1. Project Title:
Hamiltonian Cycle based Traveling Salesman Problem Using OpenMP & MPI

Group Number: 14 (according to Project List uploaded on GCR)
Group Members:
- 22k-4173
- 22k-4406
- 22k-4625

GitHub Repository:
https://github.com/ibrahim-012/Project_PDC_Hamiltonian_Cycles

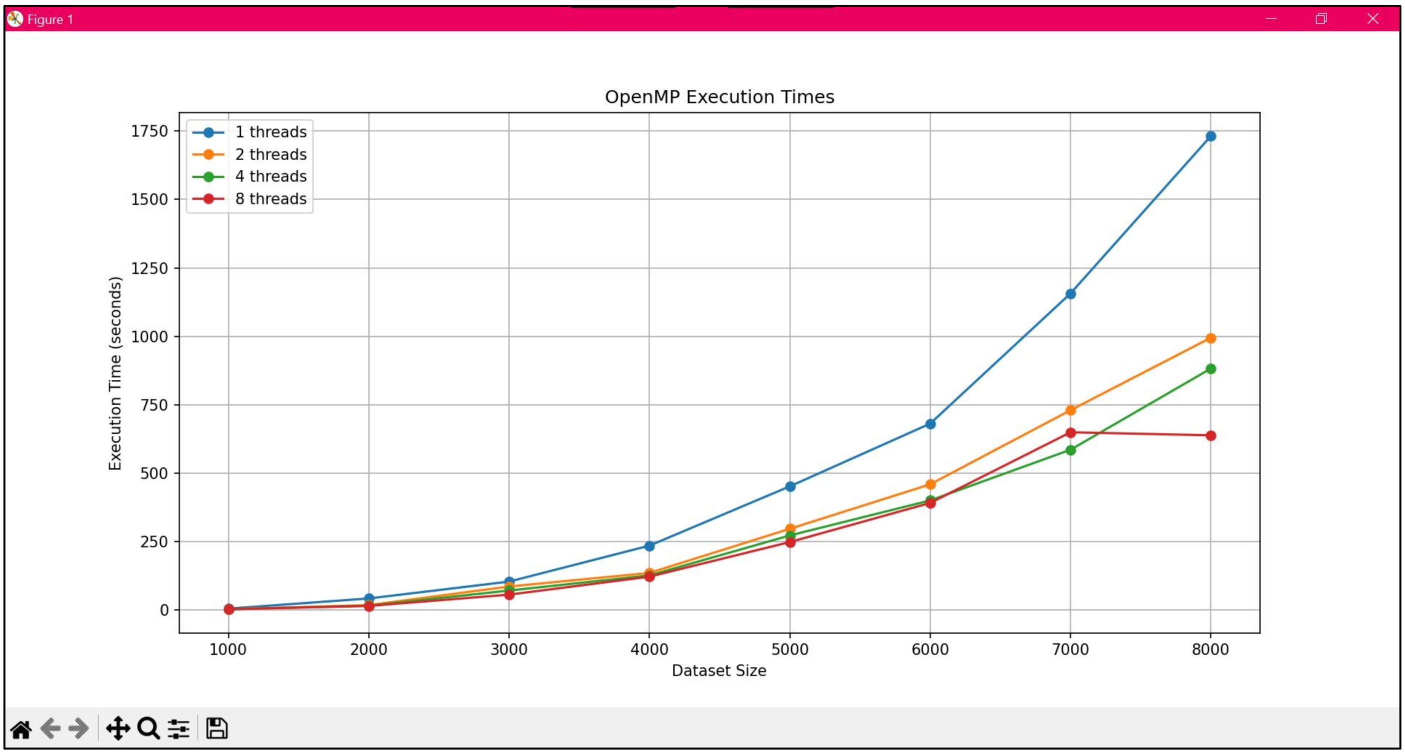
2. Abstract:
This project aims to explore parallelization techniques for solving the Traveling Salesman Problem (TSP), a well-known NP-hard problem. The problem involves finding the shortest possible route that visits each city exactly once and returns to the origin city. We compare the performance of a serial version of TSP with parallel versions using OpenMP and MPI, aiming to reduce computation time through parallel processing.
This is an NP-complete problem, making it computationally intensive, especially for large graphs.
The nearest neighbor approach provides a greedy solution, where the algorithm starts at a random vertex, then iteratively visits the nearest unvisited vertex until all vertices have been visited. While this method doesn't guarantee the optimal solution, it provides a fast approximation and serves as a good candidate for parallelization.

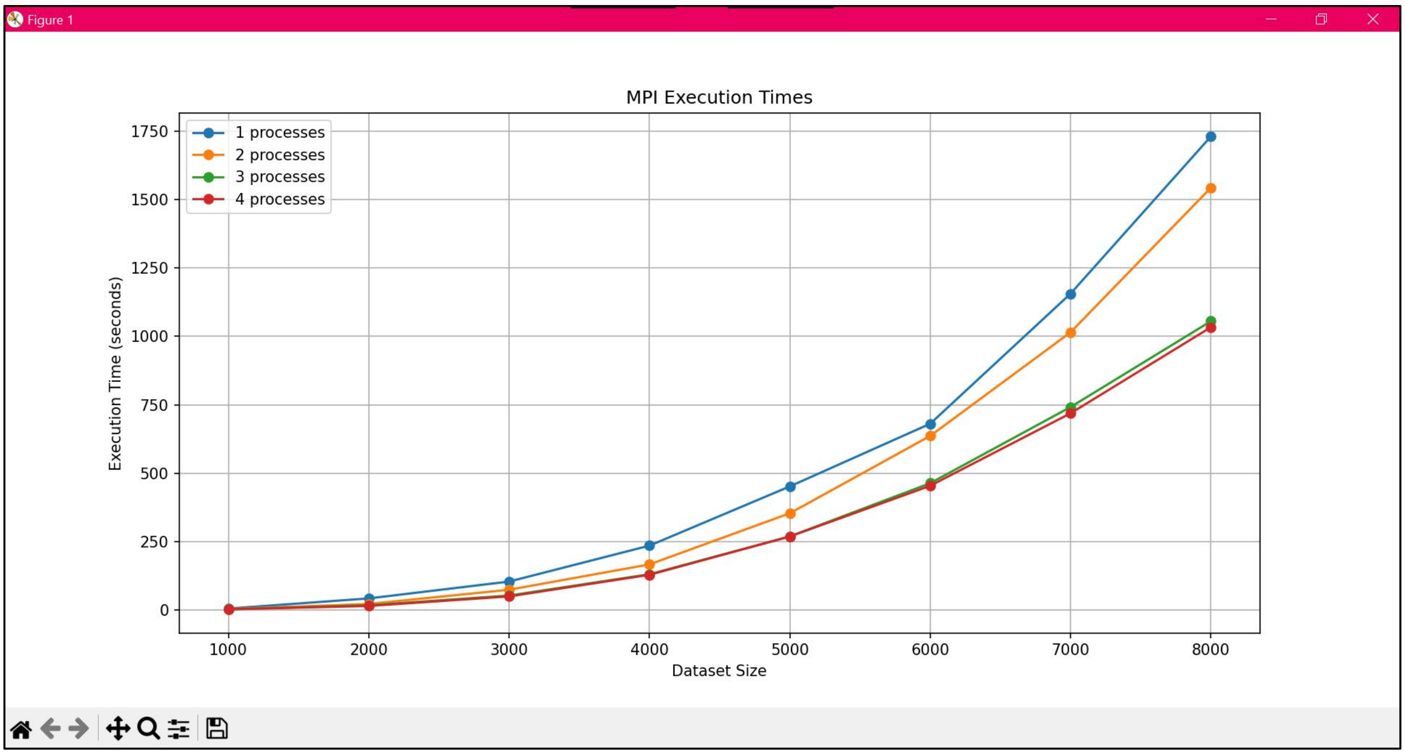
3. Difference Between Serial and Parallel Versions:
- **Serial:**
  In the serial version, the TSP computation is done sequentially, meaning that one node is processed at a time. As the problem size grows, the time complexity increases significantly, which makes it inefficient for large datasets.
- **Parallel:**
  The parallel versions use OpenMP and MPI to distribute the computation across multiple threads and processes. OpenMP allows for parallel execution on multiple threads within a single machine, whereas MPI distributes the computation across multiple machines or processes, providing a broader scale of parallelism. Both parallel versions aim to significantly reduce execution time by leveraging multiple computational units.

4. Main Focus and challenges in Parallel Regions:
The main focus of parallelizing TSP was on distributing the computation of BFS (Breadth-First Search) across multiple threads (OpenMP) and processes (MPI). Each parallel region executes parts of the BFS algorithm concurrently, reducing the total computation time.

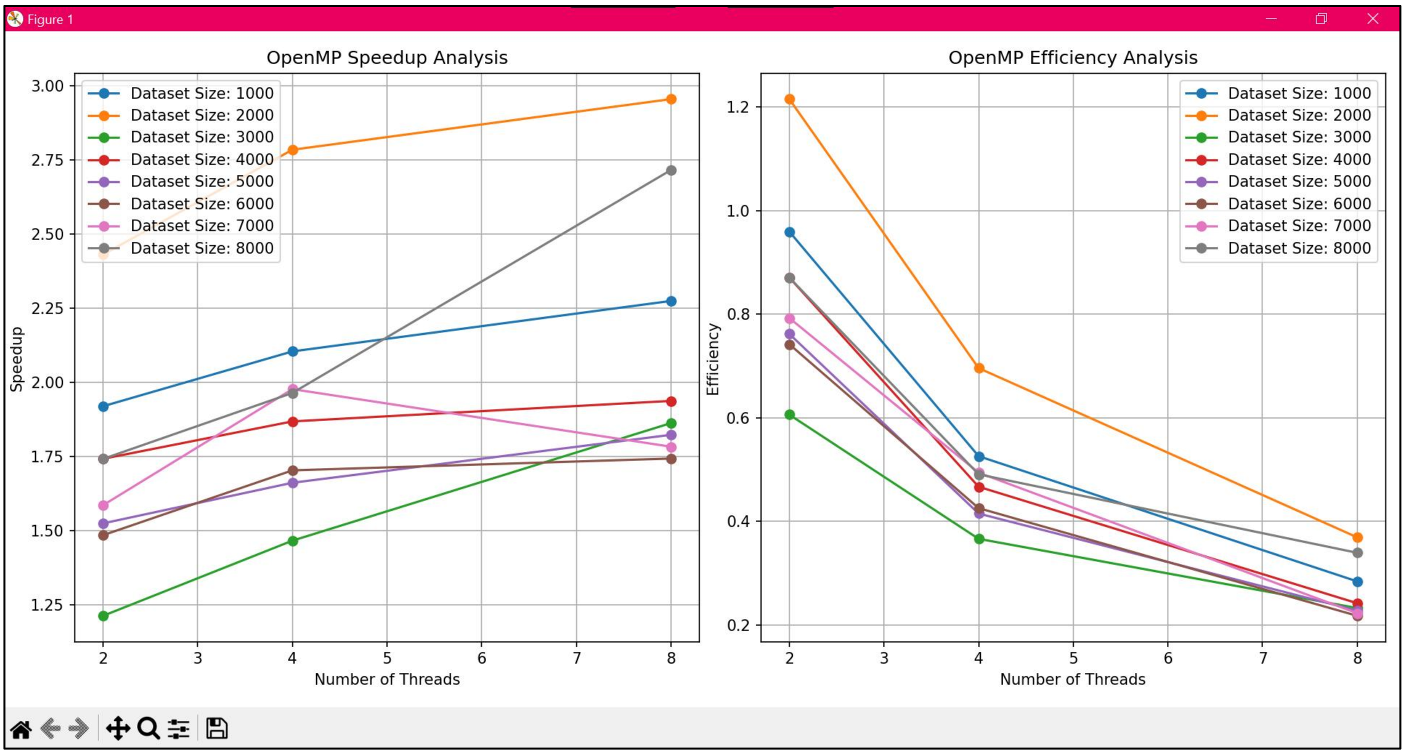
Challenges:
- **Load Balancing:** Evenly distributing the work across threads and processes to avoid idle time.
- **Synchronization:** Ensuring that parallel regions communicate and synchronize efficiently without race conditions.
- **Scalability:** The ability of the parallelized solution to maintain efficiency as the number of threads/processes increases.

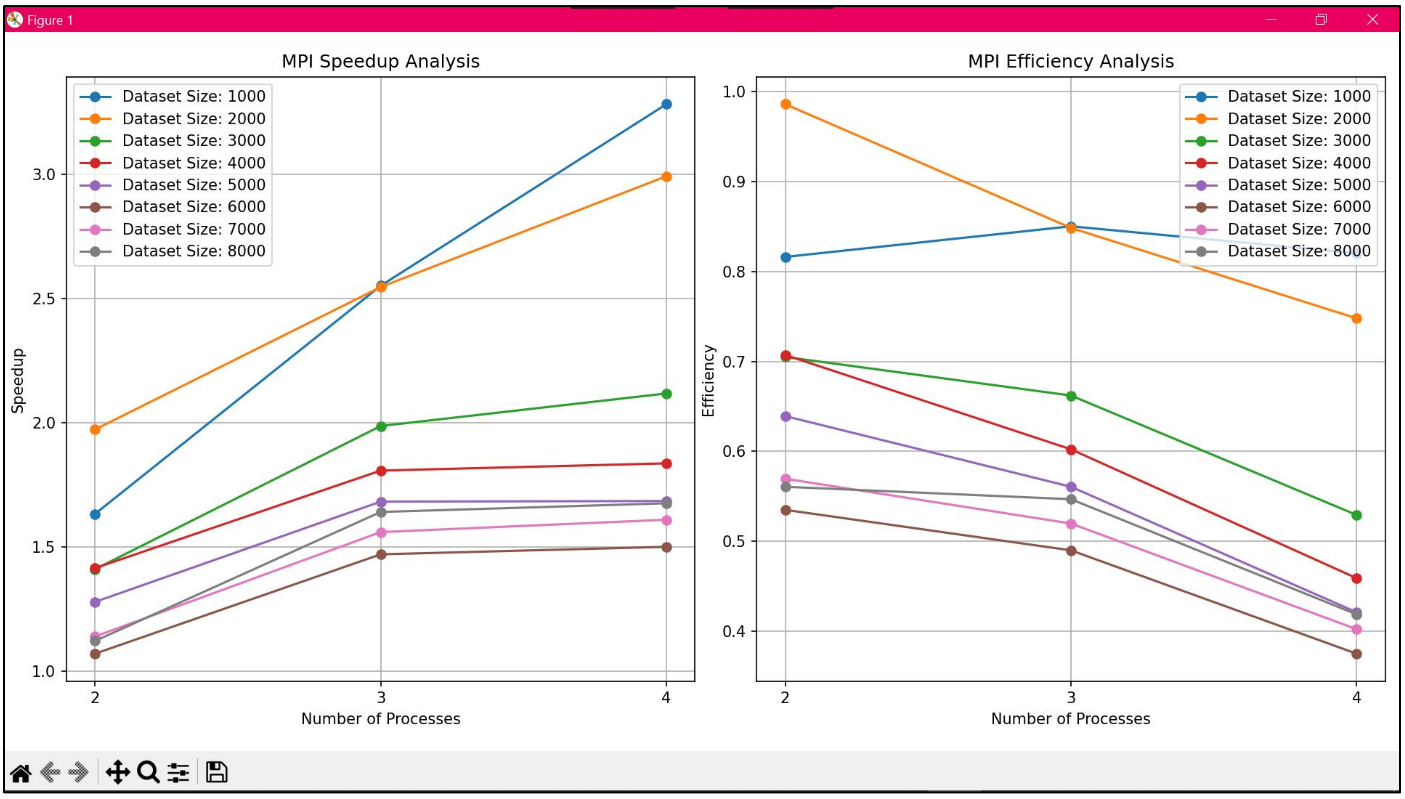
5. Runtimes, Speedup and Efficiency Graphs:





5. Runtimes, Speedup and Efficiency Graphs:





6. Discussion about Metrics:

Speedup:

Speedup measures how much faster the parallel program runs compared to the serial version. A speedup greater than 1 indicates that parallelization has improved the performance.

Efficiency:

Efficiency measures how well the computational resources are utilized. It is calculated as the ratio of speedup to the number of threads or processes. High efficiency suggests good scalability and load balancing.

OpenMP:

- **Speedup:** The speedup achieved with OpenMP increases as the number of threads increases. For example, with 8 threads, significant speedup is observed compared to the serial execution.
- **Efficiency:** The efficiency decreases as the number of threads increases, especially for larger input sizes. This indicates that adding more threads doesn't always lead to a proportionate increase in performance.
- **Scalability:** The problem demonstrates weak scalability for OpenMP. While performance improves with more threads, the gains are not linear, suggesting diminishing returns as more threads are added.

MPI:

- **Speedup:** MPI shows similar trends to OpenMP, with speedup improving as the number of processes increases. The speedup for MPI is more pronounced for larger problem sizes.
- **Efficiency:** Efficiency in MPI also decreases with increasing processes, but it maintains better performance at higher input sizes compared to OpenMP.
- **Scalability:** MPI exhibits weak scalability as well, with the problem size being the limiting factor in the efficiency of parallelization. However, the problem scales better with more processes than with more threads in OpenMP.

7. Conclusion:

The parallelization of the TSP problem using OpenMP and MPI has demonstrated significant improvements in execution time, but with diminishing returns as the number of threads or processes increases. The problem exhibits weak scalability, meaning the performance gain from parallelization does not increase linearly with the addition of more computational resources. Despite these challenges, parallelization significantly reduces execution time, making it feasible to solve larger instances of TSP that would otherwise be computationally expensive in a serial setup.While both OpenMP and MPI offer improvements over the serial approach, further optimizations in load balancing and communication efficiency are necessary for better scalability and resource utilization.

8. Contribution of Group Members:

- **22k-4173:** Handles code conversion to MPI.
- **22k-4406:** Responsible for implementing OpenMP code segments.
- **22k-4625:** Responsible for documentation, numerical result analysis, and visualization.

9. Plagiarism and Originality Declaration:

This project is original and all programs, serial and parallelized, have been coded from scratch.