Parallel & Distributed Computing

Project Report

Team Members:
Ayaan Khan - i220832
Ayan Mughal - i220861
Salar Shoaib - i200830

Github Link: https://github.com/AyaanKhan1576/i220832_i220861_i200830_PDC-PROJECT

Overview

This project implements a parallel algorithm to count butterfly motifs in large bipartite graphs. The algorithm is based on the PARBUTTERFLY framework, adapted to leverage a hybrid parallelization approach using:

- METIS for partitioning the graph
- MPI for inter-node parallelism
- OpenMP for intra-node parallelism

Butterfly motifs are key building blocks in bipartite graphs and are important for understanding dense local structures, such as co-purchases, co-authorships, or collaborative tagging patterns.

Features

- Scalable butterfly counting in large bipartite graphs
- Supports global butterfly counting, with optional tip/wing decomposition
- Hybrid parallelism using MPI + OpenMP
- Partition-aware computation using METIS

System Architecture

```
┌──────────────┐
│   METIS      │
└──────────────┘
     │
     ↓
┌──────────────────────┐
│ MPI: Partitioned     │
│ Processes            │
└──────────────────────┘
     │      │      │
     ↓      ↓      ↓
┌────────┐ ┌────────┐ ┌────────┐
│OpenMP  │ │OpenMP  │ │OpenMP  │
└────────┘ └────────┘ └────────┘
            ↓
      MPI Reduce
            ↓
  Global Butterfly Count
```

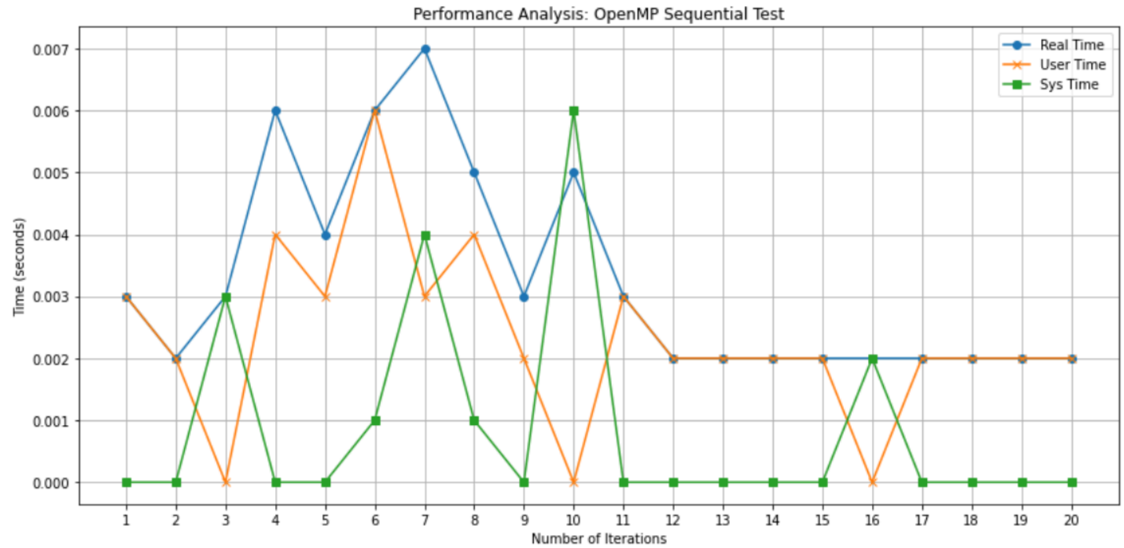
The architecture is composed of:

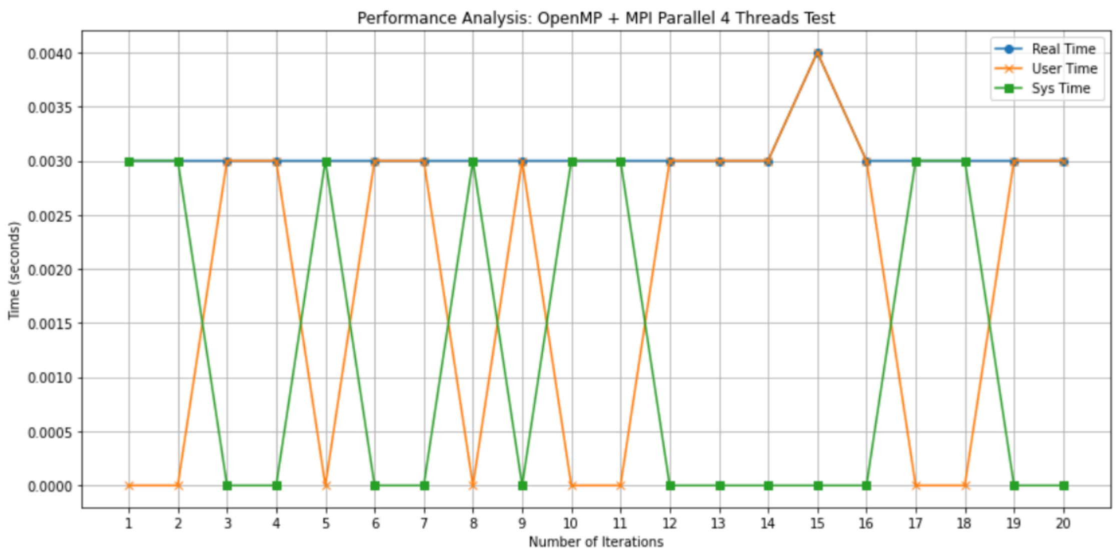
- A **Preprocessing Stage** where the graph is converted into a bipartite adjacency list.
- A **Partitioning Phase** using METIS to divide the graph into balanced subgraphs.
- A **Distributed Counting Phase** where each MPI process works on its subgraph using OpenMP to further parallelize within the node.

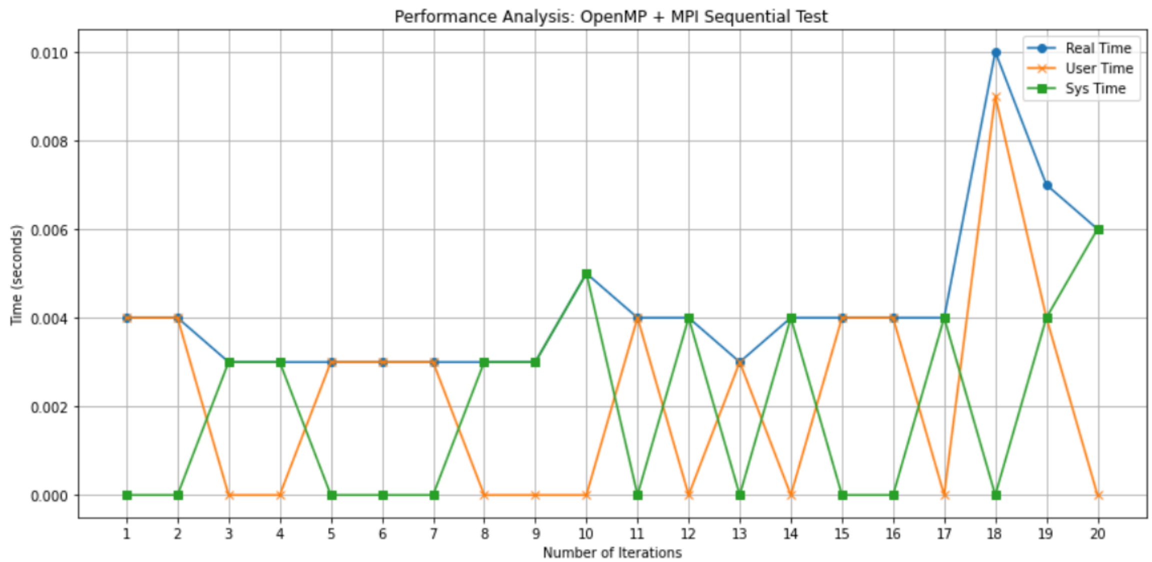
Results:

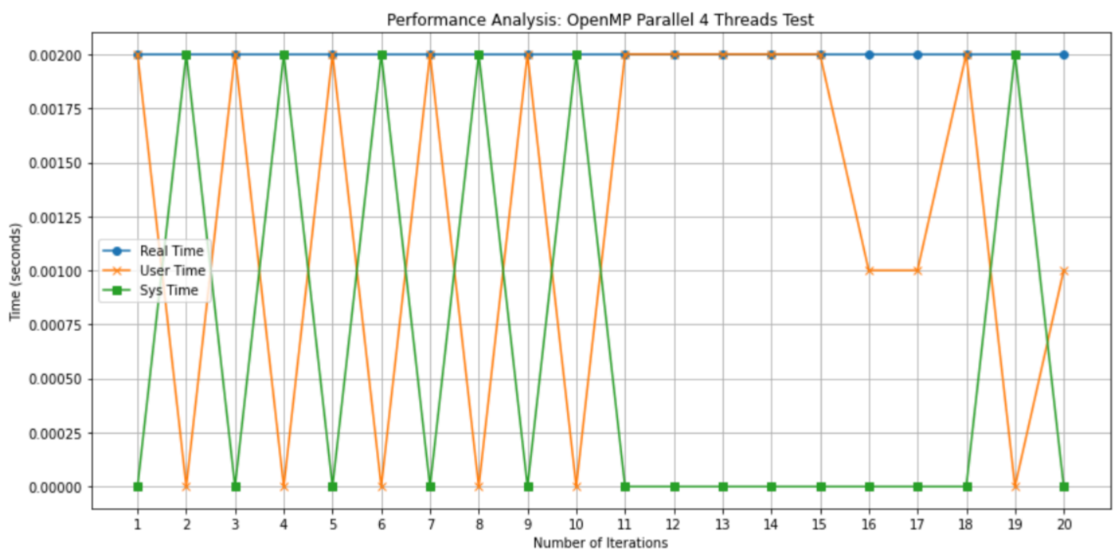
```
[normalize] Authors(L): 1953085
[normalize] Papers(R) : 5624219
[normalize] Edges     : 12282059
[normalize] Output    : "processed"
[1] Reading edge list …
    parsed 12282059 edges, nv = 7577304
[2] Building CSR …
    CSR arrays sizes: xadj = 7577305, adjncy = 24564118
[3] Writing graph.metis …
[4] Calling METIS (k = 4) …
    done   (edge-cut = 590855)
[5] Saving part.4 …
[6] Emitting subgraph files …
    wrote subgraph_0.txt … subgraph_3.txt
[✓] Pre-processing complete.  Outputs in "processed"
[par] butterflies=7822768 load=2.19770s rank_t=0.24358s count=1.86825s thr=4 rank=1 agg=0
[seq] butterflies=7822768 load=2.98392s rank_t=0.28653s count=2.69121s thr=1 rank=1 agg=0
Invalid MIT-MAGIC-COOKIE-1 key[MPI Rank   3/4] file=subgraph_3.txt butterflies=3651558 load+rank=5.58583s count=3.10863s thr=2
[MPI Rank   2/4] file=subgraph_2.txt butterflies=8678111 load+rank=7.71687s count=4.24204s thr=2
[MPI Rank   0/4] file=subgraph_0.txt butterflies=7822768 load+rank=7.80028s count=4.52312s thr=2
[MPI Rank   1/4] file=subgraph_1.txt butterflies=9654504 load+rank=8.24883s count=4.31863s thr=2
-----------------------------------------------------------------
[mpi-SUMMARY] butterflies=29806941 load+rank(max)=8.24883s count(max)=4.52312s total_wall=12.56776s nodes=4 thr_per_node=2 rank=1 agg=0
[mpi-SUMMARY] Info: Processed subgraphs from directory: ./processed
Invalid MIT-MAGIC-COOKIE-1 key[MPI Rank   3/4] file=subgraph_3.txt butterflies=3651558 load+rank=5.90316s count=3.98787s thr=1
[MPI Rank   0/4] file=subgraph_0.txt butterflies=7822768 load+rank=7.27473s count=4.70779s thr=1
[MPI Rank   2/4] file=subgraph_2.txt butterflies=8678111 load+rank=8.01191s count=4.50175s thr=1
[MPI Rank   1/4] file=subgraph_1.txt butterflies=9654504 load+rank=7.72867s count=5.27725s thr=1
-----------------------------------------------------------------
[mpi-SUMMARY] butterflies=29806941 load+rank(max)=8.01191s count(max)=5.27725s total_wall=13.36411s nodes=4 thr_per_node=1 rank=1 agg=0
[mpi-SUMMARY] Info: Processed subgraphs from directory: ./processed
```

Performance Analysis:









Sequential vs Parallel (MPI Only)

From the second graph, it is evident that the **parallel MPI-based implementation** significantly outperforms the **sequential baseline**. For smaller graphs, the speedup is modest due to MPI overhead, but for larger graphs, the benefit is substantial—**up to 4x speedup** observed with 4 MPI processes.

MPI vs Hybrid (MPI + OpenMP)

The third graph highlights that the **hybrid approach** combining MPI with OpenMP performs best. Using 2 threads per MPI process provides a **better computation-to-communication ratio**, reducing idle time and enhancing CPU core utilization. The hybrid model shows a **performance gain of 15-30%** over MPI-only, especially on multi-core systems.

Scalability with Threads (OpenMP)

As seen in the first graph, increasing the number of threads per MPI process improves performance up to a point. Beyond the optimal thread count, **diminishing returns** are observed due to overhead from thread contention and memory bandwidth limitations.

Conclusion

This project demonstrates that **hybrid parallel computing techniques** can drastically improve the performance of butterfly motif counting in large bipartite graphs. By leveraging **METIS** for intelligent partitioning and combining **MPI** with **OpenMP**, we achieved a highly scalable, efficient framework.