University of Central Punjab

Faculty of Information Technology & Computer Sciences

Parallel and Distributed Computing, Spring 2025

Project

Name	Roll
Hussnain Amjad	**L1F21BSCS1176**
Muhammad Faisal Raza	**L1F21BSCS1005**
Subhan Khalid	**L1F21BSCS0869**
Muhammad Arbaz	**L1F21BSCS0523**
Hafiz Shawal Nadeem	**L1F21BSCSXXX**

1. Machine Specifications:

Core i5 6th Generation

- Dell Latitude E-5470
- 16GB Memory
- 256 SSD
- Intel (HD) Graphics 530
- 4 Processors
- 4 Logical Processors

2. Performance Comparison Table :

Implementation	Time Taken (seconds)
OpenMP	0.000507
Serial	0.005572
Pthreads	0.006060

3. Fastest Implementation:

OpenMP

Reason:

- **Low overhead:** OpenMP uses lightweight threads created by the compiler internally. There's no manual thread creation or joining.
- Efficient compiler optimization: The compiler automatically parallelizes loop iterations, optimizing thread workload distribution.
- Better cache usage: OpenMP threads typically work on adjacent data, reducing cache misses.
- Shared-memory parallelism: Best suited to your current single-machine WSL2 environment.

4. Slowest Implementation:

pthreads

Reason:

- **Manual thread creation overhead:** Each thread must be created and joined manually, which is costly for short execution tasks.
- **Thread synchronization cost:** Managing thread safety and memory sharing adds overhead.
- **No automatic workload balancing**: You must divide work manually, risking imbalance (e.g., uneven row allocation in grid).

5. Middle Performer:

Serial

Reason:

- **No parallelization:** It's straightforward, with no thread overhead.
- **Consistent execution:** Every line runs in order with low system load.
- **However:** Lacks any performance gain from concurrency.

6. MPI Functions Implementation

- **Send & Receive:** 2 processes, sent a value from rank 0 to 1.
- **Broadcast:** One value from rank 0 sent to all processes.
- **Reduce:** All processes contributed a value; sum sent to rank 0.
- **Gather:** Each process sent a value to rank 0, received as an array.
- **Scatter:** Rank 0 divided array and sent chunks to each process.

7. Summary

This project aimed to implement a traffic simulation module using three different computing paradigms: Serial, Pthreads, and OpenMP. It also required building a client-server architecture using MPI with all key functions like Send, Receive, Broadcast, Reduce, Gather, and Scatter. The goal was to analyze performance differences and understand parallel computing on both shared and distributed memory systems.