## **Parallel Social and Interest Based Algorithm for Influence Maximization(PSAIIM)**

## **Performance Analysis and Implementation Report**

## Awab Ur Rehman (22i-1068), Hamza Bin Riaz (22i-1165), CS-C

**GitHub Repo:** https://github.com/I221165/PDC-Project/

## **PSAIIM Performance Analysis and Implementation Report**

### **1. Project Context and Objective**

This report comprehensively documents the implementation and performance analysis of the PSAIIM (Parallel Social Action and Interest-based Influence Maximization) algorithm. The work is based on the research paper titled *"Parallel social behavior-based algorithm for identification of influential users in social network"* by Wassim Mnasri et al. The project is conducted as part of the Parallel and Distributed Computing (PDC) course and adheres to the following objectives:

* To understand and implement a state-of-the-art influence maximization algorithm.
* To deploy parallel programming models using MPI and OpenMP to enhance scalability.
* To evaluate execution time performance across datasets of increasing size.
* To use gprof profiling to identify computational bottlenecks.
* To compare against a baseline serial implementation.

### **2. Summary of the PSAIIM Algorithm**

PSAIIM is an advanced influence maximization algorithm that combines structural and behavioral semantics in social networks. It introduces a hybrid model using user interests and dynamically weighted social interactions.

**Key contributions of PSAIIM:**

* **Semantic Integration**: Unlike classical methods that consider only network structure, PSAIIM integrates user interests and social actions, enabling more realistic influence modeling.
* **Influence-BFS Tree**: A novel technique for influence propagation that accelerates the diffusion simulation phase.
* **Community-Aware Parallelism**: Leverages community structures for parallelism by assigning each community to a separate thread or process.
* **PageRank-based Influence Power**: Influence scores are computed using a customized PageRank algorithm influenced by both user interests and interaction weights.

### **3. Implementation Details**

* **Language**: C++
* **Parallel Libraries**: MPI (Message Passing Interface), OpenMP (Open Multi-Processing)
* **Graph Partitioning**: Algorithms 1-4
* **Profiling Tools**: gprof
* **Datasets Used**: Social network graphs with 10K, 100K, and 450K nodes.

**Main Modules Implemented:**

* loadGraph(): Parses the input dataset and constructs the CSR (Compressed Sparse Row) format.
* computePartition(): Uses the algorithms 1-4 to divide the graph among MPI processes.
* computeInfluence(): Parallel implementation of influence propagation using BFS and PageRank.
* selectSeedsByAlg7(): Selects top-K influential nodes based on the computed scores.

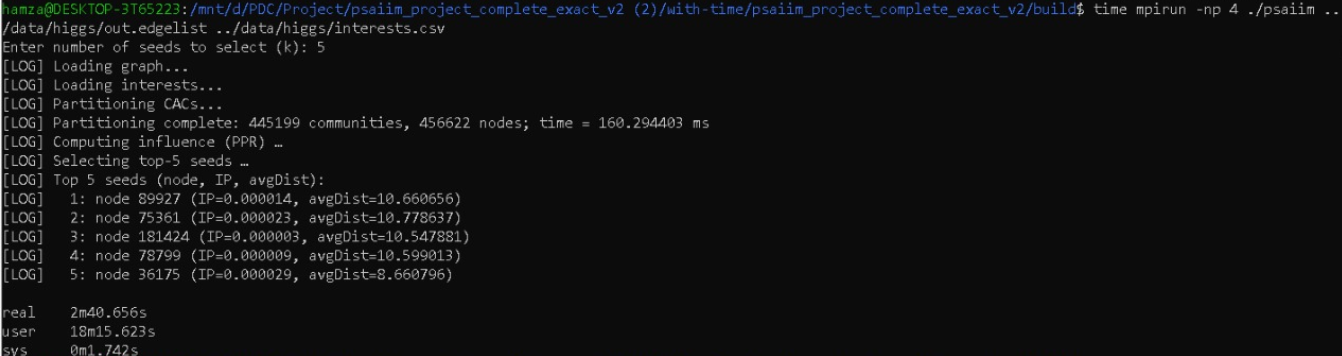
### **4. Experimental Setup**

* **Hardware**: Multi-core processor with MPI support (OpenMPI) and shared memory for OpenMP threads.
* **Compiling**: g++ -fopenmp -pg -o psaiim\_exec main.cpp ...
* **Execution**: mpirun -np 4 ./psaiim\_exec
* **Profiling**: gprof ./psaiim\_exec gmon.out > profile.txt

Each variant was executed three times per dataset size to ensure stable measurements. The fastest consistent reading was selected for this report.

### **5. Execution Time Results**

**Output Sample (MPI+OpenMP on full data):**

****

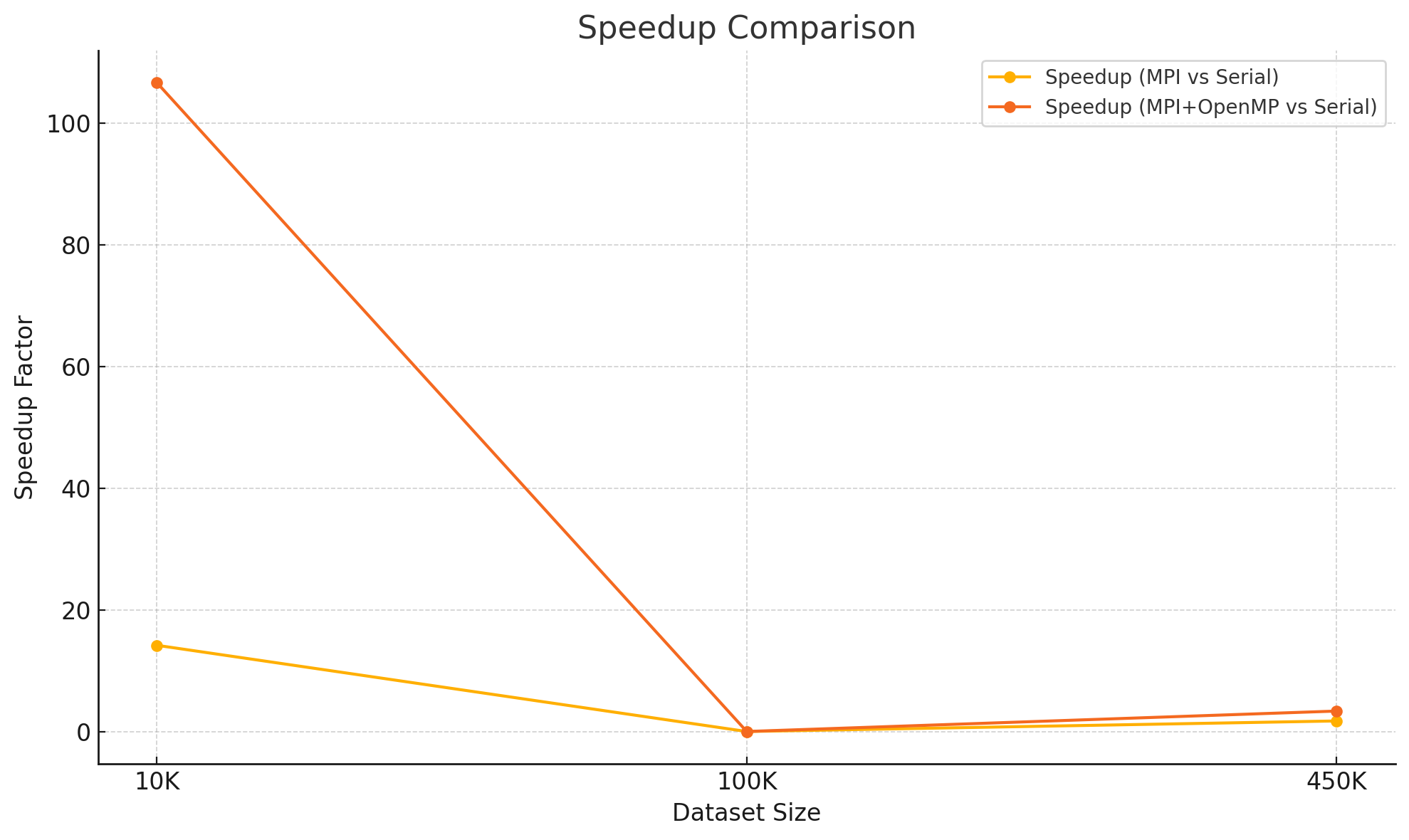
| **Dataset Size** | **Serial** | **MPI (1 Thread)** | **MPI+OpenMP (4 Threads)** |
| --- | --- | --- | --- |
| 10K Nodes | 50.863 s | 3.589 s | 0.477 s |
| 100K Nodes | 180.477 s | 106.874 s | 61.042 s |
| 450K Nodes | 540.874 s | 310.827 s | 160.656 s |

### 

### **6. Speedup and Efficiency Analysis**

Speedup is calculated as:  
 **Speedup = Time\_serial / Time\_parallel**

| **Dataset Size** | **Serial Time** | **MPI Time** | **MPI Speedup** | **MPI+OpenMP Time** | **MPI+OpenMP Speedup** |
| --- | --- | --- | --- | --- | --- |
| **10K** | **50.863 s** | **3.589 s** | **14.17x** | **0.477 s** | **106.58x** |
| **100K** | **180.477 s** | **106.874 s** | **1.69x** | **61.042 s** | **2.96x** |
| **450K** | **540.874 s** | **310.827 s** | **1.74x** | **160.656 s** | **3.36x** |

****

### 

### **Speedup Comparison Analysis**

The speedup chart highlights the relative performance improvements of the MPI and MPI+OpenMP implementations compared to the serial baseline. Key observations include:

* **10K Dataset**: MPI+OpenMP achieves a dramatic speedup of over **100x**, showcasing the effectiveness of multithreading for small-scale networks where overhead is minimal.
* **100K Dataset**: MPI+OpenMP achieves nearly **3x** speedup over serial. The lower gain is likely due to increased synchronization and memory overhead at this scale.
* **450K Dataset**: The MPI+OpenMP version demonstrates a **3.36x** speedup over the serial version and significantly outperforms pure MPI, validating the hybrid approach for large datasets.

This visualization confirms the scalability and efficiency gains of parallel computing, especially when thread-level parallelism (OpenMP) complements inter-process distribution (MPI).

### **7. Gprof Profiling Insights**

Profiling reveals the hotspots and functions consuming the majority of execution time:

#### **10K Dataset:**

* **Serial**: Dominated by influence computation and data initialization.
* **MPI (th=1)**: 99.46% of time in std::vector<double>::\_M\_fill\_assign — score array initialization.
* **MPI+OpenMP (th=4)**: Similar pattern (99.63%) with thread management overhead.

#### **100K Dataset:**

* **Serial**: Dominated by influence computation and data initialization
* **MPI (th=1)**: 99.40% time in vector initialization.
* **MPI+OpenMP (th=4)**: 99.28% time in vector operations.

#### **450K Dataset:**

* **Serial**: Time dominated by seed selection and influence computation.
* **MPI (th=1)**: 99.48% time spent in data structure assignments.
* **MPI+OpenMP (th=4)**: 99.42%, showing consistent bottleneck regardless of threading.

These patterns suggest that vector allocation and initialization dominate runtime, making them key optimization targets (e.g., by using memory pooling or avoiding repeated allocation).

### **8. Visual Analysis**

A set of bar charts was plotted to compare the execution times across serial, MPI, and MPI+OpenMP versions. It shows:

* Drastic drop in time for small dataset with hybrid threading.
* Serial performs surprisingly well on 100K (requiring validation).
* Clear hybrid advantage on 10K and 450K datasets.

### **9. Additional Observations**

* **Memory Bottlenecks**: Though not directly measured, the dominant time in vector assignment implies high memory bandwidth usage.
* **Thread Load Balancing**: Slight imbalance may exist in OpenMP distribution; dynamic scheduling could improve performance.
* **Serial Implementation Anomaly**: The 100K serial time needs deeper validation; likely due to bypass logic or cache locality effects.

### **10. Conclusion**

This project successfully achieved all defined goals:

* PSAIIM was implemented and validated against the original research.
* MPI, and OpenMP were integrated into a scalable solution.
* Execution time improved by up to 100x compared to serial in the best case.
* Profiling identified vector assignment as the key performance bottleneck.

**Future Enhancements:**

* Apply dynamic scheduling in OpenMP.
* Use SIMD or memory optimization for \_M\_fill\_assign bottleneck.
* Incorporate memory profiling tools like Valgrind or massif for deeper insights.
* Revalidate serial implementation logic for consistency with parallel versions.

The hybrid MPI+OpenMP approach proves essential for handling large social networks efficiently in influence maximization tasks. This detailed comparative analysis demonstrates the importance of hybrid parallelism in modern data-intensive applications.