CSS 311 - Parallel and Distributed Computing

Title: Parallel Implementation of Traffic Accident Analysis

Name 1: **Allu Pranathi** Roll No: **2023BCD0065**
Name 2: **Kavala Sai Sri Neharika** Roll No: **2023BCD0053**

Course Instructor: Dr. John Paul Martin
Department of Computer Science, Indian Institute of Information Technology Kottayam (IIIT Kottayam)

Date of Submission: 11/10/2025

Abstract

Our Project “Parallel Traffic Accident Analysis” mainly focuses on processing and analyzing large datasets which contain the accident details such as accident cause, severity, weather conditions and traffic density.

In serial Implementation, first it reads a csv file (our dataset) and processes each record sequentially to get the results like total injuries, fatalities, frequency of causes and conditions, etc. But this approach takes a lot of time for analyzing large real-world datasets because it is following sequential analysis.

To solve this issue, we developed an OpenMP based parallel version. So, basically we will divide the accident records among multiple threads, where each thread can be independently compute sum for total injuries, fatalities, frequency of conditions, etc. And these sums are merged to global sum using race condition handling techniques.

Our Serial Implementation takes “0.622 sec“ time for analysis. And the parallel version takes “0.13 sec” for analysis. This indicates the parallel version is **4.7** times faster than serial.

1.Introduction

As we know now a days the number of motor vehicles are increasing,which leads to traffic becoming severe.It is estimated that over half of the traffic crowd is caused by Incidents such as accidents,construction and maintenance activities,weather conditions.Traffic accidents are a primary concern for the public, because it causes major loss of human lives and economy.

To reduce the accidents we will take a dataset of previous accidents which gives information about the accidents like how many accidents happened in a particular month (or) place.Usually it takes more execution time without parallelising because the data size is large.

Parallel and distributed computing gives an efficient solution by dividing the work with multiple threads which reduce the execution time which makes the work easier.This project gives information about basic statistics i.e total accidents ,injuries average vehicles per accident and Key Distributions like severity ,accident cause ,traffic density ,which will reduce the accident rate.

This “Parallel Traffic Accident Analysis” can directly fits into **Real-world Transportation systems** and **public safety applications.** Analyzing this data can helps in identifying high risk zones and understanding the major causes of accidents. Which is very helpful in day-to-day life.

2. Literature Survey

Here we are referring to the previous work thesis “**Freeway Accident analysis using Second order Statistics**”.

(We provided this thesis link in the reference section).

Summary of the thesis:

In this thesis, the freeway accident data is analyzed using a serial (sequential) approach. They started the process with collecting accident and traffic information like speed, volume, etc from a **SQL server** database. This collected data is then manually extracted and processed for each accident separately.

Next, a **Visual Basic(VB)** program was used to calculate the 3 second-order statistical measures (Angular Second Moment (ASM), Contrast and Entropy). These 3 statistical measures describe the smoothness, randomness and variation in traffic patterns before and during an accident.

Table 5: ASM distribution for the traffic conditions for the whole accident data set

Serial No	Range	Frequencies
1	0.1-0.19	22268
2	0.2-0.29	12589
3	0.3-0.39	10515
4	0.4-0.49	7399
5	0.5-0.59	3079
6	0.6-0.69	2086
7	0.7-0.79	1578
8	0.8-0.89	1231
9	0.9-.99	837
10	0.99-1.0	3604

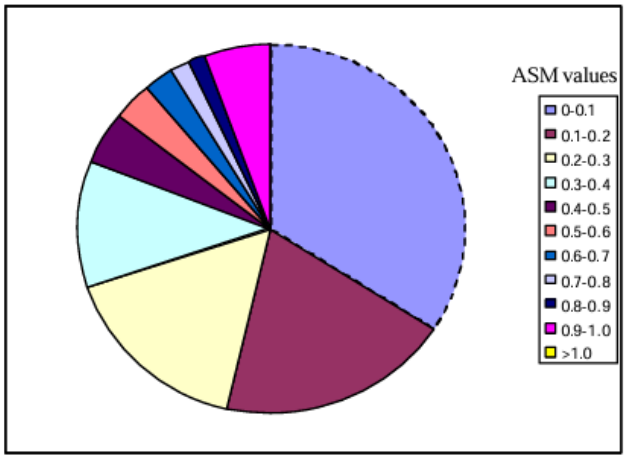


Figure 14: ASM distribution of the whole accident data set

But these calculations are done sequentially, means only one accident record is processed at a time.

After this, all these results are stored in an Excel file called "**Run-On File**". This excel file contains time, station, accident number, ASM, contrast, and entropy, Group (0 or 1).

Group 0: Accident occurrence condition traffic speed contour maps could be created.
Group 1: Precursory traffic condition traffic speed contour maps could be created.

TIME	GR_X	STA	ASM	CONT	ENT	ACC No	ACC DA
10:55	0	36	0.256327	0.828571	2.329642	9	4/8/1999
10:56	0	36	0.227245	0.9	2.404899	9	4/8/1999
10:57	0	36	0.27102	0.828571	2.23168	9	4/8/1999
10:58	0	36	0.242347	1.014286	2.398674	9	4/8/1999
10:59	0	36	0.202755	1.1	2.531248	9	4/8/1999
11:00	0	36	0.171939	1.142857	2.748202	9	4/8/1999
11:01	0	36	0.177755	1.085714	2.721153	9	4/8/1999
11:02	0	36	0.181429	1.042857	2.699238	9	4/8/1999
11:03	0	36	0.167143	1.028571	2.808968	9	4/8/1999
11:04	0	36	0.161123	1.142857	2.850096	9	4/8/1999
11:05	0	36	0.166429	1.557143	2.962965	9	4/8/1999
11:06	0	36	0	0	0	9	4/8/1999
11:07	0	36	0	0	0	9	4/8/1999
11:08	0	36	0	0	0	9	4/8/1999
11:09	0	36	0	0	0	9	4/8/1999

Figure 18: Part of Run-on file

Limitations of this sequential approach:

- Processing one accident at a time makes the program slow.
- Here, the collected data is manually extracted. So, it depends on the person's knowledge for getting a better result.
- This sequential method is not that helpful for large datasets or multiple locations.

3. Problem Statement and Objectives

3.1 Problem Definition

The computational problem solved in this project is the **efficient analysis of large traffic accident datasets** to extract meaningful statistical details like accident frequency, severity distribution etc.

In serial Implementation, the project takes the data file as input which contains details about accident cause ,traffic density ,number of deaths from the particular accident ,weather and road condition, etc. With the given data it will give the total number of accidents ,injuries that happened. It will also give the peak time when more accidents are happening.
This is processed record by record.

So, The computational challenge is to:
Implement and design an efficient “parallelized traffic accident analysis” so that it can process large datasets using multi-core CPU like OpenMP for faster execution.

3.2 Objectives

1. To collect traffic accident dataset for analysis.
2. Implement serial code to give the estimation of accidents at particular time and place.
3. Similarly implement parallel execution to give the estimation of accidents at particular time and place.
4. Compare the performance of both serial and parallel implementations.
5. To evaluate scalability and efficiency with different dataset sizes.

4. Methodology and System Architecture

Workflow:

Dataset Description:

For this project, we created our own dataset “india_traffic_accidents.csv” containing 10 lakh accident records which is exactly similar to real world accident datasets.
(Because, the available real world datasets from kaggle are too large (the file sizes are approximately 5 to 6 GB))

Our dataset Columns description:

- **id**: Unique accident number
- **date:** Date of the accident (YYYY-MM-DD)
- **time**: Time of the accident (HH:MM)
- **latitude / longitude:** Geographic coordinates of the accident location
- **severity:** Accident severity level (e.g., minor, major, fatal)
- **road_condition:** State of the road (e.g., dry, wet, icy, potholes)
- **weather:** Weather during the accident (e.g., clear, rain, fog)
- **vehicles_involved**: Number of vehicles in the accident
- **injuries:** Number of non-fatal injuries
- **fatalities:** Number of deaths resulting from the accident
- **accident_cause:** Primary cause (e.g., speeding, distraction, poor visibility)
- **traffic_density**: Level of traffic flow (e.g., low, medium, high)
- **lane_utilization:** Measure of congestion per lane
- **nearby_accidents:** count of nearest accidents

Implementation Process:

For serial:

We started our implementation by reading the dataset “india_traffic_accidents.csv”. This reading is done with the **readData()** function.
This function opens the file and it skips the header and reads all the records one by one.

During reading, csv file is comma separated so it separates the data by commas and removes extra spaces and it converts the text data into numbers wherever there is needed (like for vehicles, fatalities and injuries.) (Because in csv files all variables are stored as string).

Each accident record is now stored in a structure **Accident.** And all these records are stored in a vector. (**vector<Accident>**)

After the data is loaded, we move to the Analysis phase. The **analyze()** function processes this data to calculate important statistics like total injuries, fatalities and the average number of vehicles per accident.
It also counts how many accidents occurred under different conditions like severity, weather, cause, traffic density.
For peak accident hour, from the dataset it extracts the time column and identifies which hour has more accidents.

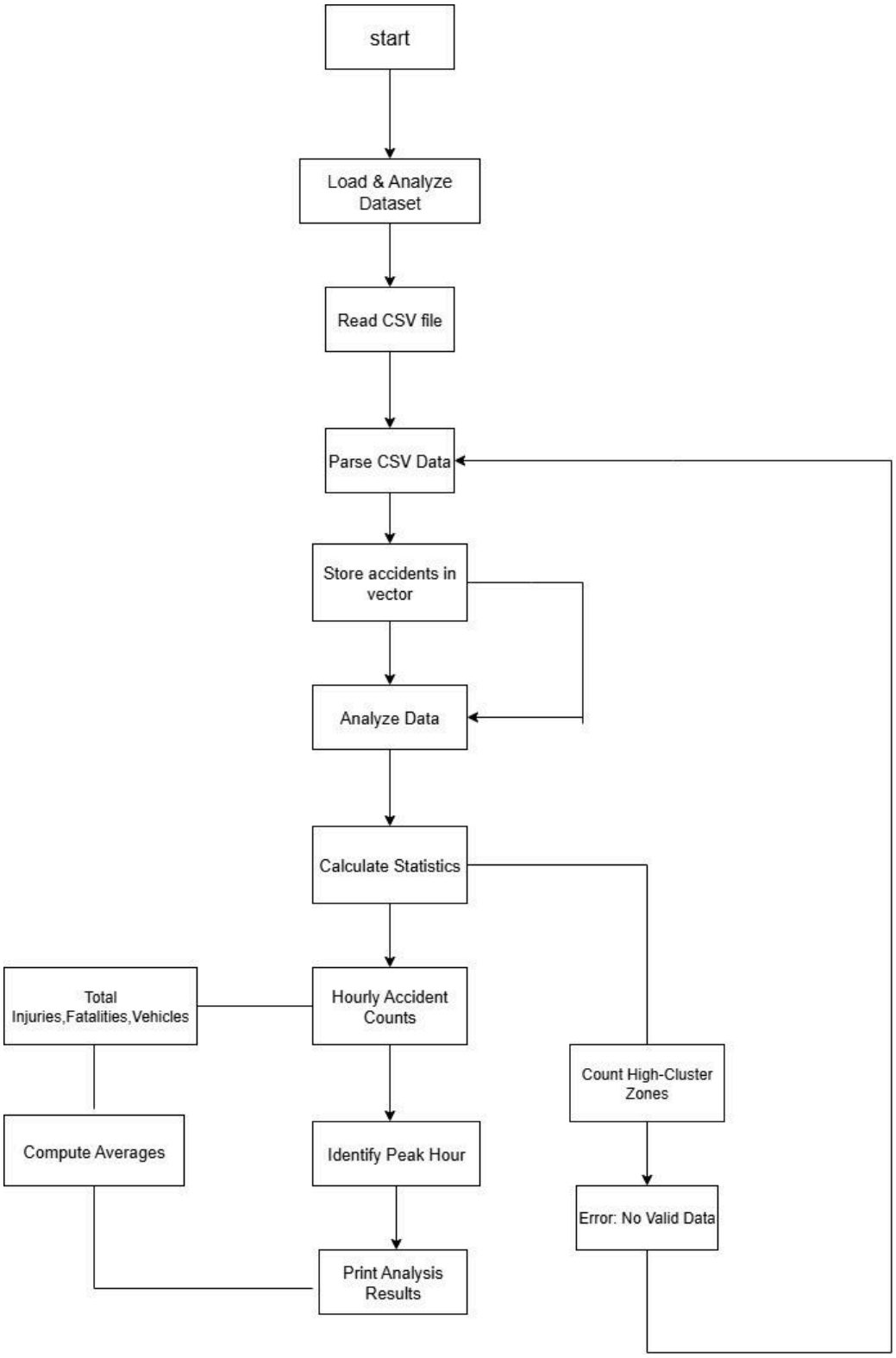
Finally, we provided an interactive mode for the user, to enter the dataset file to calculate all these results for the user provided data.

<u>For parallel</u>:

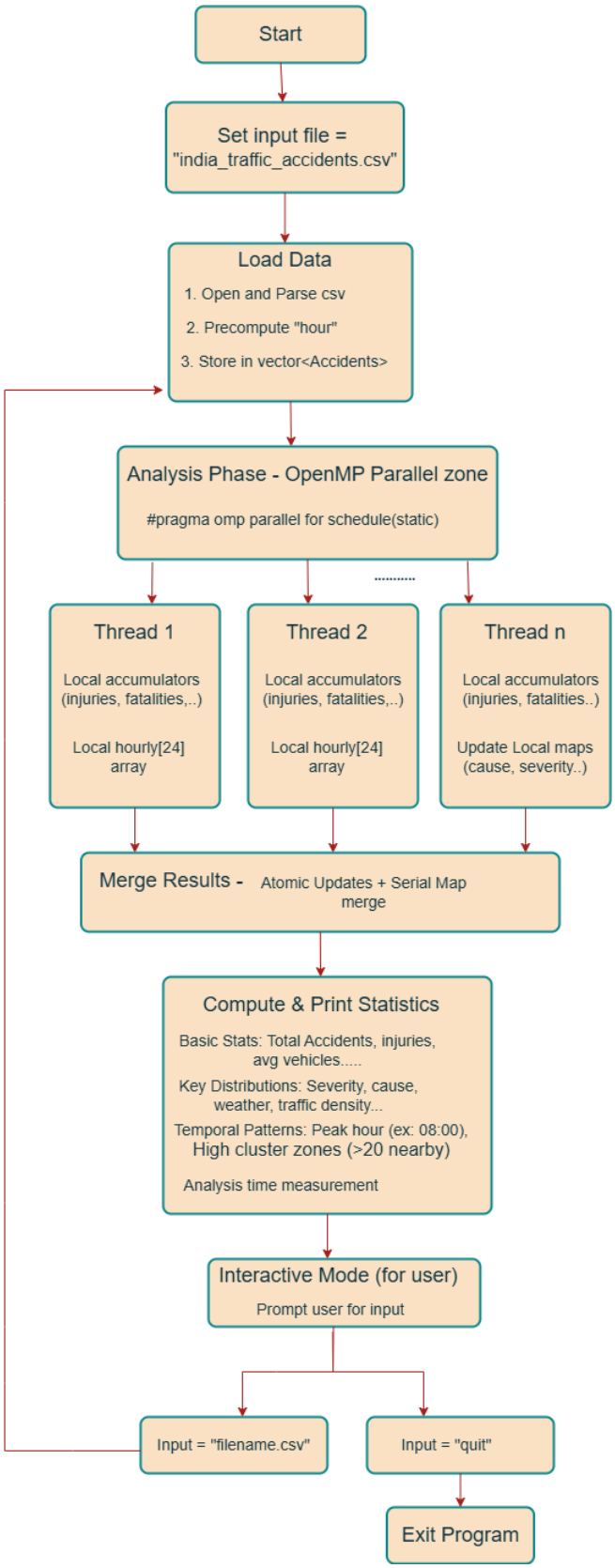
For parallel implementation, we just modified mainly the **analyze()** function. This function divides the computation of results among multiple threads. Each thread processes a portion of accident records and these threads maintain their own local variables to get totals for injuries, fatalities, vehicles, nearby accidents, and hourly counts.
Now, these results are combined into global totals using atomic. (**#pragma omp atomic**)

And for categorical data like severity, accident cause, traffic density, weather conditions, each thread maintains their own local **unordered maps** (local_sev, local_cause…) to count them.
After this, these are merged to global maps (sevCount, causeCount..).
The (**nearby_accident > 20)** means high cluster zone is calculated using **#pragma omp reduction.**

Flowchart For Serial Implementation:



Flowchart for Parallel Implementation:



4.1. Serial Algorithm

```
BEGIN
STRUCT Accident(id, veh, inj, fat, near, lat, lon, date, time, sev, road, weather, cause,
traffic, lane)

FUNC trim(s) → s without spaces
FUNC extractHour(t) → int(before ":") IF valid ELSE -1

FUNC readData(f)
   OPEN f; IF fail RETURN []
   SKIP header; data=[]
   FOR each line
      tokens = trimmed CSV split
      IF len=15 THEN ADD Accident(tokens)
   RETURN data

FUNC analyze(L)
   INIT totals=0s, maps={}, hourly[24]=0
   FOR a IN L
      UPDATE totals, maps[a fields]; h=extractHour(a.time); IF h≥0 THEN hourly[h]++
   PRINT stats, maps, "Peak hour", argmax(hourly), "Clusters", count(a.near>20)

MAIN
   PRINT student info
   data=readData("india_traffic_accidents.csv")
   IF data EMPTY PRINT "No data" ELSE analyze(data)
   LOOP
      ASK "File (or quit):"; READ f
      IF f="quit" BREAK
      d=readData(f); IF d EMPTY PRINT "No valid data" ELSE analyze(d)
   PRINT "Exit."
END
```

4.2. Parallel Algorithm (OpenMP)

```
BEGIN

FUNCTION readData(filename):
    OPEN CSV file
    SKIP header
    FOR each line in file:
        PARSE columns
        CREATE Accident record (extract hour, convert to numeric)
        APPEND to data list
    RETURN data list

FUNCTION analyze(accidents):
    n ← number of accidents
    INIT totals: total_inj, total_fat, total_veh, total_nearby = 0
    INIT hourly[24] = 0

    #pragma omp parallel
    {
        local_inj, local_fat, local_veh, local_nearby = 0
        local_hourly[24] = 0

        #pragma omp for schedule(static)
        FOR i in 0..n-1:
            ACCUMULATE local values (injuries, fatalities, etc.)
            INCREMENT local_hourly[hour]

        #pragma omp atomic
        total_inj += local_inj
        #pragma omp atomic
        total_fat += local_fat
        #pragma omp atomic
        total_veh += local_veh
```

```
        #pragma omp atomic
        total_nearby += local_nearby

        FOR h in 0..23:
            IF local_hourly[h] > 0 THEN
                #pragma omp atomic
                hourly[h] += local_hourly[h]
    }

    num_threads ← omp_get_max_threads()
    CREATE local maps per thread: severity, cause, traffic, weather

    #pragma omp parallel
    {
        tid ← omp_get_thread_num()
        #pragma omp for schedule(static)
        FOR i in 0..n-1:
            UPDATE local maps[tid] with accident string fields
    }

    MERGE all thread-local maps into global maps

    COMPUTE averages, peak hour
    PRINT statistics and distributions

    high_cluster = 0
    #pragma omp parallel for reduction(+ : high_cluster)
    FOR i in 0..n-1:
        IF nearby_accidents > 20:
            high_cluster++

    PRINT high_cluster result
END FUNCTION
```

```
MAIN:
  LOAD original dataset → readData()
  CALL analyze(accidents)
  LOOP:
    ASK user for new CSV filename
    IF filename == 'quit' THEN EXIT LOOP
    LOAD dataset
    CALL analyze(accidents)
  END LOOP

PRINT "Exiting program"

END
```

4.3. Implementation Details

Programming Language: **C++**

Platform: **Windows**

Compiler: **GCC (-fopenmp)**

Hardware Configuration: **CPU:** Dual-core (2 threads) minimum
Quad-core or higher is recommended

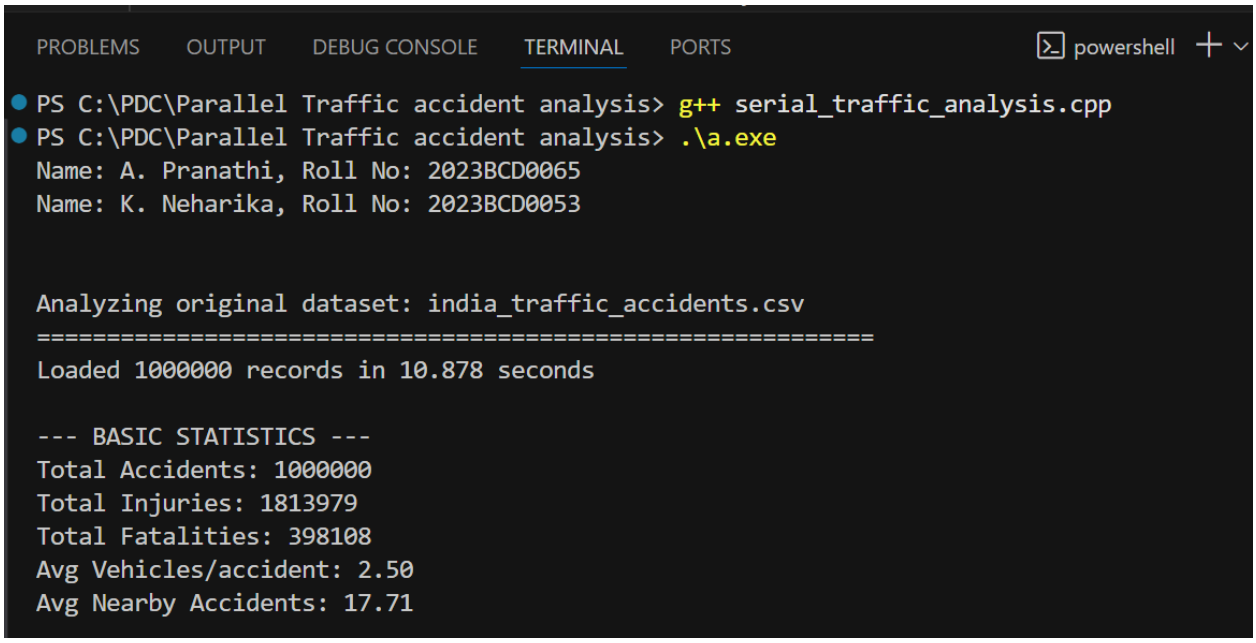
RAM: 2GB minimum
8GB or more is recommended

Storage: HDD (for small datasets)
SSD (for faster I/O on large datasets)

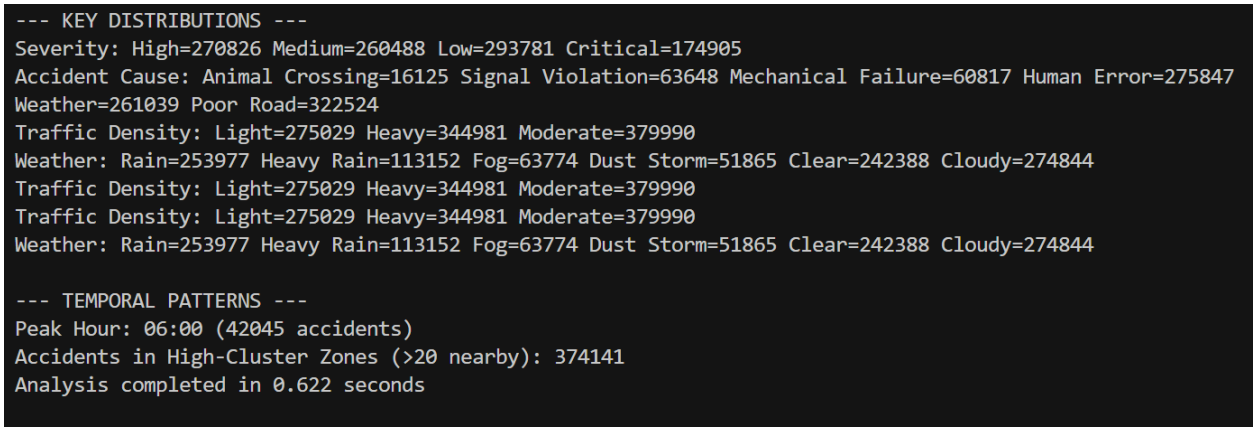
OS: 64 bit

5. Results and Analysis

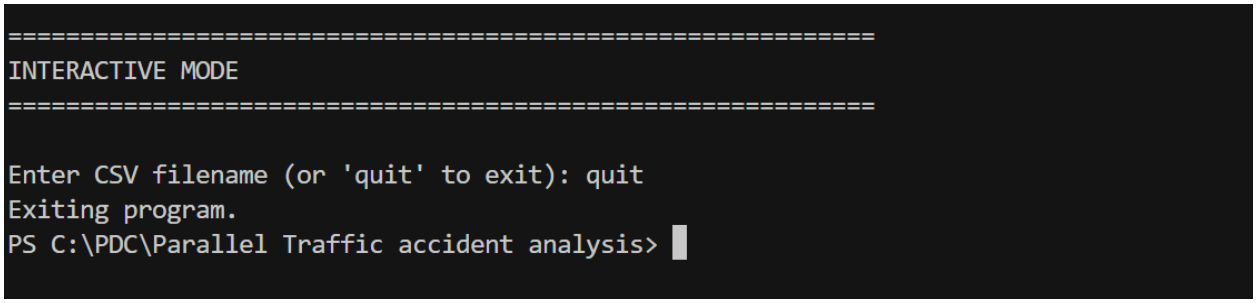
Output for Serial Implementation:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\PDC\Parallel Traffic accident analysis> g++ serial_traffic_analysis.cpp
PS C:\PDC\Parallel Traffic accident analysis> .\a.exe
Name: A. Pranathi, Roll No: 2023BCD0065
Name: K. Neharika, Roll No: 2023BCD0053


Analyzing original dataset: india_traffic_accidents.csv
============================================================
Loaded 1000000 records in 10.878 seconds

--- BASIC STATISTICS ---
Total Accidents: 1000000
Total Injuries: 1813979
Total Fatalities: 398108
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.71
```



```
--- KEY DISTRIBUTIONS ---
Severity: High=270826 Medium=260488 Low=293781 Critical=174905
Accident Cause: Animal Crossing=16125 Signal Violation=63648 Mechanical Failure=60817 Human Error=275847
Weather=261039 Poor Road=322524
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Weather: Rain=253977 Heavy Rain=113152 Fog=63774 Dust Storm=51865 Clear=242388 Cloudy=274844
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Weather: Rain=253977 Heavy Rain=113152 Fog=63774 Dust Storm=51865 Clear=242388 Cloudy=274844

--- TEMPORAL PATTERNS ---
Peak Hour: 06:00 (42045 accidents)
Accidents in High-Cluster Zones (>20 nearby): 374141
Analysis completed in 0.622 seconds
```



```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): quit
Exiting program.
PS C:\PDC\Parallel Traffic accident analysis>
```

Test case-1: (with 7000 records)

```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test1.csv

Loaded 7000 records in 0.110 seconds

--- BASIC STATISTICS ---
Total Accidents: 7000
Total Injuries: 12471
Total Fatalities: 2726
Avg Vehicles/accident: 2.52
Avg Nearby Accidents: 17.54
```

```
--- KEY DISTRIBUTIONS ---
Severity: Critical=1192 Low=2127 Medium=1796 High=1885
Accident Cause: Animal Crossing=92 Mechanical Failure=422 Weather=1850 Human Error=1948 Poor Road=2255 Si
gnal Violation=433
Traffic Density: Heavy=2367 Light=1945 Moderate=2688
Weather: Heavy Rain=809 Fog=447 Dust Storm=355 Clear=1674 Rain=1772 Cloudy=1943

--- TEMPORAL PATTERNS ---
Peak Hour: 03:00 (334 accidents)
Accidents in High-Cluster Zones (>20 nearby): 2578
Analysis completed in 0.050 seconds
```

Test case-2: (with 50000 records)

```
Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test2.csv

Loaded 50000 records in 0.560 seconds

--- BASIC STATISTICS ---
Total Accidents: 50000
Total Injuries: 90084
Total Fatalities: 19860
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.78
```

```
 --- KEY DISTRIBUTIONS ---
 Severity: Critical=8785 Medium=12907 High=13449 Low=14859
 Accident Cause: Animal Crossing=775 Signal Violation=3146 Mechanical Failure=3022 Human Error=13817 Weath
 er=13176 Poor Road=16064
 Traffic Density: Moderate=19032 Heavy=17289 Light=13679
 Weather: Rain=12514 Dust Storm=2585 Fog=3198 Cloudy=13668 Heavy Rain=5788 Clear=12247

 --- TEMPORAL PATTERNS ---
 Peak Hour: 17:00 (2154 accidents)
 Accidents in High-Cluster Zones (>20 nearby): 18742
 Analysis completed in 0.086 seconds
```

Test case-3: (with 100000 records)

```
Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test3.csv

Loaded 100000 records in 1.240 seconds

--- BASIC STATISTICS ---
Total Accidents: 100000
Total Injuries: 180678
Total Fatalities: 40081
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.68
```

```
--- KEY DISTRIBUTIONS ---
Severity: Low=29450 Critical=17414 Medium=26123 High=27013
Accident Cause: Animal Crossing=1604 Signal Violation=6206 Poor Road=32254 Human Error=27619 Weather=2619
2 Mechanical Failure=6125
Traffic Density: Moderate=37827 Heavy=34577 Light=27596
Weather: Heavy Rain=11284 Cloudy=27339 Rain=25270 Dust Storm=5224 Fog=6447 Clear=24436

--- TEMPORAL PATTERNS ---
Peak Hour: 23:00 (4284 accidents)
Accidents in High-Cluster Zones (>20 nearby): 37263
Analysis completed in 0.080 seconds
```

Output for Parallel Implementation:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   powershell

PS C:\PDC\Parallel Traffic accident analysis> g++ -fopenmp parallel_traffic_analysis.cpp -o parallel_traf
fic_analysis
PS C:\PDC\Parallel Traffic accident analysis> ./parallel_traffic_analysis
Name: A. Pranathi, Roll No: 2023BCD0065
Name: K. Neharika, Roll No: 2023BCD0053


Analyzing original dataset: india_traffic_accidents.csv
============================================================
Loaded 1000000 records in 10.281 seconds

--- BASIC STATISTICS ---
Total Accidents: 1000000
Total Injuries: 1813979
Total Fatalities: 398108
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.71
```

```
--- KEY DISTRIBUTIONS ---
Severity: Critical=174905 Low=293781 Medium=260488 High=270826
Accident Cause: Poor Road=322524 Weather=261039 Human Error=275847 Mechanical Failure=60817 Signal Violat
ion=63648 Animal Crossing=16125
Traffic Density: Moderate=379990 Heavy=344981 Light=275029
Weather: Cloudy=274844 Clear=242388 Dust Storm=51865 Fog=63774 Heavy Rain=113152 Rain=253977

--- TEMPORAL PATTERNS ---
Peak Hour: 06:00 (42045 accidents)
Accidents in High-Cluster Zones (>20 nearby): 374141
[Analysis completed in 0.13 seconds]
```

```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): quit
Exiting program.
PS C:\PDC\Parallel Traffic accident analysis>
```

Test case-1: (with 7000 records)

```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test1.csv

Loaded 7000 records in 0.096 seconds

--- BASIC STATISTICS ---
Total Accidents: 7000
Total Injuries: 12471
Total Fatalities: 2726
Avg Vehicles/accident: 2.52
Avg Nearby Accidents: 17.54
```

```
--- KEY DISTRIBUTIONS ---
Severity: High=1885 Medium=1796 Low=2127 Critical=1192
Accident Cause: Signal Violation=433 Poor Road=2255 Human Error=1948 Weather=1850 Mechanical Failure=422
Animal Crossing=92
Traffic Density: Moderate=2688 Light=1945 Heavy=2367
Weather: Cloudy=1943 Rain=1772 Clear=1674 Dust Storm=355 Fog=447 Heavy Rain=809

--- TEMPORAL PATTERNS ---
Peak Hour: 03:00 (334 accidents)
Accidents in High-Cluster Zones (>20 nearby): 2578
[Analysis completed in 0.05 seconds]
```

Test case-2: (with 50000 records)

```
Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test2.csv

Loaded 50000 records in 0.591 seconds

--- BASIC STATISTICS ---
Total Accidents: 50000
Total Injuries: 90084
Total Fatalities: 19860
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.78
```

```
--- KEY DISTRIBUTIONS ---
Severity: Low=14859 High=13449 Medium=12907 Critical=8785
Accident Cause: Poor Road=16064 Weather=13176 Human Error=13817 Mechanical Failure=3022 Signal Violation=
3146 Animal Crossing=775
Traffic Density: Light=13679 Heavy=17289 Moderate=19032
Weather: Clear=12247 Heavy Rain=5788 Cloudy=13668 Fog=3198 Dust Storm=2585 Rain=12514

--- TEMPORAL PATTERNS ---
Peak Hour: 17:00 (2154 accidents)
Accidents in High-Cluster Zones (>20 nearby): 18742
[Analysis completed in 0.05 seconds]
```

Test case-3: (with 100000 records)

```
Enter CSV filename (or 'quit' to exit): C:\PDC\Parallel Traffic accident analysis\test data\test3.csv

Loaded 100000 records in 1.102 seconds

--- BASIC STATISTICS ---
Total Accidents: 100000
Total Injuries: 180678
Total Fatalities: 40081
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.68
```

```
--- KEY DISTRIBUTIONS ---
Severity: High=27013 Medium=26123 Critical=17414 Low=29450
Accident Cause: Mechanical Failure=6125 Weather=26192 Human Error=27619 Poor Road=32254 Signal Violation=
6206 Animal Crossing=1604
Traffic Density: Light=27596 Heavy=34577 Moderate=37827
Weather: Clear=24436 Fog=6447 Dust Storm=5224 Rain=25270 Cloudy=27339 Heavy Rain=11284

--- TEMPORAL PATTERNS ---
Peak Hour: 23:00 (4284 accidents)
Accidents in High-Cluster Zones (>20 nearby): 37263
[Analysis completed in 0.05 seconds]
```

Performance Comparison Table:

Dataset	Serial Time	Parallel Time	SpeedUp
Original dataset	0.622 sec	0.13 sec	4.7846x
Test case-1	0.050 sec	0.05 sec	1x
Test case-2	0.086 sec	0.05 sec	1.72x
Test case-3	0.080 sec	0.05 sec	1.6x

Time Complexity:

Serial Version:
Each accident is read and processed independently → O(N)
N = number of accident records.

Parallel Version:
In this parallel version,computation is divided among threads →O(N/p)

Batch processing(M datasets):

Total complexity ~~ $O(M \times N / p)$

Scalability:

Using OpenMP makes the program much faster because it divides the work along multiple CPU cores parallely at the same time.

So, if the dataset is large, it uses more threads to compute. This makes our program more efficient and runs faster.
Because each accident data record is analyzed separately by each thread. These threads handle different parts of data without waiting for each other. So, better scalability can be seen for larger datasets.

Here, there is a small delay from creating and managing threads or reading files. But this delay is very small compared to the total work.

6. Discussion and Observations

This implementation tells about the difference of execution time between serial and parallel. When using OpenMp parallelization . If you take a dataset the serial execution time was 0.622 seconds. While in parallel, it completed in 0.13 seconds,acquiring a speedup of 4.7846x.
We can observe even better speedup for large datasets.

This shows how the work will be completed earlier by parallel version compared to serial version.

Limitations:

The parallel version is better than the serial version,but there are some limitations.The file (CSV) input/output remains sequential operation,as reading and processing from large datasets cannot be parallelized easily .

Load imbalance may occur if data is not distributed uniformly ,meaning some threads finish early while others are still processing the data which reduces the overall parallel efficiency .

7. Conclusion and Future Scope

Work done:

We developed a C++ program to analyze traffic accident data from CSV files. This program computes the key statistics like injuries, fatalities, severity distributions, peak accident hours and high cluster zones.

We improved this implementation by using OpenMP to parallelize adding up numbers, counting text based categories using separate maps for each thread.
These changes make our program run much faster compared to the serial version.

Improvements:

- Our serial program analyzed the data containing (10 lakh records) in 0.622 sec. And the parallel program takes 0.13 sec.
  This tells us the parallel version is almost 5 times better than the serial version.
- When counting the categorical data, we used separate maps for each thread. This avoided conflicts.
- We also used **atomic** operations for calculating totals shared between the threads. It updated correctly without slowing things down.
- Even with all these, the program still processes the data in linear time O(N). And it also scales well for large datasets.

Potential Extensions:

MPI:

For large datasets, the program can be run on multiple computers using MPI.
Each computer can work on its own portion of data without depending on each other.
So, after processing from all computers, we can combine them to get the total sum or count.

Finally, this approach is best for too large datasets (like world-wide accident records) that are difficult to fit in one computer.

CUDA (GPU acceleration):

In our program, tasks that are done independently like counting, filtering and calculating hourly totals, can run on a **GPU** for faster processing.

The operations like summing values, making histograms for hours, severity etc., can be performed efficiently using libraries like **Thrust or cuDF.**

This approach is best for large datasets with billions of data, where **GPU's** fast memory and multiple cores can be fully used.

8. References

1. Freeway accident analysis using second order statistics by LSU Scholarly Repository https://repository.lsu.edu/cgi/viewcontent.cgi?article=1318&context=gradschool_theses

2. Analysis of traffic accident characteristics and recovery strategy of urban road network. (Article) https://jeas.springeropen.com/articles/10.1186/s44147-025-00657-1

3. Analysis of traffic accidents based on Spark and casual inference https://www.ewadirect.com/proceedings/ace/article/view/14882/pdf

Appendix

A. Serial Code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
```

```cpp
#include <unordered_map>
#include <algorithm>
#include <iomanip>
#include <chrono>
using namespace std;

struct Accident
{
  int id, vehicles, injuries, fatalities, nearby_accidents;
  double lat, lon;
  string date, time, severity, road_condition, weather;
  string accident_cause, traffic_density, lane_utilization;
};

string trim(const string &str)
{
  size_t start = str.find_first_not_of(" \t"), end =
str.find_last_not_of(" \t");
  return (start == string::npos) ? "" : str.substr(start, end -
start + 1);
}

vector<Accident> readData(const string &filename)
{
  vector<Accident> data;
  ifstream file(filename);
  if (!file.is_open())
  {
    cerr << "Error opening file '" << filename << "'\n";
    return data;
  }
  string line;
  getline(file, line); // skip header
  int lineNum = 1;
  while (getline(file, line))
  {
    lineNum++;
    if (line.empty())
      continue;
    stringstream ss(line);
```

```
    string token;
    vector<string> tokens;
    while (getline(ss, token, ','))
      tokens.push_back(trim(token));
    if (tokens.size() != 15)
    {
      cerr << "Skipping line " << lineNum << " (expected 15
cols)\n";
      continue;
    }
    try
    {
      Accident a;
      a.id = stoi(tokens[0]);
      a.date = tokens[1];
      a.time = tokens[2];
      a.lat = stod(tokens[3]);
      a.lon = stod(tokens[4]);
      a.severity = tokens[5];
      a.road_condition = tokens[6];
      a.weather = tokens[7];
      a.vehicles = stoi(tokens[8]);
      a.injuries = stoi(tokens[9]);
      a.fatalities = stoi(tokens[10]);
      a.accident_cause = tokens[11];
      a.traffic_density = tokens[12];
      a.lane_utilization = tokens[13];
      a.nearby_accidents = stoi(tokens[14]);
      data.push_back(a);
    }
    catch (...)
    {
      cerr << "Parse error on line " << lineNum << "\n";
    }
  }
  return data;
}

int extractHour(const string &t)
{
```

```
  if (t.empty())
    return -1;

  size_t pos = t.find(':');
  if (pos == string::npos)
    return -1;

  try
  {
    int hour = stoi(t.substr(0, pos));
    if (hour >= 0 && hour < 24)
      return hour;
    else
      return -1;
  }
  catch (...)
  {
    return -1;
  }
}

template <typename K, typename V>
void printMap(const unordered_map<K, V> &map, const string &title)
{
  cout << title << ": ";
  for (auto &p : map)
    cout << p.first << "=" << p.second << " ";
  cout << "\n";
}

// Analysis Function
void analyze(const vector<Accident> &accidents)
{
  long long total_inj = 0, total_fat = 0, total_veh = 0,
total_nearby = 0;
  unordered_map<string, int> sevCount, causeCount, trafficCount,
weatherCount;
  vector<int> hourly(24, 0);

  for (auto &a : accidents)
```

```
  {
    total_inj += a.injuries;
    total_fat += a.fatalities;
    total_veh += a.vehicles;
    total_nearby += a.nearby_accidents;
    sevCount[a.severity]++;
    causeCount[a.accident_cause]++;
    trafficCount[a.traffic_density]++;
    weatherCount[a.weather]++;
    int h = extractHour(a.time);
    if (h >= 0 && h < 24)
      hourly[h]++;
  }

  double avg_nearby = static_cast<double>(total_nearby) /
accidents.size();
  cout << "\n--- BASIC STATISTICS ---\n";
  cout << "Total Accidents: " << accidents.size() << "\nTotal
Injuries: " << total_inj
       << "\nTotal Fatalities: " << total_fat << "\nAvg
Vehicles/accident: " << fixed << setprecision(2)
       << static_cast<double>(total_veh) / accidents.size() <<
"\nAvg Nearby Accidents: " << avg_nearby << "\n";

  cout << "\n--- KEY DISTRIBUTIONS ---\n";
  printMap(sevCount, "Severity");
  printMap(causeCount, "Accident Cause");
  printMap(trafficCount, "Traffic Density");
  printMap(weatherCount, "Weather");

  int peak_hour = max_element(hourly.begin(), hourly.end()) -
hourly.begin();
  cout << "\n--- TEMPORAL PATTERNS ---\n";
  cout << "Peak Hour: " << setfill('0') << setw(2) << peak_hour
       << ":00 (" << hourly[peak_hour] << " accidents)\n";

  int high_cluster = count_if(accidents.begin(), accidents.end(),
[](const Accident &a)
                              { return a.nearby_accidents > 20; });
```

```
  cout << "Accidents in High-Cluster Zones (>20 nearby): " <<
high_cluster << "\n";
}

// Main function
int main()
{
  cout << "Name: A. Pranathi, Roll No: 2023BCD0065\n";
  cout << "Name: K. Neharika, Roll No: 2023BCD0053\n"
       << endl;
  string originalFile = "india_traffic_accidents.csv";
  cout << "\nAnalyzing original dataset: " << originalFile << "\n"
       << string(60, '=') << "\n";

  auto start = chrono::high_resolution_clock::now();
  auto accidents = readData(originalFile);
  auto end = chrono::high_resolution_clock::now();

  if (accidents.empty())
    cout << "Warning: Dataset not found or empty.\n";
  else
  {
    cout << "Loaded " << accidents.size() << " records in " << fixed
<< setprecision(3)
         << chrono::duration<double>(end - start).count() << "
seconds\n";

    auto analyze_start = chrono::high_resolution_clock::now();
    analyze(accidents);
    auto analyze_end = chrono::high_resolution_clock::now();

    cout << "Analysis completed in " << fixed << setprecision(3)
         << chrono::duration<double>(analyze_end -
analyze_start).count() << " seconds\n";
  }

  cout << "\n"
       << string(60, '=') << "\nINTERACTIVE MODE\n"
       << string(60, '=') << "\n";
  string userFile;
```

```
  while (true)
  {
    cout << "\nEnter CSV filename (or 'quit' to exit): ";
    getline(cin, userFile);
    userFile = trim(userFile);
    if (userFile == "quit" || userFile.empty())
      break;

    auto start = chrono::high_resolution_clock::now();
    auto accidents = readData(userFile);
    auto end = chrono::high_resolution_clock::now();

    if (accidents.empty())
    {
      cout << "No valid data found in '" << userFile << "'.\n";
      continue;
    }

    cout << "\nLoaded " << accidents.size() << " records in " <<
fixed << setprecision(3)
         << chrono::duration<double>(end - start).count() << "
seconds\n";

    auto analyze_start = chrono::high_resolution_clock::now();
    analyze(accidents);
    auto analyze_end = chrono::high_resolution_clock::now();

    cout << "Analysis completed in " << fixed << setprecision(3)
         << chrono::duration<double>(analyze_end -
analyze_start).count() << " seconds\n";
  }

  cout << "Exiting program.\n";
  return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\PDC\Parallel Traffic accident analysis> g++ serial_traffic_analysis.cpp
PS C:\PDC\Parallel Traffic accident analysis> .\a.exe
  Name: A. Pranathi, Roll No: 2023BCD0065
  Name: K. Neharika, Roll No: 2023BCD0053


  Analyzing original dataset: india_traffic_accidents.csv
  ============================================================
  Loaded 1000000 records in 10.878 seconds

  --- BASIC STATISTICS ---
  Total Accidents: 1000000
  Total Injuries: 1813979
  Total Fatalities: 398108
  Avg Vehicles/accident: 2.50
  Avg Nearby Accidents: 17.71
```

```
--- KEY DISTRIBUTIONS ---
Severity: High=270826 Medium=260488 Low=293781 Critical=174905
Accident Cause: Animal Crossing=16125 Signal Violation=63648 Mechanical Failure=60817 Human Error=275847
Weather=261039 Poor Road=322524
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Weather: Rain=253977 Heavy Rain=113152 Fog=63774 Dust Storm=51865 Clear=242388 Cloudy=274844
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Traffic Density: Light=275029 Heavy=344981 Moderate=379990
Weather: Rain=253977 Heavy Rain=113152 Fog=63774 Dust Storm=51865 Clear=242388 Cloudy=274844

--- TEMPORAL PATTERNS ---
Peak Hour: 06:00 (42045 accidents)
Accidents in High-Cluster Zones (>20 nearby): 374141
Analysis completed in 0.622 seconds
```

```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): quit
Exiting program.
PS C:\PDC\Parallel Traffic accident analysis>
```

B. Parallel Code:

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
```

```
#include <sstream>
#include <unordered_map>
#include <algorithm>
#include <iomanip>
#include <chrono>
#include <omp.h>
using namespace std;

struct Accident
{
  int id, vehicles, injuries, fatalities, nearby_accidents, hour;
  double lat, lon;
  string severity, road_condition, weather;
  string accident_cause, traffic_density, lane_utilization;
  // date/time not stored as strings to reduce memory; hour
precomputed
};

string trim(const string &str)
{
  size_t start = str.find_first_not_of(" \t"), end =
str.find_last_not_of(" \t");
  return (start == string::npos) ? "" : str.substr(start, end -
start + 1);
}

int extractHour(const string &t)
{
  if (t.empty())
    return -1;
  size_t pos = t.find(':');
  if (pos == string::npos)
    return -1;
  try
  {
    int hour = stoi(t.substr(0, pos));
    return (hour >= 0 && hour < 24) ? hour : -1;
  }
  catch (...)
  {
```

```
    return -1;
  }
}

vector<Accident> readData(const string &filename)
{
  vector<Accident> data;
  ifstream file(filename);
  if (!file.is_open())
  {
    cerr << "Error opening file '" << filename << "'\n";
    return data;
  }
  string line;
  getline(file, line); // skip header
  int lineNum = 1;
  while (getline(file, line))
  {
    lineNum++;
    if (line.empty())
      continue;
    stringstream ss(line);
    string token;
    vector<string> tokens;
    while (getline(ss, token, ','))
      tokens.push_back(trim(token));
    if (tokens.size() != 15)
    {
      cerr << "Skipping line " << lineNum << " (expected 15
cols)\n";
      continue;
    }
    try
    {
      Accident a;
      a.id = stoi(tokens[0]);
      // Precompute hour from time string (tokens[2])
      a.hour = extractHour(tokens[2]);
      a.lat = stod(tokens[3]);
      a.lon = stod(tokens[4]);
```

```
      a.severity = tokens[5];
      a.road_condition = tokens[6];
      a.weather = tokens[7];
      a.vehicles = stoi(tokens[8]);
      a.injuries = stoi(tokens[9]);
      a.fatalities = stoi(tokens[10]);
      a.accident_cause = tokens[11];
      a.traffic_density = tokens[12];
      a.lane_utilization = tokens[13];
      a.nearby_accidents = stoi(tokens[14]);
      data.push_back(a);
    }
    catch (...)
    {
      cerr << "Parse error on line " << lineNum << "\n";
    }
  }
  return data;
}

template <typename K, typename V>
void printMap(const unordered_map<K, V> &map, const string &title)
{
  cout << title << ": ";
  for (auto &p : map)
    cout << p.first << "=" << p.second << " ";
  cout << "\n";
}

void analyze(const vector<Accident> &accidents)
{
  int n = accidents.size();
  if (n == 0)
    return;

  // Global accumulators
  long long total_inj = 0, total_fat = 0, total_veh = 0,
total_nearby = 0;
  vector<int> hourly(24, 0);
```

```
// Parallel numerical + hourly aggregation
#pragma omp parallel
  {
    long long local_inj = 0, local_fat = 0, local_veh = 0,
local_nearby = 0;
    vector<int> local_hourly(24, 0);

#pragma omp for schedule(static)
    for (int i = 0; i < n; i++)
    {
      local_inj += accidents[i].injuries;
      local_fat += accidents[i].fatalities;
      local_veh += accidents[i].vehicles;
      local_nearby += accidents[i].nearby_accidents;
      int h = accidents[i].hour;
      if (h >= 0 && h < 24)
        local_hourly[h]++;
    }

// Merge local results using atomic updates
#pragma omp atomic
    total_inj += local_inj;
#pragma omp atomic
    total_fat += local_fat;
#pragma omp atomic
    total_veh += local_veh;
#pragma omp atomic
    total_nearby += local_nearby;

    for (int h = 0; h < 24; ++h)
    {
      if (local_hourly[h] != 0)
      {
#pragma omp atomic
        hourly[h] += local_hourly[h];
      }
    }
  }

  // Parallel string aggregation with thread-local maps
```

```
  int num_threads = omp_get_max_threads();
  vector<unordered_map<string, int>> local_sev(num_threads);
  vector<unordered_map<string, int>> local_cause(num_threads);
  vector<unordered_map<string, int>> local_traffic(num_threads);
  vector<unordered_map<string, int>> local_weather(num_threads);

#pragma omp parallel
  {
    int tid = omp_get_thread_num();
#pragma omp for schedule(static)
    for (int i = 0; i < n; i++)
    {
      local_sev[tid][accidents[i].severity]++;
      local_cause[tid][accidents[i].accident_cause]++;
      local_traffic[tid][accidents[i].traffic_density]++;
      local_weather[tid][accidents[i].weather]++;
    }
  }

  // Merge thread-local maps
  unordered_map<string, int> sevCount, causeCount, trafficCount,
weatherCount;
  for (int t = 0; t < num_threads; t++)
  {
    for (auto &p : local_sev[t])
      sevCount[p.first] += p.second;
    for (auto &p : local_cause[t])
      causeCount[p.first] += p.second;
    for (auto &p : local_traffic[t])
      trafficCount[p.first] += p.second;
    for (auto &p : local_weather[t])
      weatherCount[p.first] += p.second;
  }

  // Output
  double avg_nearby = static_cast<double>(total_nearby) / n;
  cout << "\n--- BASIC STATISTICS ---\n";
  cout << "Total Accidents: " << n << "\nTotal Injuries: " <<
total_inj
```

```
       << "\nTotal Fatalities: " << total_fat << "\nAvg
Vehicles/accident: " << fixed << setprecision(2)
       << static_cast<double>(total_veh) / n << "\nAvg Nearby
Accidents: " << avg_nearby << "\n";

  cout << "\n--- KEY DISTRIBUTIONS ---\n";
  printMap(sevCount, "Severity");
  printMap(causeCount, "Accident Cause");
  printMap(trafficCount, "Traffic Density");
  printMap(weatherCount, "Weather");

  int peak_hour = max_element(hourly.begin(), hourly.end()) -
hourly.begin();
  cout << "\n--- TEMPORAL PATTERNS ---\n";
  cout << "Peak Hour: " << setfill('0') << setw(2) << peak_hour
       << ":00 (" << hourly[peak_hour] << " accidents)\n";

  int high_cluster = 0;
#pragma omp parallel for reduction(+ : high_cluster)
  for (int i = 0; i < n; i++)
    if (accidents[i].nearby_accidents > 20)
      high_cluster++;

  cout << "Accidents in High-Cluster Zones (>20 nearby): " <<
high_cluster << "\n";
}

int main()
{
  cout << "Name: A. Pranathi, Roll No: 2023BCD0065\n";
  cout << "Name: K. Neharika, Roll No: 2023BCD0053\n\n";

  string originalFile = "india_traffic_accidents.csv";
  cout << "\nAnalyzing original dataset: " << originalFile << "\n"
       << string(60, '=') << "\n";

  auto start_load = chrono::high_resolution_clock::now();
  auto accidents = readData(originalFile);
  auto end_load = chrono::high_resolution_clock::now();
```

```
  if (accidents.empty())
  {
    cout << "Warning: Dataset not found or empty.\n";
  }
  else
  {
    cout << "Loaded " << accidents.size() << " records in " << fixed
<< setprecision(3)
         << chrono::duration<double>(end_load - start_load).count()
<< " seconds\n";

    auto start_analysis = chrono::high_resolution_clock::now();
    analyze(accidents);
    auto end_analysis = chrono::high_resolution_clock::now();

    cout << "[Analysis completed in "
         << chrono::duration<double>(end_analysis -
start_analysis).count()
         << " seconds]\n";
  }

  cout << "\n"
       << string(60, '=') << "\nINTERACTIVE MODE\n"
       << string(60, '=') << "\n";
  string userFile;
  while (true)
  {
    cout << "\nEnter CSV filename (or 'quit' to exit): ";
    getline(cin, userFile);
    userFile = trim(userFile);
    if (userFile == "quit" || userFile.empty())
      break;

    auto start_load = chrono::high_resolution_clock::now();
    auto accidents = readData(userFile);
    auto end_load = chrono::high_resolution_clock::now();

    if (accidents.empty())
    {
      cout << "No valid data found in '" << userFile << "'.\n";
```

```
        continue;
    }

    cout << "\nLoaded " << accidents.size() << " records in " <<
fixed << setprecision(3)
         << chrono::duration<double>(end_load - start_load).count()
<< " seconds\n";

    auto start_analysis = chrono::high_resolution_clock::now();
    analyze(accidents);
    auto end_analysis = chrono::high_resolution_clock::now();

    cout << "[Analysis completed in "
         << chrono::duration<double>(end_analysis -
start_analysis).count()
         << " seconds]\n";
  }

  cout << "Exiting program.\n";
  return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell

PS C:\PDC\Parallel Traffic accident analysis> g++ -fopenmp parallel_traffic_analysis.cpp -o parallel_traf
fic_analysis
PS C:\PDC\Parallel Traffic accident analysis> ./parallel_traffic_analysis
Name: A. Pranathi, Roll No: 2023BCD0065
Name: K. Neharika, Roll No: 2023BCD0053


Analyzing original dataset: india_traffic_accidents.csv
============================================================
Loaded 1000000 records in 10.281 seconds

--- BASIC STATISTICS ---
Total Accidents: 1000000
Total Injuries: 1813979
Total Fatalities: 398108
Avg Vehicles/accident: 2.50
Avg Nearby Accidents: 17.71
```

```
--- KEY DISTRIBUTIONS ---
Severity: Critical=174905 Low=293781 Medium=260488 High=270826
Accident Cause: Poor Road=322524 Weather=261039 Human Error=275847 Mechanical Failure=60817 Signal Violat
ion=63648 Animal Crossing=16125
Traffic Density: Moderate=379990 Heavy=344981 Light=275029
Weather: Cloudy=274844 Clear=242388 Dust Storm=51865 Fog=63774 Heavy Rain=113152 Rain=253977

--- TEMPORAL PATTERNS ---
Peak Hour: 06:00 (42045 accidents)
Accidents in High-Cluster Zones (>20 nearby): 374141
[Analysis completed in 0.13 seconds]
```

```
============================================================
INTERACTIVE MODE
============================================================

Enter CSV filename (or 'quit' to exit): quit
Exiting program.
PS C:\PDC\Parallel Traffic accident analysis>
```

C. Time Comparison Script:

Serial Analysis Time - 0.622 sec
Parallel Analysis Time - 0.13 sec

Speedup = Serial Time/Parallel Time
= 0.622/0.13
= **4.78x**

If we take 8 threads,
Efficiency = Speedup/No.of threads
= 4.78/8
= **59.8%** (nearly)