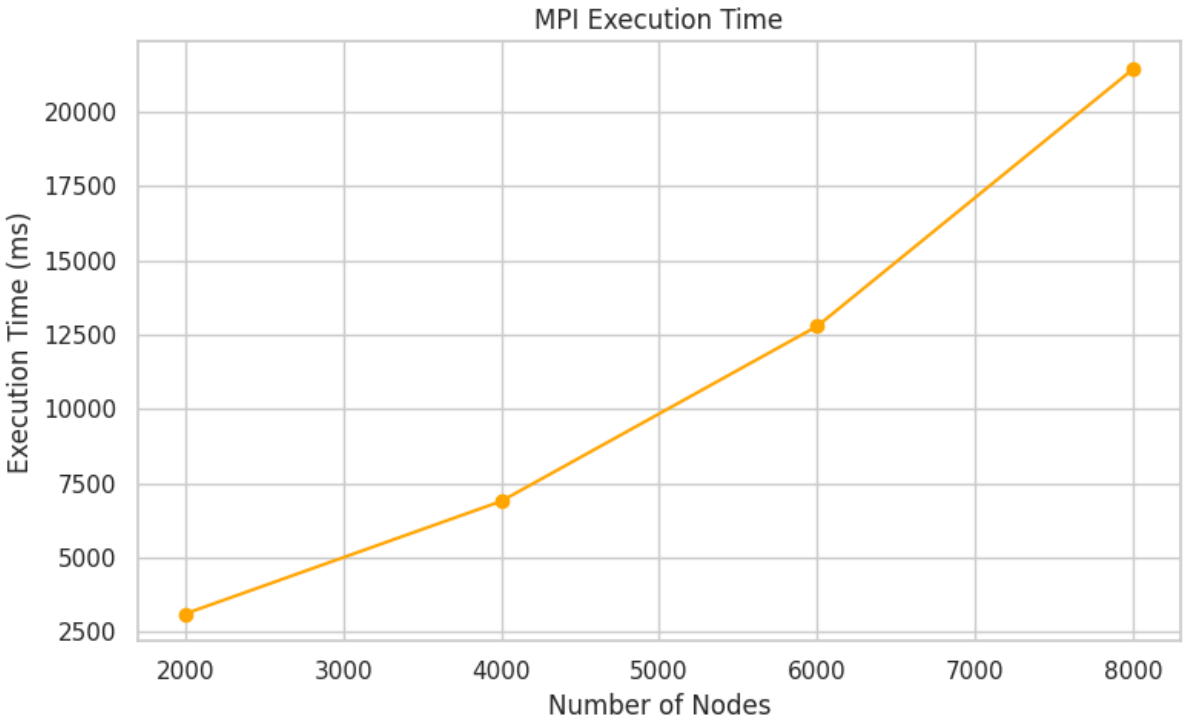

figure: line chart for initialization and execution time mpi – approach

- **MPI + OpenMP implementation:**

  With a speedup of 1.66× , the MPI + OpenMP based parallelization implementation , showed slight improvement from only the mpi version. The chart below shows the initialization and execution time for a data set of 2000 nodes.

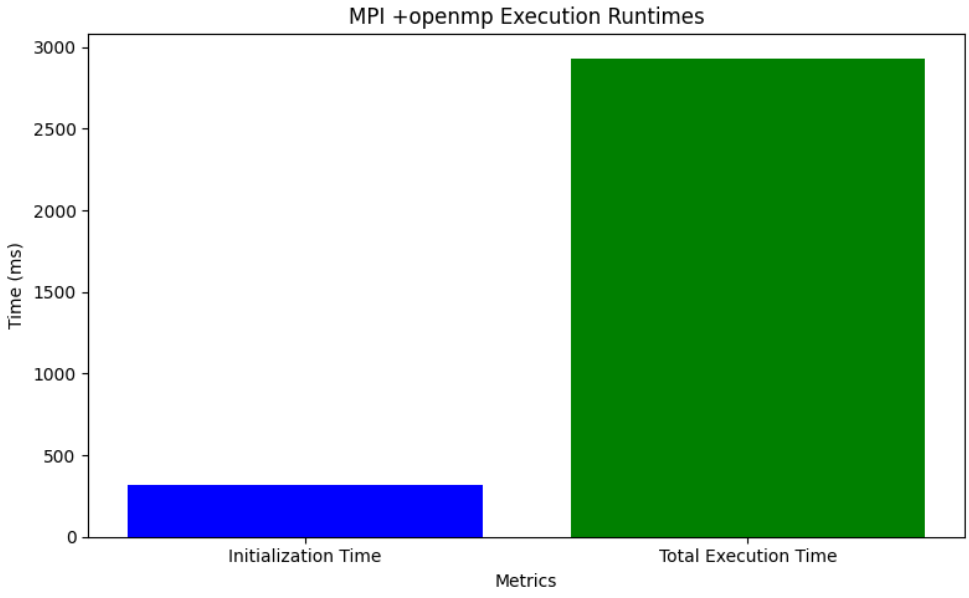

figure: bar chart for initialization and execution time mpi + OpenMP – approach

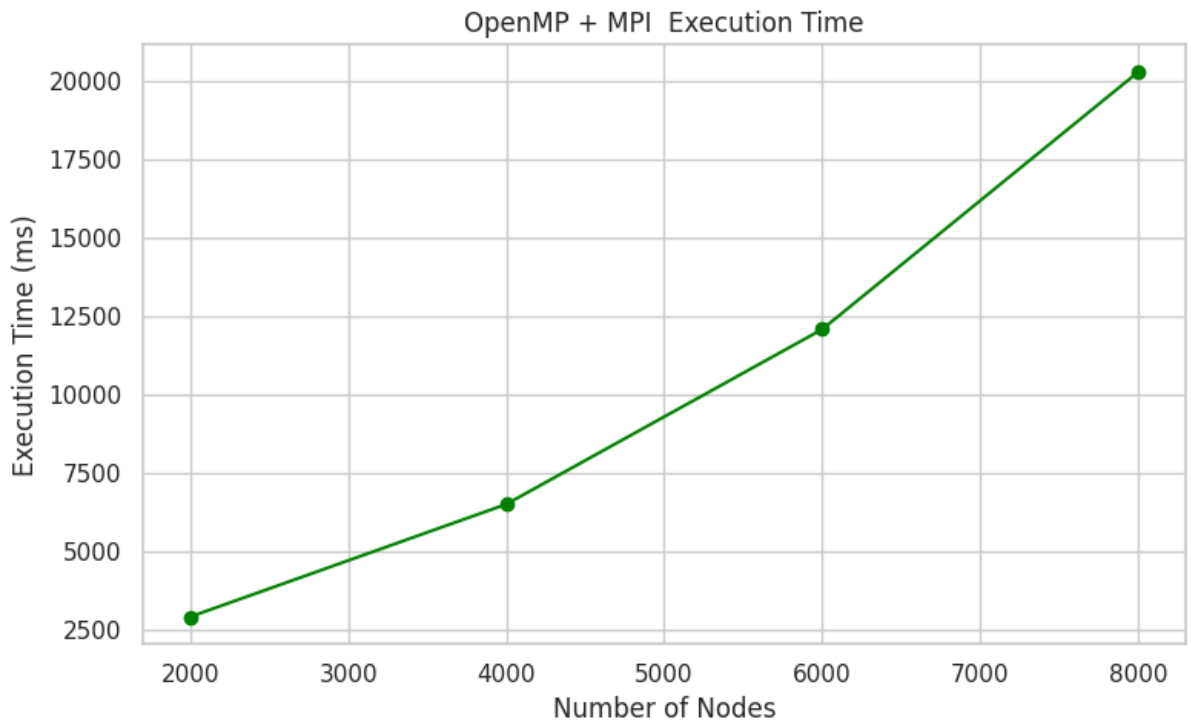

figure: line chart for initialization and execution time mpi + OpenMP – approach

4.3 Scalability Discussion

- **Strong Scaling**:
  - Fixed problem size, increasing MPI processes or nodes improves performance until communication overhead dominates.
- **Weak Scaling**:
  - Performance maintained with increasing dataset size when more MPI nodes are added.

---

5. Discussion

- **Challenges**:
  - Synchronizing results across processes.
  - MPI latency and bandwidth bottlenecks.
  - Preprocessing and cleaning data

- Literature was very intricate and advanced compared to our usual course of studies in university.

- **METIS Benefits**:
  - Reduced edge cuts and improved load balancing.
- **Hybrid Model Trade-off**:
  - OpenMP adds local parallelism but increases memory usage.
- **Accuracy vs Performance**:

  Results consistent across all versions, hybrid model provides optimal balance.
  At the end we got our required results of most influential nodes.

  Here is a scatter plot showing the findings of the influence power metric (calculated in the code) and the number of followers they had.

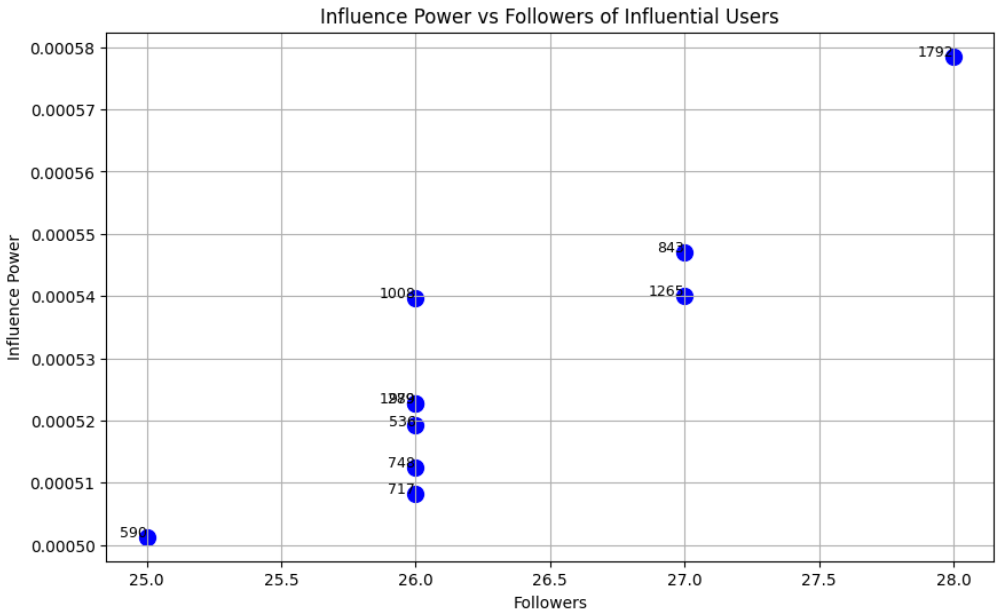

figure: scatter plot of most influential nodes (influence_power vs. num_followers)

---

6. Conclusion and Future Work

- **Summary**:
  - PSAIIM shows significant runtime improvements via hybrid parallelization.
- **Best Configuration**:

- MPI + OpenMP for distributed environments.

- **Bonus work:**
  - Implementation of multiple device cluster.
  - Static html pages for user friendly documentation.
  - Implementation of a GUI (still in development)

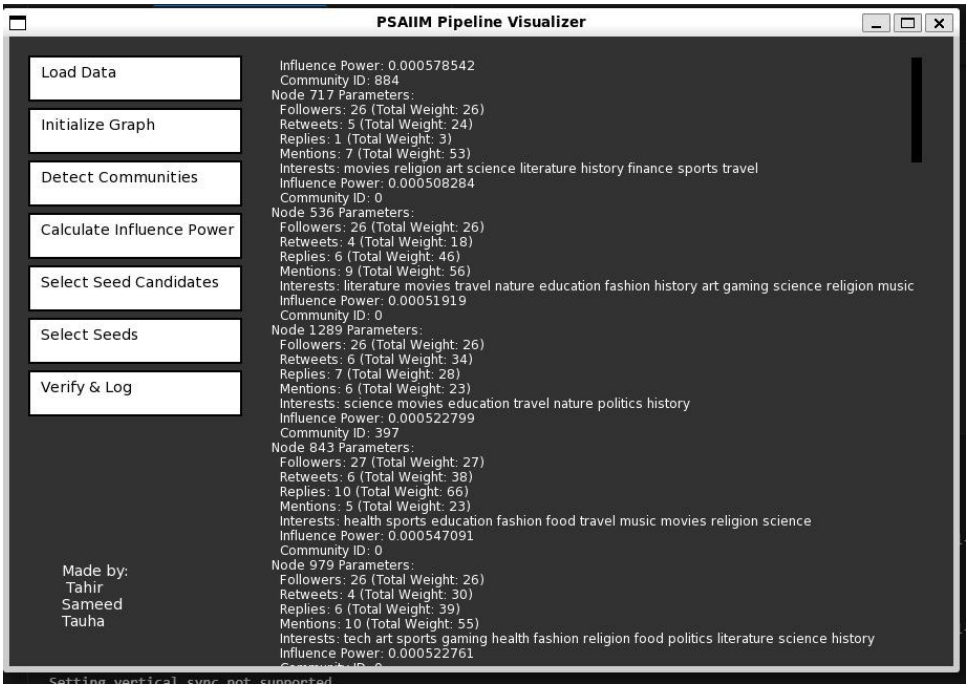

figure: screenshot of Gui under development

- **Future Plans**:
  - GPU-accelerated PageRank (OpenCL/CUDA).
  - Dynamic rebalancing during execution.
  - Integration into Apache Spark pipeline.

---

7. References

- PSAIIM: Parallel Social Behavior-Based Influence Model (Original Research Paper)
- METIS Documentation: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
- MPICH: https://www.mpich.org/
- Higgs Dataset (SNAP): https://snap.stanford.edu/data/
- OpenMP API Docs: https://www.openmp.org/

8. Appendix

- **Code Snippet: MPI Broadcast**

```
MPI_Bcast(&global_rank_vector[0],  size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

- **Run Script**

```
mpirun -np 3 ./psaiim input.graph
```

- **GitHub Repo**
  https://github.com/tauhaimran/Parallel-social-behavior-based-algorithm-for-identification-of-influential-users-in-social-network