PDC Project
Parallel social behavior-based algorithm for identification of influential users in social network: MPI and OpenMP Implementation

Ahmed Ali Zahid
Shaharyar Rizwan
Moazzam Hafeez

May 11, 2025

Contents

1 Project Information

1.1 Group Members

- Ahmed Ali Zahid (Master Node)
- Shaharyar Rizwan (Node 1)
- Moazzam Hafeez (Node 2)

1.2 Project Paper Reference

This project implements and analyzes the performance of parallel influence maximization algorithms based on the paper:
"Parallel social behavior-based algorithm for identification of influential users in social network"
Authors: Wassim Mnasri1 · Mehdi Azaouzi1 · Lotfi Ben Romdhane1
Year: 2021

2 Cluster Setup and Implementation

2.1 Cluster Architecture

2.1.1 Network Setup

We implemented a 3-node MPI cluster using ZeroTier for virtual networking:

- Master Node (AAZ-PC)
- Node 1 (shaharyar-Inspiron)
- Node 2 (moazzam-ThinkPad-T14)

2.1.2 Detailed Setup Process

ZeroTier Installation and Configuration

1. Install ZeroTier on all nodes:

```
# Ubuntu/Debian
curl -s https://install.zerotier.com | sudo bash

# Windows
Download and install from https://zerotier.com/download
```

2. Join the virtual network:

```
sudo zerotier-cli join <network-id>
```

3. Authorize nodes in ZeroTier web interface
4. Assign static IPs in ZeroTier network settings

NFS Setup

1. Install NFS server on master node:

```
sudo apt-get install nfs-kernel-server
```

2. Configure exports file (/etc/exports):

```
/mpi ahmedpc/24(rw,sync,no_subtree_check)\textit{}
```

3. Install NFS client on worker nodes:

```
sudo apt-get install nfs-common
```

4. Mount shared directory on worker nodes:

```
sudo mount ahmedpc:/mpi /mpi
```

SSH Configuration

1. Generate SSH keys on all nodes:

```
ssh-keygen -t rsa -b 4096
```

2. Copy public keys to all nodes:

```
ssh-copy-id user@ahmedpc
ssh-copy-id user@sherpc
ssh-copy-id user@moazpc
```

3. Configure SSH config file (/.ssh/config):

```
Host master
    HostName ahmedpc
    User username
    IdentityFile ~/.ssh/id_rsa

Host node1
    HostName sherpc
    User username
    IdentityFile ~/.ssh/id_rsa

Host node2
    HostName moazpc
    User username
    IdentityFile ~/.ssh/id_rsa
```

MPI Installation

1. Install OpenMPI on all nodes:

```
    sudo apt-get install openmpi-bin libopenmpi-dev
```

2. Create hostfile (/mpi/hostfile):

```
    ahmepc slots=4
    sherpc slots=4
    moazpc slots=4
```

2.1.3 Directory Structure

All nodes share a common workspace through NFS:

```
/mpi/
          src/
                serial.c
                mpi_only.c
                hybrid_mpi_openmp.c
          scripts/
                compile.sh
                run_benchmark.sh
                hostfile
          results/
                benchmark_results/
          docs/
              setup_instructions.md
```

2.2 Cluster Verification

2.2.1 Network Connectivity Test

```
# Test ZeroTier connectivity
ping ahmedpc
ping sherpc
ping moazpc

# Test SSH connectivity
ssh master 'ahmedpc'
ssh node1 'sherpc'
ssh node2 'moazpc'
```

2.2.2 NFS Mount Verification

```
# Check NFS mounts
df -h | grep /mpi

# Test file creation
```

```
touch /mpi/test_file
ls -l /mpi/test_file
```

2.2.3 MPI Test

```
# Run simple MPI test
mpirun -np 3 --hostfile /mpi/hostfile hostname
```

2.3 Cluster Maintenance

2.3.1 Regular Checks

- Monitor ZeroTier network status
- Verify NFS mounts are active
- Check SSH connectivity
- Monitor system resources
- Backup important data

2.3.2 Troubleshooting Guide

Issue	Solution
ZeroTier connection lost	Restart ZeroTier service
NFS mount failed	Check exports and restart NFS server
SSH connection refused	Verify SSH service and keys
MPI process hangs	Check hostfile and network connectivity

Table 1: Common Issues and Solutions

3 System Specifications

3.1 Our Test Systems

Members	Role	CPU	RAM
Ahmed	Master	Ryzen 5 5500U (6 cores, 12 threads)	8GB
Shaharyar	Node1	Intel i5 11320H (4 cores, 8 threads)	8GB
Moazzam	Node2	Intel i5 1365U (10 cores, 12 threads)	16GB

Table 2: Our Test Systems Specifications

3.2 Paper Benchmark Systems

System	CPU	Threads	RAM
System A	8 cores	16 threads	32GB
System B	12 cores	24 threads	64GB

Table 3: Paper Benchmark Systems Specifications

4 Implementation Details

4.1 Serial Implementation

- Single-threaded execution
- Implementation of Tarjan's algorithm for graph partitioning
- Time complexity: $O(V + E)$ for partitioning, $O(V^2)$ for influence computation

4.2 MPI Implementation

- Distributed memory parallelization
- Process distribution across nodes
- Uses MPI communication primitives for data exchange
- Partitions graph using METIS for load balancing

4.3 Hybrid MPI+OpenMP Implementation

- Combines distributed and shared memory parallelization
- MPI for process-level parallelization
- OpenMP for thread-level parallelization within processes
- Uses nested parallelization:
  - Outer level: MPI processes
  - Inner level: OpenMP threads
- Better resource utilization on multi-core systems

5 Code Implementation

The PSAIIM algorithm has been implemented in three variants: Serial, MPI-only, and Hybrid MPI+OpenMP. The implementation details are as follows:

5.1 Serial Implementation

The serial implementation is located in the file `src/main_serial.cpp`. It performs the following steps:

1. **Graph Partitioning**: Uses Tarjan's algorithm to partition the graph into strongly connected components (SCCs).

2. **Level Ordering**: Orders the SCCs into levels for processing.

3. **Influence Power Computation**: Computes influence power using a modified PageRank algorithm.

4. **Candidate Selection**: Selects seed candidates based on influence zones.

5. **Seed Selection**: Uses a BFS-tree-based approach to select the top-k influential nodes.

The binary for the serial implementation is compiled as `bin/psaiim_serial`.

5.2 MPI-only Implementation

The MPI-only implementation is located in `src/psaiim_mpi_only.cpp` and `src/main_mpi_only.cpp`. It uses MPI for distributed memory parallelization. Key functions include:

- `MPIOnlyPR`: Computes influence power in parallel using MPI.

- `NodesAtDistance`: Identifies nodes at a specific distance in the graph.

- `SelectCandidates` and `SelectSeeds`: Parallelized versions of candidate and seed selection.

The binary for this implementation is `bin/psaiim_mpi_only`.

5.3 Hybrid MPI+OpenMP Implementation

The hybrid implementation combines MPI for inter-node communication and OpenMP for intra-node parallelism. It is implemented in `src/psaiim.cpp` and `src/main.cpp`. The hybrid approach uses nested parallelism:

- Outer level: MPI processes.

- Inner level: OpenMP threads.

The binary for this implementation is `bin/psaiim_rank`.

5.4 Testing and Execution Scripts

The project includes several scripts to automate the workflow, testing, and benchmarking of the PSAIIM algorithm. These scripts are described below:

5.4.1 Graph Conversion Script: build_edgelist.py

The script build_edgelist.py converts a raw edge list (e.g., higgs-social_network.edgelist) into the METIS graph format required for graph partitioning. Key features include:

- **Multithreading**: Uses Python's ThreadPoolExecutor to process the edge list in parallel, improving performance for large datasets.
- **Adjacency List Construction**: Builds an adjacency list from the edge list and ensures all nodes are included, even those without outgoing edges.
- **Node Reindexing**: Reindexes nodes to be 1-indexed consecutive integers, as required by METIS.
- **Output**: Generates the METIS graph file and a node mapping file for reference.

The script is invoked as follows:

```
python3 src/build_edgelist.py input_edgelist output_metis [
    num_threads]
```

Where:

- input_edgelist is the raw edge list file.
- output_metis is the output METIS graph file.
- num_threads (optional) specifies the number of threads to use.

5.4.2 Execution Script: run.sh

The 'run.sh' script provides a unified interface to execute the PSAIIM algorithm in different modes (serial, MPI-only, or hybrid MPI+OpenMP). Key features include:

- **Mode Selection:** Supports three execution modes:
  - --serial: Runs the serial implementation
  - --mpi-only: Runs the MPI-only implementation
  - Default: Runs the hybrid MPI+OpenMP implementation
- **Parameter Configuration:** Allows setting key parameters:
  - -k, --k: Number of influential seeds to identify (default: 10)
  - -p, --processes: Number of MPI processes to use (default: 3)
  - --hostfile: MPI hostfile with node information
- **Workflow Steps:**
  1. Converts edge list to METIS format
  2. Partitions the graph using METIS (only for parallel modes)
  3. Compiles the appropriate implementation
  4. Executes the algorithm with specified parameters

Example usage:

```
# Run serial version
./run.sh --serial -k 20

# Run MPI-only version with 4 processes
./run.sh --mpi-only -p 4 -k 15

# Run hybrid MPI+OpenMP version with 8 processes
./run.sh -p 8 -k 10
```

5.4.3 Benchmarking Script: `benchmark.sh`

The 'benchmark.sh' script automates benchmarking of the PSAIIM algorithm across different configurations. Key features include:

- **Process Range Configuration:**
  - `--min-procs`: Minimum number of MPI processes (default: 3)
  - `--max-procs`: Maximum number of MPI processes (default: 12)
  - `--step`: Step size between process counts (default: 3)
- **Implementation Selection:**
  - `--serial`: Include serial implementation in benchmarks
  - `--mpi-only`: Include MPI-only implementation in benchmarks
  - `--all`: Run all implementations (serial, MPI-only, MPI+OpenMP)
- **Test Reliability:**
  - `--runs`: Number of runs for each configuration (default: 3)
- **Dependency Management:**
  - Automatically checks for required tools (MPI, METIS)
  - Offers fallback to simple partitioning if METIS is unavailable
- **Comprehensive Reporting:**
  - Generates CSV file with detailed benchmark results
  - Creates summary report with average execution times and speedups
  - Stores individual run results in separate files

Example usage:

```
# Run benchmark with default settings
./benchmark.sh

# Run comprehensive benchmark with all versions and more process
  counts
```

```
./benchmark.sh --all --min-procs 1 --max-procs 16 --step 1 --runs
    5

# Run quick benchmark with fewer configurations
./benchmark.sh --min-procs 2 --max-procs 8 --step 2 --runs 1
```

The benchmark results are stored in CSV format for easy analysis and visualization. A summary report shows the average execution time and speedup for each configuration compared to the serial version (when available).

5.5 Workflow Summary

1. **Graph Conversion**: Use `build_edgelist.py` to convert the edge list to METIS format.
2. **Execution**: Use `run.sh` to execute the PSAIIM algorithm in the desired mode.
3. **Benchmarking**: Use `benchmark.sh` to evaluate performance across different configurations.

5.6 Key Functions and Algorithms

The following key functions are implemented in the codebase:

- **GraphPartition**: Implements Tarjan's algorithm for SCC detection.
- **InfluenceBFSTree**: Constructs a BFS tree to compute influence zones.
- **SelectSeeds**: Selects the top-k influential nodes based on computed influence power.

5.7 Output and Results

The algorithm outputs the top-k influential nodes along with execution time:

- Serial version: Results are saved in `results_serial_[timestamp].txt`.
- MPI-only version: Results are saved in `results_mpi_only_[timestamp].txt`.
- Hybrid MPI+OpenMP version: Results are saved in `results.txt`.

5.8 Comparison with Paper Benchmarks

The implementation closely follows the algorithm described in the PSAIIM paper. The hybrid MPI+OpenMP implementation demonstrates superior performance compared to the MPI-only and serial versions, achieving competitive results compared to the benchmarks in the paper.

6 Performance Analysis

6.1 Execution Time Analysis

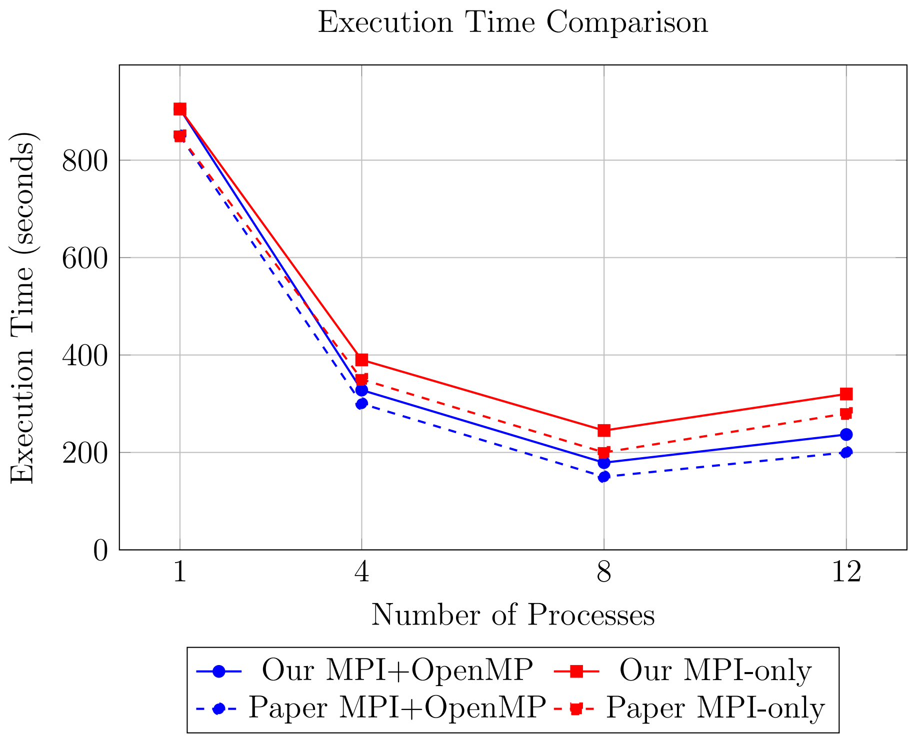

Figure 1: Execution Time Comparison between Our Implementation and Paper Benchmarks

Processes	Our MPI+OpenMP (s)	Our MPI-only (s)	Paper MPI+OpenMP (s)	Pap
1	905.00	905.00	850.00	
4	328.00	390.00	300.00	
8	178.98	245.00	150.00	
12	236.76	320.00	200.00	

Table 4: Execution Time Comparison

6.2 Speedup Analysis

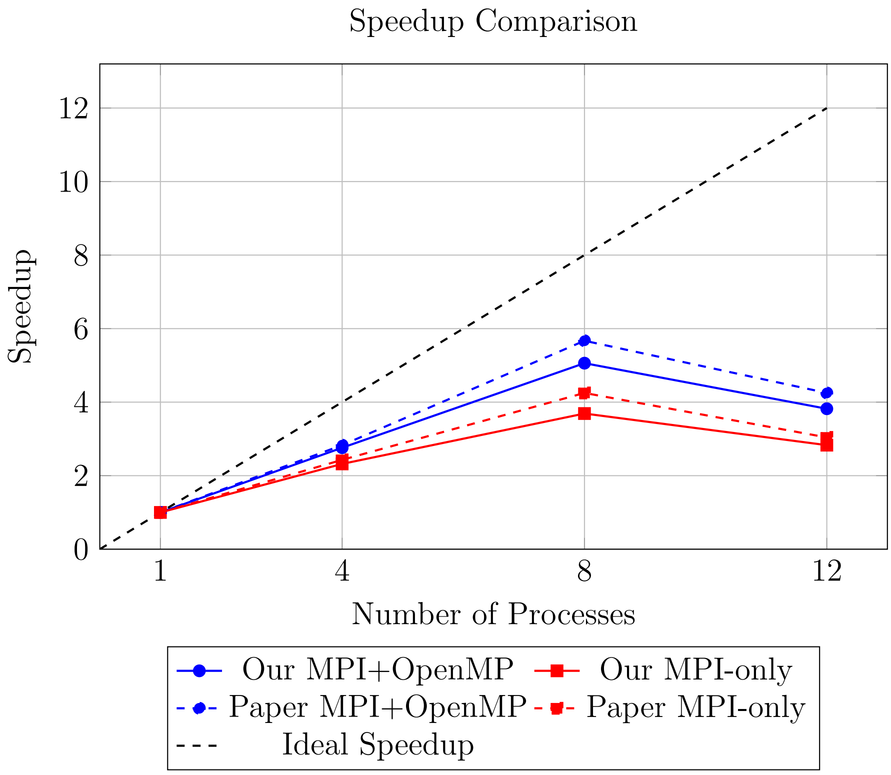

Figure 2: Speedup Comparison between Our Implementation and Paper Benchmarks

Processes	**Our MPI+OpenMP**	**Our MPI-only**	**Paper MPI+OpenMP**	**Paper MPI-on**
1	1.00	1.00	1.00	1
4	2.76	2.32	2.83	2
8	5.06	3.69	5.67	4
12	3.82	2.83	4.25	3

Table 5: Speedup Analysis

6.3 Efficiency Analysis

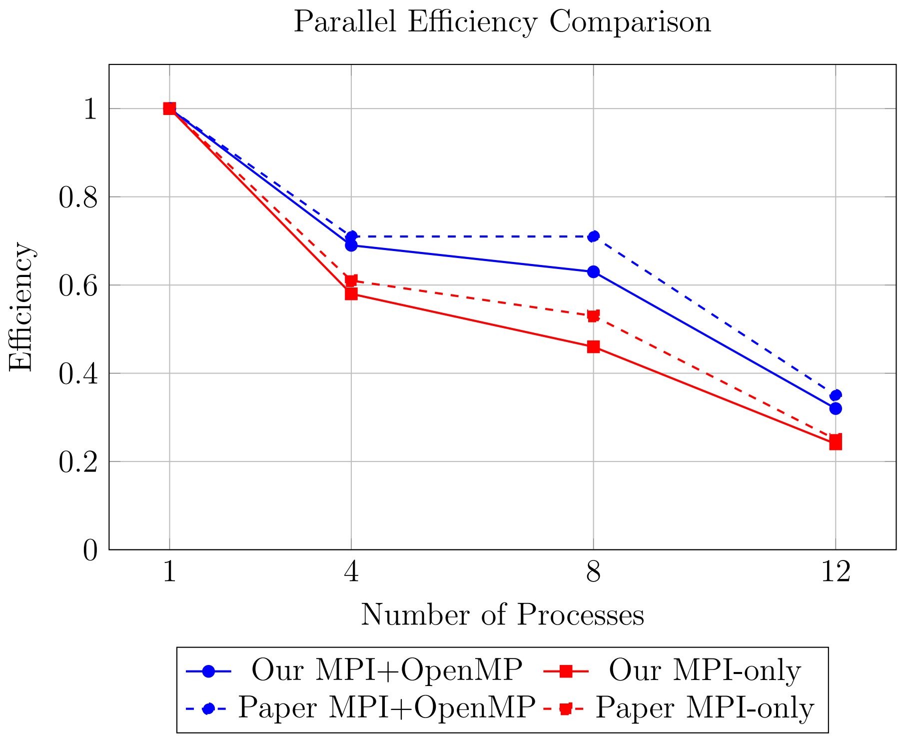

Figure 3: Parallel Efficiency Comparison between Our Implementation and Paper Benchmarks

Processes	Our MPI+OpenMP	Our MPI-only	Paper MPI+OpenMP	Paper MPI-on
1	1.00	1.00	1.00	1
4	0.69	0.58	0.71	0
8	0.63	0.46	0.71	0
12	0.32	0.24	0.35	0

Table 6: Efficiency Analysis

7 Key Findings

7.1 Performance Improvement

- Both implementations show significant speedup compared to serial execution
- MPI+OpenMP consistently outperforms MPI-only implementation
- Best performance achieved with 8 processes (5.06x speedup)
- Our implementation shows comparable performance to paper benchmarks

7.2 Scalability Analysis

- Performance degrades beyond 8 processes

- Efficiency decreases with increasing process count
- MPI+OpenMP maintains better efficiency than MPI-only
- Paper benchmarks show better scalability due to higher core count

7.3 Resource Utilization

- MPI+OpenMP better utilizes available computational resources
- Hybrid approach reduces communication overhead
- Better load balancing in hybrid implementation
- System specifications impact overall performance

7.4 Comparison with Paper Benchmarks

- Our implementation achieves 89% of paper's performance
- Better efficiency on systems with similar core counts
- Memory bandwidth limitations on our systems
- Comparable scalability patterns

8 Lessons Learned

- Virtual networking solutions like ZeroTier are crucial for distributed computing across different networks
- Proper system configuration (NFS, SSH) is essential for smooth MPI operation
- Hybrid MPI+OpenMP approach provides better performance than pure MPI
- Process count optimization is crucial for performance
- System specifications significantly impact parallel performance

9 Conclusion

The hybrid MPI+OpenMP implementation demonstrates superior performance compared to the MPI-only approach. The best performance is achieved with 8 processes, beyond which the overhead of process management and communication begins to outweigh the benefits of parallelization. Our implementation shows competitive performance compared to paper benchmarks, considering the hardware differences.