Project Documentation: Parallel Traffic Congestion Analyzer

1. Project Scope and Objectives

1.1. Scope

The goal of this project is to develop an advanced, high-performance traffic data analysis application capable of processing large datasets from multiple traffic lights concurrently. It moves beyond simple traffic volume counting to derive metrics that truly reflect the **intensity and volatility of congestion** at specific intersections, enabling better decision-making for urban planning and adaptive traffic signal control.

1.2. Objectives

1. **Enhance Analysis Beyond Total Counts:** Calculate sophisticated metrics, specifically the Average Congestion Index (ACI) and Max Car Spike.
2. **Achieve High Performance:** Utilize **MPI (Message Passing Interface)** for data distribution across multiple processes and **OpenMP** for thread-level concurrency within each process, ensuring low-latency processing of massive traffic logs.
3. **Produce Actionable Intelligence:** Generate a prioritized report ranking traffic lights by their chronic congestion (ACI) and volatility (Max Spike).

2. Design and Methodology

2.1. Parallel Architecture

The analyzer adopts a **Hybrid Parallelism Model**:

- **MPI (Distributed Memory):** Used for coarse-grained parallelism. The master process (Rank 0) reads the entire dataset and distributes chunks of data to all slave processes. This is ideal for scaling across multiple compute nodes.
- **OpenMP (Shared Memory):** Used for fine-grained parallelism. Each MPI process uses multiple threads to calculate its local metrics (Total Cars, Observation Count, Max Spike) concurrently, significantly speeding up the local aggregation phase.

2.2. Advanced Metrics

To surpass the capability of a simple total car count analysis, two key metrics are introduced:

Metric	Calculation	Purpose
Average Congestion		Represents the average

Index (ACI)		*intensity* of traffic per measurement interval. This prevents frequently measured lights from artificially dominating the rankings based on raw volume.
Max Car Spike	Maximum single recorded at the light.	Identifies **traffic volatility** and lights prone to extreme, short-term congestion events (spikes). This is critical for planning preventative or adaptive signal adjustments.

2.3. Data Flow Diagram

The process follows a standard parallel reduction pattern:

1. **Scatter:** Master (Rank 0) reads the data file, determines the distribution sizes (sendcounts, displs), and uses MPI_Scatterv to send chunks of raw TrafficData to all processes (Master and Slaves).
2. **Local Processing:** Each process executes process_local_data. This function uses **OpenMP Thread-Local Storage (TLS)** to aggregate metrics into separate maps per thread before a sequential merge, minimizing expensive locking or atomic operations.
3. **Gather:** Each process serializes its local LocalMetrics map into a vector of AggregatedData structs.
4. **Reduce/Finalize:** Master uses MPI_Gatherv to collect all AggregatedData structs into a single global vector. The master then aggregates these local results, calculating the global sums for total cars and observations, and the global maximum for the Max Car Spike.
5. **Report:** Master sorts the results by ACI and prints the final report.

3. Implementation Details

3.1. Data Structures and MPI Type Creation

A custom MPI derived datatype (MPI_AGGREGATED_DATA) is mandatory for the collection phase (MPI_Gatherv) because the AggregatedData struct contains mixed data types (int, long long, int, int). The offsetof macro is correctly used to ensure accurate structure alignment across different systems.

3.2. OpenMP Thread-Local Aggregation

The process_local_data function employs OpenMP to parallelize the heavy data aggregation loop. Instead of using a global critical section (which would serialize the work), it utilizes **Thread-Local Storage (TLS)**:

1. A std::vector<std::map<int, LocalMetrics>> thread_maps is created, one map per thread.
2. The main loop is parallelized (#pragma omp for) and threads update their own unique map (thread_maps[tid]).
3. A final, fast **sequential merge** combines the results from the thread-local maps into the single local_congestion_map.

3.3. Final Analysis Logic

The analyze_congestion function on the Master performs the final, critical reduction:

- It iterates through the global_results (collected from all processes).
- For total_cars and observation_count, it uses a **summation (aggregation)**.
- For max_spike, it uses a **maximum reduction** (finding the single highest spike observed across all processes and time points).
- It then calculates the ACI and sorts the intersections by this new, congestion-intensity-focused metric.

4. Evaluation and Example Run

4.1. Example Execution Report

The following output demonstrates a successful execution using **4 MPI processes ()** on a dataset containing 2,389 records. The report highlights the top 5 most congested lights, ranked by their **Average Congestion Index (ACI)**, a measure of chronic traffic intensity.

```
Master: Read 2389 total records. Distributing among 4 processes.

===========================================================================
=
      Top 5 Traffic Lights by Average Congestion Index (ACI)
===========================================================================
=
Rank        Traffic Light ID    ACI (Avg Cars)       Max Car Spike
-----------------------------------------------------------------------
1           205                 73.21                157
2           550                 66.22                132
3           402                 61.99                142
4           101                 57.54                133
5           310                 50.60                163
===========================================================================
```

=

--- Performance Summary (P=4) ---
Total Wall Clock Time: 0.005547 seconds
Local Processing Time (per process estimate): 0.000357 seconds
----

4.2. Analysis of Results

- **Congestion Hotspots:** Traffic Light **205** is identified as the most chronically congested intersection with an ACI of **73.21**. This indicates the highest average traffic intensity.
- **Volatility Indicator:** Light **310** has the highest **Max Car Spike** (**163**), indicating it experiences the most acute, short-term traffic surges, despite ranking lower in chronic intensity (ACI: 50.60). This suggests a volatile location that needs real-time signal adjustments.
- **Performance:** The total wall clock time of approximately **5.5 milliseconds** on a dataset of 2,389 records demonstrates the high efficiency of the hybrid MPI/OpenMP parallel approach. The extremely low local processing time confirms the successful optimization using **Thread-Local Storage (TLS)**.

4.3. Future Improvements

- **Variance Calculation:** Adding a **standard deviation** metric to measure how much the car count deviates from the mean (ACI). This would be an even better indicator of volatility than the simple Max Spike.
- **Time-Series Analysis Integration:** Integrating the initial hourly grouping logic (from the prior project) into the parallel processing, allowing the ACI and Max Spike to be calculated on a per-hour basis for even more detailed temporal analysis.
- **Fault Tolerance:** Implementing a basic mechanism to check for non-zero return codes from file operations or MPI calls to ensure graceful handling of system errors.