

Parallel Computing

Exercise 4 (May, 2026)

Alois Zoitl

alois.zoitl@jku.at

Adriano Marques Garcia

adriano.garcia@jku.at

The result is to be submitted by the deadline stated above via the Moodle interface. The submitted result is a .zip or .tgz file which contains:

- **a single PDF (.pdf) file** with:
 - a cover page with the title of the course, your name(s), Matrikelnummer(s), and email address (es);
 - answers to the tasks provided in the requirements below
- **Output generated by the program.**
- The actual **source code** of your solutions.

If you work in a team, the quality of the solution is expected to be higher, and grading will take this into account.

Tips:

- Understand the parallel codes explored in the lecture.
- See the code examples provided.
- Check the C++ documentation: <https://en.cppreference.com/>
- In case of compilation errors, try to understand what the compiler is throwing because the compilers are usually helpful.
- Understand the sequential program provided.

Installation requirements

It is recommended to compile and run the code in a Linux environment, such as a machine running Ubuntu.

Windows users can easily set up a deployment environment, such as [using WSL and Visual Studio Code](#)

Using Linux is important for compilers support and performance.

The code examples and assignment support C++-17. The recommended compiler to use is G++ 9 or newer. One can check if G++ is installed and which version with the command:

```
g++ --version
```

For instance, this is the output when G++ 9.4 is installed:

```
g++ (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
```

In case it is not installed, the package build-essential should be installed (e.g., in Ubuntu):

```
sudo apt install build-essential
```

The parallel algorithms also require Threading Building Blocks, which can be installed in Linux with:

```
sudo apt install libtbb-dev
```

Goal

The assignment is to introduce parallelism to the sequential application to significantly accelerate the execution (reducing the execution time and increasing the throughput).

The application folder contains:

- Sequential CPP code version: Code to be parallelized
- Makefile: Compilation helper: the command *make* should compile the application, which can then be run with the command: *./trafficCongestionAlert_sequential*
- Input: is a folder with a CSV file used as input and a CSV file with previous values to be used as a baseline.

The program reads a CSV file where each line is an entry from a sensor that detects the speed of vehicles passing by it. Each line in the CSV has values such as a *sensor_id*, a timestamp, and measured speed. The application processes the lines sequentially, with operations of loading, filtering, aggregating, and generating alerts according to predefined rules. A correct execution output is:

```
./trafficCongestionAlert_sequential
```

```
[...list of 63 Alerts...]
```

```
Total alerts generated (slowdown + congestion): 63
```

```
Execution Time(seconds): 30.34
```

```
Throughput (records/s): 32962.75
```

Please note that the focus is on supporting parallelism instead of optimizing business logic code or adding optimizations at compile time.

Requirements

- (1) The final program should execute in parallel.
- (2) The final program should provide a correct output in terms of alerts and total alert counting, comparable to the sequential implementation.
- (3) provide significant performance improvements compared to the sequential implementation.
- (4) A performance analysis with at least 3 repetitions of each execution.
- (5) A results discussion explaining (i) the design adopted for the parallel solution, (ii) how the implementation works, (iii) the rationale for the performance results, and (iv) a discussion on what the expected advantages and limitations of the parallel implementation are.