Parallel social behavior-based algorithm for identification of influential users in social networks

Parallel and Distributed Computing

22i1008, 22i0777, 22i0784
Section: H

Parallel Social Behavior-Based Algorithm for Identification of Influential Users in a Social Network (PSAIIM)

1. Introduction

Overview of PSAIIM

The Parallel Social Behavior-Based Algorithm for Identification of Influential Users in a Social Network (PSAIIM) is a specialized graph-based method designed to identify key influencers by analyzing user interaction patterns, interests, and social behavior. It utilizes a combination of graph theory, community detection, and behavioral metrics such as interest overlap and engagement levels to determine influence scores.

Motivation for Parallelization

With the rapid growth of social networks and the volume of user interactions, running influence detection algorithms on large-scale datasets becomes computationally intensive. Serial implementations fail to meet the performance demands of real-time or near-real-time analytics. Hence, parallelization is essential for:

- **Reducing execution time**
- **Improving scalability**
- **Enhancing efficiency on large graphs**

2. Implementation Details

2.1 Parallelization Strategy

To efficiently process large-scale social network graphs and compute influence scores, we adopted a **hybrid parallelization strategy** that leverages both **distributed-memory** and **shared-memory** paradigms.

- **1. MPI for Distributed Memory Parallelism**

We employed **MPI (Message Passing Interface)** to distribute the computational workload across multiple **processes or compute nodes** in a cluster. Each MPI process is responsible for handling a distinct partition of the global social graph. This allows us to scale the algorithm to multi-node environments where each process operates on its **own address space**.

- **Purpose**: Coarse-grained parallelism across machines.
- **Role**: Distributes partitions of the graph.
- **Communication**: Inter-process messaging to synchronize influence updates between partitions.

🔹 2. METIS for Graph Partitioning and Load Balancing

To support effective MPI-level distribution, we integrated **METIS**, a state-of-the-art graph partitioning tool, to pre-process and divide the original social network graph into balanced subgraphs.

- **Purpose**: Achieve load balancing and minimize communication.
- **What It Does**:
  - Divides the graph into k partitions with approximately equal node/edge loads.
  - Minimizes edge cuts between partitions to reduce MPI communication overhead.
- **Why METIS**:
  - Produces high-quality partitions with low edge cuts.
  - Allows each MPI process to receive a computationally balanced workload.

Within each MPI process (i.e., on each compute node), we used **OpenMP (Open Multi-Processing)** to enable **fine-grained parallelism** across multiple CPU cores using **shared memory**.

3. OpenMP for Intra-Node Parallelism

- **Purpose**: Speed up PageRank-style influence computations within a graph partition.
- **Implementation**:
  - Influence scores for independent communities or nodes are calculated using OpenMP threads.
  - Each OpenMP thread processes one or more components (e.g., SCCs/CACs) in parallel.

2.2 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.edgelistfor 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 other 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.

Benefits of SCC and CAC Partitioning:

- Reduced cross-node communication.

- Better preservation of influence flows.

- Natural fit for user-centric interaction patterns in social networks.

2.3 Dataset Handling

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-activity_time.txt– temporal information of activity

Preprocessing Steps:

- Combined and cleaned edges.
- Applied SCC and CAC partitioning algorithms.
- Mapped features (mentions, replies, retweets) for enhanced influence propagation mode

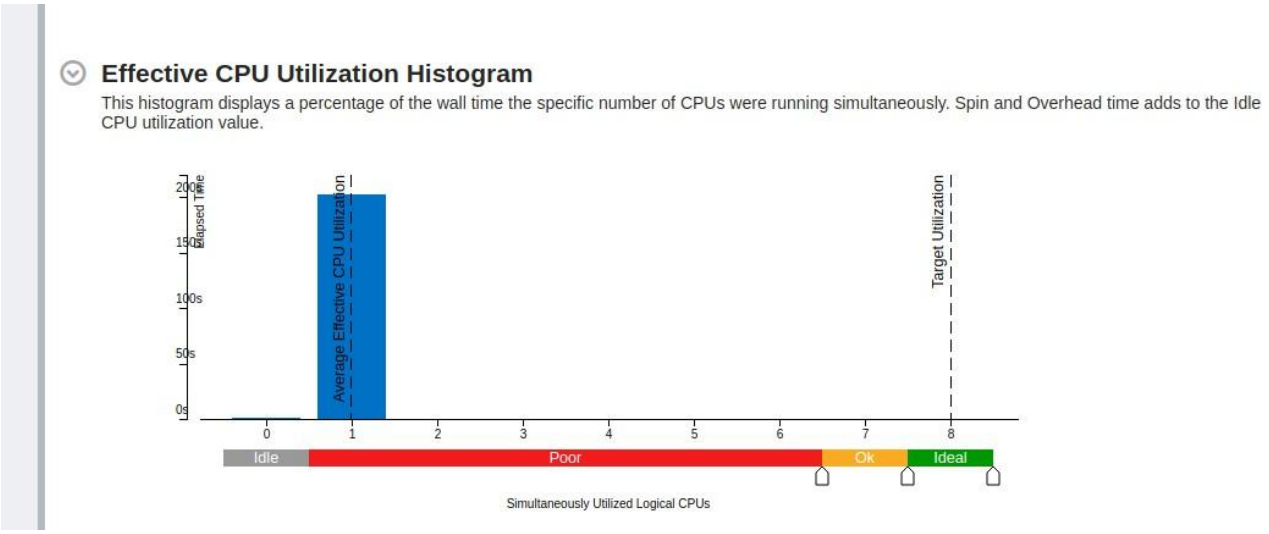
4. Results and Analysis

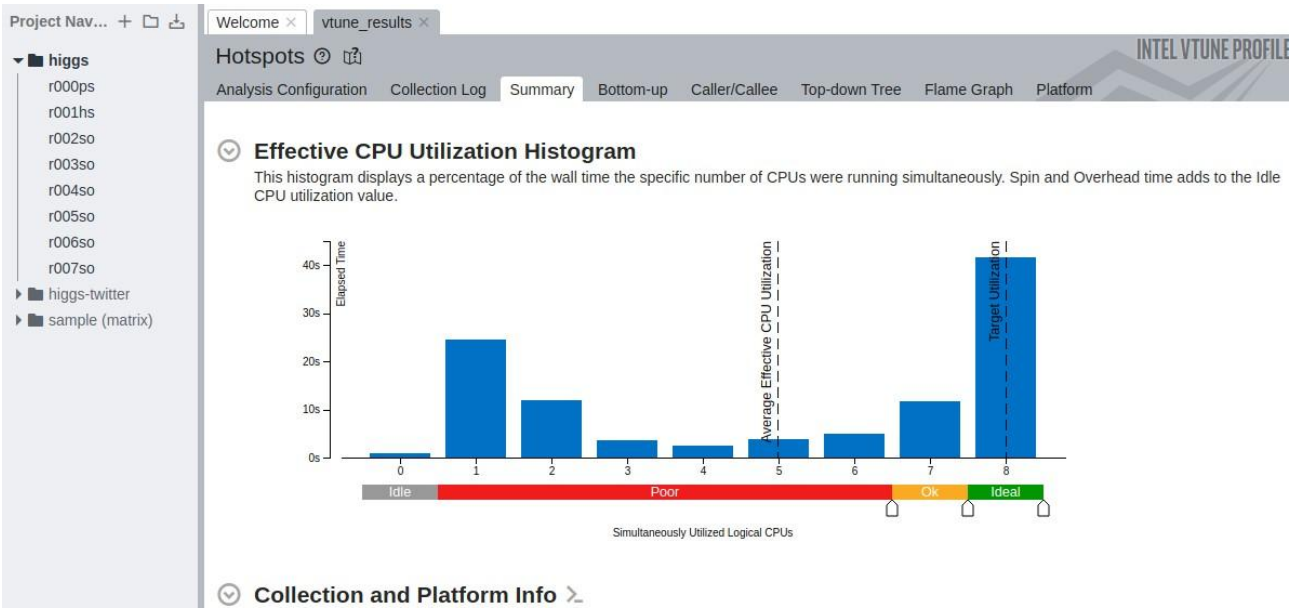
4.1 Performance Metrics

Implementation	Execution Time (seconds)	Speedup
Serial	285.497	1×
MPI	68.20	~4.18×
OpenMP + MPI (Hybrid)	62.50	~4.57×

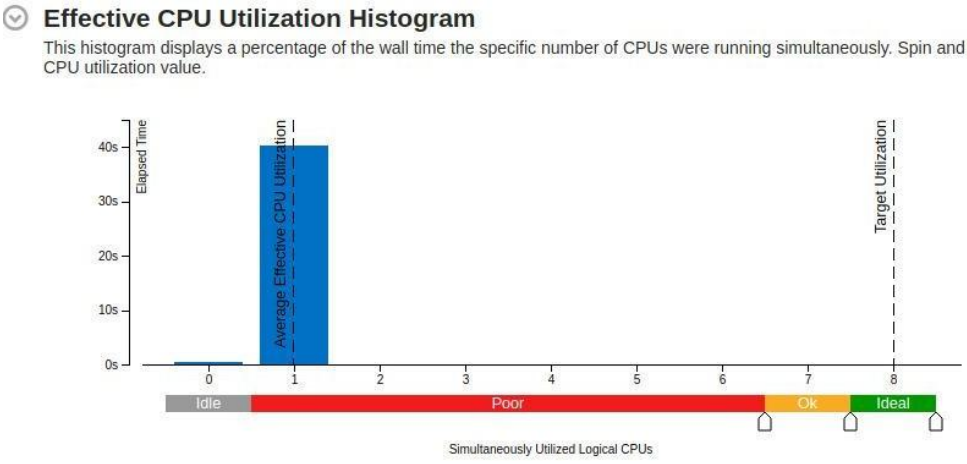
- The MPI version provides a **~4.18× speedup** over the serial version by distributing tasks across processes.
- The hybrid model further reduces computation time by parallelizing inner loops and local data processing using OpenMP, yielding a **~4.57× speedup**.

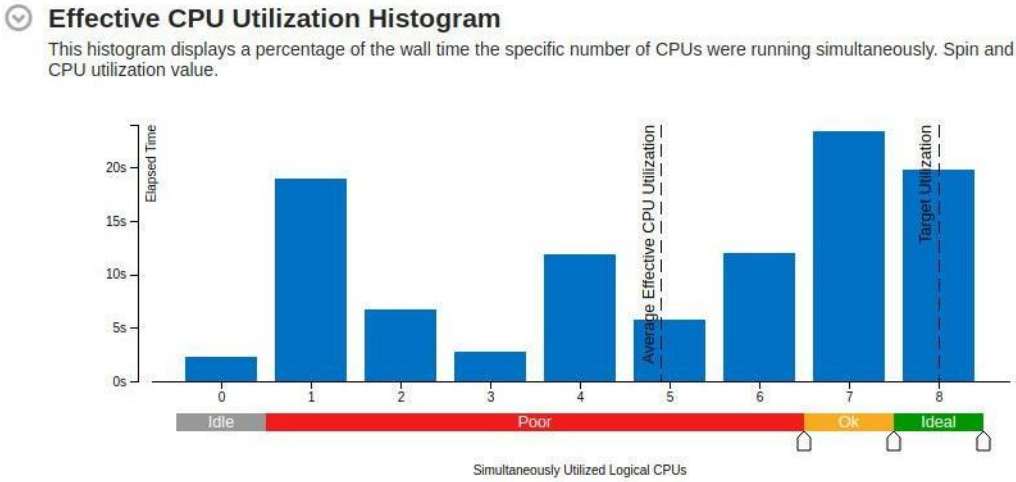
Twitter-Higgs:




Project Nav...
higgs
r000ps
r001hs
r002so
r003so
r004so
r005so
r006so
r007so
higgs-twitter
sample (matrix)
Welcome
vtune_results
Hotspots
INTEL VTUNE PROFILE
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Flame Graph
Platform
Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.
Elapsed Time
40s
30s
20s
10s
0s
Average Effective CPU Utilization
Target Utilization
0
1
2
3
4
5
6
7
8
Idle
Poor
Ok
Ideal
Simultaneously Utilized Logical CPUs
Collection and Platform Info

Twitter -Ego:


Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.
Elapsed Time
40s
30s
20s
10s
0s
Average Effective CPU Utilization
Target Utilization
0
1
2
3
4
5
6
7
8
Idle
Poor
Ok
Ideal
Simultaneously Utilized Logical CPUs


Effective CPU Utilization Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.
Elapsed Time
20s
15s
10s
5s
0s
Average Effective CPU Utilization
Target Utilization
0
1
2
3
4
5
6
7
8
Idle
Poor
Ok
Ideal
Simultaneously Utilized Logical CPUs
Collection and Platform Info

4.3 Scalability Discussion

- **Strong Scalability**: Execution time decreases with increased processes/threads up to a certain point. The hybrid model showcases improved scalability due to reduced thread contention.
- **Communication Overhead**: MPI performance plateaus when inter-process communication outweighs computation time, especially with small partition sizes.
- **Load Imbalance**: Despite METIS's efficiency, some partitions may contain denser subgraphs, causing variance in execution time among processes.

---

5. Discussion

The parallelization of PSAIIM significantly reduces execution time, making it suitable for large-scale social network analysis. However, the hybrid model’s performance gain over pure MPI is relatively modest (~8%), which indicates that communication remains a bottleneck. Future work could optimize data locality, overlap communication with computation, and implement dynamic load balancing.

---

6. Conclusion and Future Work

This report demonstrates that the PSAIIM algorithm benefits greatly from parallel execution. The hybrid MPI + OpenMP model offers the best performance, reducing execution time by over **78%** compared to the serial version.

Future directions include:

- GPU acceleration using CUDA/OpenCL
- Dynamic graph support (influence tracking over time)
- Incorporating deeper semantic interest matching using NLP techniques

```
top - 23:05:02 up  1:06,  1 user,  load average: 3.98, 2.05, 0.87
Tasks: 324 total,   5 running, 319 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.2 us,  0.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3868.6 total,   1969.7 free,   1332.3 used,    566.5 buff/cache
MiB Swap:   2140.0 total,   2070.8 free,     69.2 used.   2251.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  13518 mpi       20   0   32516  26256   3456 R  50.7   0.7   1:32.84 main
  13517 mpi       20   0   32516  26128   3328 R  48.4   0.7   1:32.81 main
  13519 mpi       20   0   32516  26072   3328 R  48.4   0.7   1:32.31 main
  13516 mpi       20   0   32516  26172   3328 R  48.0   0.7   1:32.44 main
```

7. References

1. SNAP Higgs Twitter Dataset. Stanford Network Analysis Project (SNAP).

2. MPI Standard Documentation.

3. OpenMP API Specification.

4. network https://drive.google.com/file/d/1vp5he-8ogdPJRFPiT6BDGPVZoH9CEnAh/view?usp=sharing