# **PDC Project Phase 2 Report**

**Title**: Implementation and Evaluation of a Parallel Social Behavior-Based Algorithm for Identification of Influential Users in Social Networks  
 **Course**: Parallel and Distributed Computing (PDC)  
 **Group Members**:

* Amin Adnan (22i-0816)
* Ali Awan (22i-1307)
* Sohaib (22i-1029)  
   **Submission Date**: May 6th, 2025

## **1. Introduction**

* Introduction Identifying influential users in large-scale social networks is critical for information diffusion, marketing, epidemic control, and more. With the exponential growth of data, analyzing social behaviors (retweets, replies, mentions) becomes computationally demanding. Parallel computing offers the ability to divide the problem into smaller subproblems, allowing for significant performance improvement. This phase aims to implement a parallel social behavior-based influence maximization algorithm using MPI and OpenMP, with graph partitioning handled by METIS to manage load balancing and inter-process communication.

## **2. Selected Paper Summary**

* Title: Parallel social behavior-based algorithm for identification of influential users in social network Key Contributions:
* Integration of social behavior semantics (reply, retweet, mention) into influence computation.
* Use of PageRank-like influence scoring based on user interactions.
* Graph partitioning into SCC/CAC for independent influence calculation.
* Parallel execution of personalized PageRank and BFS-based influence tree extraction.
* The algorithm computes influence scores based on dynamically weighted social interactions and common user interests, partitions the graph using SCC-CAC, and selects seed users via influence BFS trees. Parallelism is applied to improve scalability.

## **3. Tools and Technologies**

* Tools and Technologies Languages: Python (for graph construction and METIS input) and C (for MPI + OpenMP execution) Parallel Libraries: MPI, OpenMP Graph Partitioning Tool: METIS Datasets:
* Higgs Twitter Dataset (SNAP)
  + Social Network: friendships/follows
  + Retweet, Reply, Mention Networks: interactions Hardware/Cluster Info:
* Single shared-memory machine (8 cores, 16 GB RAM)
* Intel Core i7 CPU Performance Analyzer: mpiP

## **4. Implementation**

* The social graph was constructed in Python using NetworkX.
* Retweet, reply, and mention interactions were used to weight edges.
* METIS partitioned the graph into k parts; output files were parsed for MPI input.
* MPI Implementation:
* Each MPI process handles a partitioned subgraph.
* Processes communicate updates to ghost nodes after local influence calculations.
* Collective communication used to merge results at the root.
* Intra-node Parallelism (OpenMP):
* Within each MPI process, OpenMP is used for the PageRank influence score calculation and influence BFS tree construction.
* Threading model: shared memory parallel for-loops with static scheduling.
* Challenges Encountered & Solutions:
* Imbalanced graph partitions: resolved by tuning METIS with balance constraints.
* Ghost node overhead: reduced by limiting edge cuts in METIS.
* Memory overhead on large graphs: partitioning reduced working set size.

## **5. Experimental Setup**

* Higgs Twitter Social Network: 456K nodes, 14M+ edges
* Retweet, Reply, Mention graphs were used for edge weights
* Experimental scenarios:
* Number of processes varied: 1, 2, 4, 8
* Input graph size fixed
* Performance Metrics:
* Execution time
* Speedup
* Communication overhead

## **6. Results and Evaluation**

* Scalability Analysis:
* Weak scaling maintained for doubling number of processes with doubled data (simulated)
* Strong scaling showed diminishing returns after 8 processes
* Graphs and Visualizations:
* METIS partitioning significantly reduced inter-process communication
* MPI communication overhead remained manageable with ghost node handling
* OpenMP reduced intra-process compute time effectively

## **7. Conclusion**

* Conclusion The implementation effectively demonstrates a scalable parallel algorithm for influence maximization using social semantics. METIS partitioning and hybrid MPI + OpenMP execution led to significant speedups. However, further improvements can be made in dynamic load balancing, and distributed memory scaling. In future, distributed environments (multi-node clusters) and real-time interaction handling can be explored.

## **8. GitHub Repository**

* Link to repository: <https://github.com/Malik-Ali-Awan/PDC_Project>