# Distributed & Parallel Systems in Truck Platooning

Namık Mert Tunçbilek
Matric No : 7213712
*Embedded Systems Engineering*
*Fachhochschule Dortmund*
Dortmund, Germany
namk.tuncbilek001@stud.fh-dortmund.de
Contribution Percentage: 25

Nijat Dashdamirov
Matric No : 7213892
*Embedded Systems Engineering*
*Fachhochschule Dortmund*
Dortmund, Germany
nijat.dashdamirov002@stud.fh-dortmund.de
Contribution Percentage: 25

Omar Mammadov
Matric No : 7213766
*Embedded Systems Engineering*
*Fachhochschule Dortmund*
Dortmund, Germany
omar.mammadov001@stud.fh-dortmund.de
Contribution Percentage: 25

Vignesh Arumugam
Matric No : 7213710
*Embedded Systems Engineering*
*Fachhochschule Dortmund*
Dortmund, Germany
vignesh.arumugam002@stud.fh-dortmund.de
Contribution Percentage: 25

***Abstract*—To explore and understand the need of distributed and parallel systems, a real time and challenging system development of Truck Platooning technology is discussed in this paper. Also, it deals with the design and implementation of parallel programming using multi-threading in CPP and tried GPU programming with CUDA. Choosing the appropriate hardware for the system with the evidence of scheduling are carried out in this paper.**

***Keywords—Platooning, Distributed, Parallel, GPU programming, Multi-core, CUDA, Multi-threading***

## I. INTRODUCTION

Transportation is vital to society and economy, and street cargo transportation accounts for about 60% of all surface cargo transportation. The request for street cargo transport is anticipated to extend within the coming a long time. As appeared within the American Trucking Association's 2015 report, the trucking industry comprises about 80% of a 1.33 trillion-dollar shipping and coordination's industry within the US. In any case, bounty of fuel utilization and nursery gas outflow have been created. For case, street transport speaks to around 27% of the vitality utilization of the European Union. Furthermore, indicated that vehicles account for 20% of the entire carbon emanation of which a quarter comes from overwhelming obligation vehicles (HDVs). Subsequently, the natural impacts amid the method of transport got to be diminished direly. In expansion, the fetched of fuel has an expansive share of add up to transportation costs. Fuel fetched spoken to about 30% of the life cycle taken a toll of owning and working a truck. Additionally, agreeing to the American Transportation Investigate Institute's (ATRI) later report, fuel is respected as the moment biggest taken a toll, where the most elevated is faculty cost. With a huge sum of HDVs and the expanding request for street cargo, it can be anticipated that indeed small advances in fuel efficiency can decipher into significant taken a toll diminishment [1]. And it is additionally advantageous to realize the objective of natural assurance due to less deplete gas. As a result, it is of awesome advantage to move forward fuel economy, and how to decrease fuel utilization amid traveling has turned into a prevalent theme in later a long time. Luckily, the improvements of shrewdly transportation frameworks (ITSs) have empowered strategies to improve the vitality productivity of transportation systems. A promising approach to managing with that issue is to decrease the crevice between vehicles on the street, which is ordinarily called truck companies. Truck companies, moreover, known as caravans, are a set of vehicles shaping a street prepare by traveling closely in single record to encounter diminished discuss drag. This may altogether diminish fuel utilization since around one-fourth of the fuel utilization is important to streamlined drag. As a result, fuel economy can be progressed, and natural invitingness can be accomplished due to less nursery gas emanation in a unit. Separated from fuel reserve funds, truck units can contribute to an increment of street capacity and can ease activity blockage by a littler hole between vehicles. In later a long time, with the advancement of independent driving innovation, vehicles are prepared with a few sensors that empower them to watch their environment and choose in genuine time what activity ought to be taken, which are called "autonomous vehicles" or "driverless vehicles." Driverless vehicles can arrange their way when driving, and they can travel in a company with littler interims to diminish fuel utilization [2]. Moreover, when driving naturally in a company, it is conceivable to diminish the hazard of rear-end collisions and to move forward activity security. With incredible points of interest said over, vehicle companies have pulled in the consideration from numerous governments and inquire about teach. As a result, a few ventures related to companies were proposed. The primary thinks about on truck computerization were "Chauffeur" inside the EU venture T-TAP from the mid-1990s to the starting of 2000. Amid the extend, Cap and Fritz conducted an explore with two trucks coupled by an "Electronic Tow Bar" to measure the fuel reserve funds. A while later, the California Way program begun it inquire about on heavy truck platooning. Within the Way program, all vehicles were completely mechanized, counting the pioneer. For case, in 2004, the program performed a fuel utilization test with two pair trucks connected by an electronic control framework for diverse spaces.

This paper [3] really speaks clearly about the important terms in the platooning, and it have the definitions of String Stability, Stability Margin, and Coherence behaviour. Achieving all these definitions are very important for the platooning system.

## II. Analysis

### A. *Interaction and Communication*

Vehicle-to-vehicle (V2V) communication is a technology that allows vehicles to talk with one another via dedicated short-range communication (DSRC). The V2V communication system's primary function is to transmit chosen messages from one platoon vehicle's local CAN (Controller–area network) to the CAN of another platoon vehicle. The DSRC/IEEE 802.11p WAVE (Wireless Access in Vehicular Environment) protocol, which operates at 5.9 GHz for wireless communication, is used by all cars in the platoon to communicate with one another. CAN is a wired communication bus standard that allows microcontrollers and gadgets in automobiles to connect with one another. The V2V node serves as a wireless gateway to the other cars' CAN bus. This permits vehicles in the platoon to share local vehicle signal and sensor data. The platoon's control algorithms make use of the shared signals. Following signals are being used:

- Front trajectory
- Rear trajectory
- Current Rear trajectory
- Relative distance (longitudinal/lateral)
- Velocity
- Yaw rate
- Steering angle
- Kingpin angle
- Camera/Radar fusion data
- In-Vehicle Network chassis signals
- User Input to HMI

MAP, LIDAR, CAMERA, IMU and GPS sensors are used to obtain these signals.

The Dedicated Short Range Communications (DSRC) spectrum band was assigned to the IEEE 802.11p WAVE standardization process. Wi-Fi-based V2X communication, unlike cellular networks, lacks a pre-installed infrastructure and practically complete spatial coverage. It is totally distributed and so does not require a network infrastructure for coordination. When compared to an indirect transmission via infrastructure, data is transmitted directly among surrounding vehicles with a very short delay. Network administration is kept to a bare minimum, allowing for real-time data sharing between cars without the need for cumbersome signalling procedures [4].

Sensors like as radar and cameras are used to detect vehicles and lanes ahead and to execute autonomous driving by controlling the vehicle longitudinally and laterally. Because the car cannot receive accurate GPS signals in some settings, such as while traveling through tunnels, the Following Vehicle's autonomous driving algorithm does not rely on GPS. The following vehicle calculates the steering angle based on a geometrical principle using the preceding vehicle's relative longitudinal and lateral distance. Off-tracking is a significant contributor to the platoon's lateral dynamics being unstable. To avoid off-tracking, the following vehicle must use the tractor's trajectory rather than the trailer of the preceding vehicle as its goal path. The Ffollowing Vvehicle, on the other hand, cannot just use the camera to determine the positioning of the preceding vehicle's tractor. The target path is generated via V2V communication relying on the trajectory of the previous vehicle. To perceive the centre point of the previous vehicle's bumper, a mono camera and a radar are positioned upon the dashboard and front bumper, respectively. Dual antennae are inserted within the left and right side-mirrors of the V2V module to reduce the region of communication blind spot [5].

The Leading Vehicle controller creates its driven trajectory and transmits it to the Following Vehicle via V2V communication. Using the Leading Vehicle's trajectory, the Following Vehicle performs path planning (calculates its own target path to follow). Finally, the Following Vehicle implements path tracking control to follow the target path. In the same way, target paths are created between any adjacent Following Vehicles, so in essence, all Following Vehicles can follow the trajectory of the Leading Vehicle. In order for the Following Vehicle to create its own target path using the proposed path planning algorithm, longitudinal speed, lateral speed, yaw rate, and kingpin angle are required. Among them, vehicle speed, yaw rate, and kingpin angle can be measured, but their values are vulnerable to sensor noise. Since all signals measured by each truck are measured in its own coordinate system, the driving trajectory of the preceding vehicle received via V2V communication must be converted to fit the local coordinate system of the recipient truck.

For our specific model of truck platooning, each truck has a so called – lifespan through forming the platoon and a certain truck may undergo through the process several times continuously rather than only once. Figure 1 depicts an abstract state machine diagram of alignment behavior of each truck.

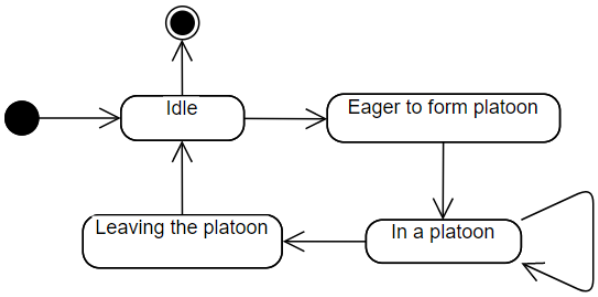


Figure 1: State Machine Diagram of Alignment Behavior of each Truck

On the other hand, the operational behavior of the system appears to seem more complex, as it considers all states and consequences through forming a platoon, truck position detection, platoon validity, controlling trucks along the platoon, applying the corresponding logic to trucks etc. The inspection should be consistently executed, and the platoon should be steadily ready for the new task. If required to form a new platoon, some variables, including but not limited to acceleration, velocity, response time and sensitivity are being assigned and generated, and the date is being saved regularly. Figure 2 illustrates state machine diagram of platooning in details.

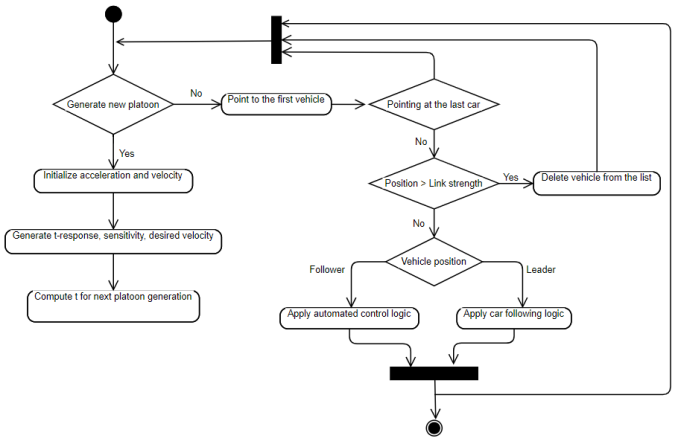


Figure 2: State Machine Diagram of Platooning

### *B. Control Behaviour*

Platoon is a group of vehicles moving together while maintaining a specific geometrical configuration without using any materials. Before specifying the various platoon configurations, lateral and longitudinal distances (Fig. 3) must be established. The geometrical definition of a platoon formation is based on these parameters. Lateral distance is the horizontal distance between two adjacent vehicles, while longitudinal distance is the vertical distance between two adjacent vehicles.

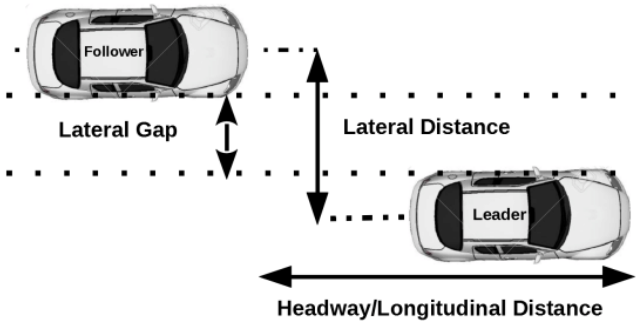


Figure 3: Lateral and Longitudinal Distances

The specification of both lateral and longitudinal distance is required when forming a platoon formation. Several platoon configurations can then be built based on the values of these variables. For example, widely used techniques (Figure 4) include but not limited to column configuration, line configuration, echelon configuration etc. On the other hand, column configuration of trucks has been concerned for our specific project which reflects the typical platooning method of placing vehicles one behind the other. The lateral required distance is null in this arrangement. Platoon column layouts are typically used to move passengers in urban or highway transit networks.

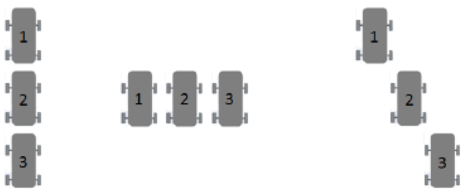


Figure 4: Column, Line and Echelon Configurations of Platoon

Two primary tendencies can be seen in the literature on platoon system control which are global and local approaches. Vehicles are localized in relation to a shared global reference in global approaches. Global techniques can be centralized or decentralized, depending on whether decision-making and control are handled by a single device. The use of a global strategy reveals exact trajectory matching. They are, however, dependent on advanced global positioning systems and dependable vehicle-to-vehicle communication.

Local approaches, on the other hand, are constantly decentralized where every vehicle determines a specific reference based solely on its own observations [6]. These rely solely on distance-measuring devices, which are typically less expensive than global solutions. The results produced in trajectory matching are not as good as in global systems because just a local view is used. In general, curves exhibit anticipation error [7][8]. The military solutions listed in the introduction might be classified as global or local. Relative to the unit center and approaches that are based on a leader can be thought of as global (the control can be centralized or decentralized).

Neighbor-referenced techniques, on the other hand, are local and decentralized. Considering all, it is possible to concentrate on how to implement the platoon function in a variety of configurations. To deal with this, several previous efforts in the field of mobile robots can provide inspiration for this problem [9]. Leader-following approach is being used in order to avoid the control problem of robot formation, where a robot or in our case truck, is regarded as a leader and follows a preset trajectory [10]. Other trucks follow the leader and keep a set distance and orientation from him. The fundamental flaw with this strategy is that it is heavily reliant on the leader vehicle to achieve its objectives.

Moreover, further problems may appear while considering control behavior such as distance instability, communication failure etc. Certain solutions should be developed, and these steps need to be applied throughout the procedure. Several precautions have been considered in order to guarantee the distance to the precedence truck:

1. Applying Desired Acceleration, for which the source is Wireless communication and RADAR.
2. Maintaining Relative Velocity, where the source is RADAR and Wireless communication as well.
3. Maintaining & Calculating Distance, for which the source RADAR and Camera.

Another important failure type to keep in mind is communication failure as the system should be designed in a style that it remains stable in such situations. For truck platooning design, the robustness of the system is ensured for two types of failures:

1. Wireless communication failure - loss of desired acceleration from the leading truck.
    - Since it is a major failure, it will impact platooning highly.
    - As per current design, without the wireless communication the platooning will not be continued.
    - The communication lost trucks will exit the platooning.
    - By applying gradual braking and with proper signaling, the truck will be stopped in the same lane.

2. Radar failure - loss of distance & relative velocity data between the trucks.
    - It is a minor failure, when compared with the loss of wireless communication.
    - The loss of distance & relative velocity data can be retrieved from cameras and wireless communication respectively.
    - Even with the loss of radar failure, the platooning will continue its work with the help of wireless communication & cameras.

Figure 5 depicts the activity diagram for control behavior in details.

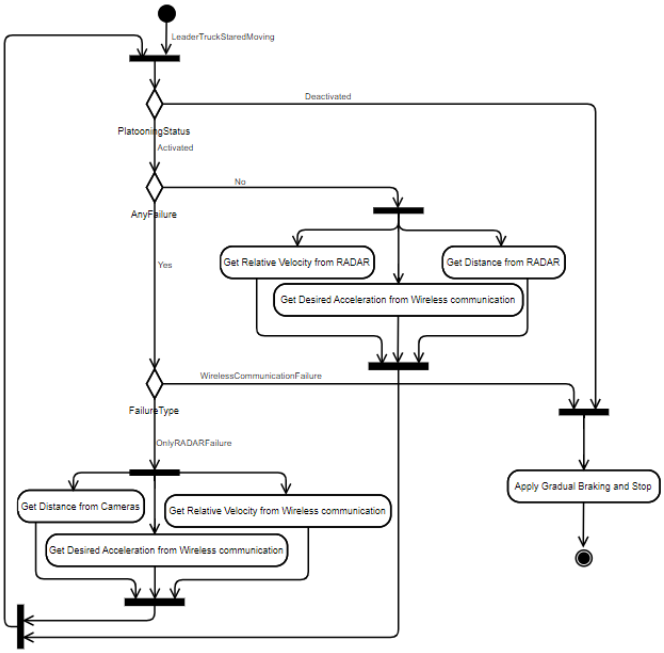


Figure 5: Activity Diagram of Control Behavior

## C. *Concurrent Programming*

Another important aspect to any project analysis is mapping it to a code. The software flow of the system should return understandable outputs to explain steps and results clearly. For this example of truck platooning C code has been supplied in order to implement the pure functional behavior of the project. As the project involves multiple functions happening simultaneously, it is not so called desirable, to miss any of the outputs, which brings new terms like pthread and openmp into action.

As the sketch of proof, concurrent programming is tried using pthread. Some of the instruction and loop has been tried to run concurrently using pthread. As C language doesn't have in-build library for multi-threading, P-thread is used to implement the parallel programming. P-thread is a very low-level programming which directly deals with the threads of the core. Also gone through the parallel programming using OpenMP.

Regarding the functional behavior of the platooning project, the expected outputs of the program are x and y coordinates of each truck updating dynamically, after including the desired number of trucks, the distance between the trucks and the starting coordinates of the leader truck. The platoon may continue the drive with the same increment, or it may be stopped entering preset default (0 for this case). Figure 6 depicts command window of the program:

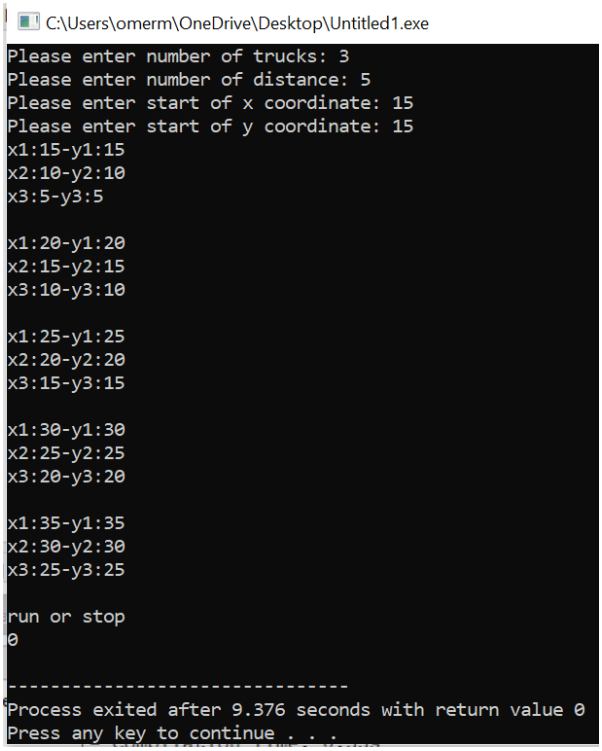


Figure 6: Command Window of Pure Functional Program

Figure 7 illustrates steps followed to obtain desired outputs.

```
arrx[0] = x1;
arry[0] = y1;
for(int i = 1; i < nTrucks; i++)
{
    arrx[i] = arrx[i-1] - dis;
    arry[i] = arry[i-1] - dis;
}

while(true)
{
    for(int k = j*nSteps; k < nSteps*(j+1); k++)
    {
        for(int i = 0; i < nTrucks; i++)
        {
            printf("x%d:%d-y%d:%d\n",i+1,arrx[i] + k*dis,i+1,arry[i] + k*dis);
        }
        printf("\n");

    }
```

Figure 7: Pure Functional Program Part-1

It should be mentioned that x1 and y1 stands for the coordinates of the leader truck and it continues correspondingly. The program updates and prints out the results for the entered value of desired times. After execution, there is an option of continuing to update the variables by entering any number other than zero, or we may stop the program simply entering zero. Figure 8 depicts the concept clearly.

```
        printf("run or stop ");
        printf("\n");
        scanf("%d",&platooning);
        //printf("%d",platooning);
        switch(platooning)
        {
            case 0:
                goto EndWhile;
                break;

        }
        j++;
    }
```

Figure 8: Pure Functional Program Part-2

### D. Architecture and Platooning's Architecture Case Studies

A concrete and successful system needs an architectural model before its implementation. It shall be the outline, base and primary one for the structure of an any system. The model needs to fulfill both present and future requirements [11]. The authors of this paper [12], have done research on electronic coupling in the freight truck. And their main technology source of dependence is on Advance Driving Assistance System (ADAS), Vehicle infrastructure communication (UMTS) for communication between the trucks and the central server, Adaptive cruise control (ACC) for front sensor between the trucks to manage the cruise, Automatic Guidance (AG) to calculate the distance gap required to maintain the platoon, and vehicle to vehicle communication for remaining interactions between the trucks. Handling the communication messages in the platoon was clearly discussed in [13]. This research paper proposes a new type of Intelligent Transportation Systems (ITS) message called ECE messages based on the Cooperative-ITS (C-ITS) message structure as Cooperative Awareness Message (CAM), Decentralized Environmental Notification Message (DENM), Signal Phase and Timing (SPAT), and Map message (MAP). The message was handled by the Edge computing.

### E. Distributed Architecture

The designing of distributed architecture deals with communication entities, communication paradigm, roles & responsibility of the entities and the pattern of the network structure [1]. The designing of the architecture took placed in four phases, and they are as follows,

#### 1) Entities Identification

The nodes of the platooning distributed system are the trucks (A leading truck and all following trucks). But mentioning the truck as node is a system representation, but when it comes to programming point of view, we need to adopt better more abstraction showing the problem domain. On considering this, trucks are taken as nodes – component.

#### 2) Communicaton paradigm selection

And the communication paradigm is going to be "message passing" a type of Remote Invocation. This comes under Request-reply protocols, but this is varying with the message synchronization. In Request-reply protocol, the sender sent message and wait till the reply message receives from the receiver, i.e., message synchronization. Whereas it is also possible to have the Request- reply protocol to be asynchronous, here the sender doesn't need to wait. It is not mandatory for the sender and receiver to be aligned with the message and aware of each other.

#### 3) Roles and Responsibility of the entities

Basic idea behind this platooning distributed architecture is going to be in client-server. All the trucks are going to be the client and cloud computing is the server. Many cloud computing providers are there in the market, in those Amazon Web Service (AWS) gives exclusive service for Edge computing called Greengrass. The choice of edge computing is to handle the computation near the cellular range of platooning, and it also helps to tackle the network connectivity issue.

#### 4) Architectural pattern

The pattern of the architecture followed in this distributed system is layered. Specifically, it is Two-tier client-server design. The client [truck] is the one tier and the server [AWS] is another tier. When it comes to client type, it is going to be Fat-client modal. Because all the computation are designed to happen in the client side i.e., within the truck's platoon control unit and all the process related to the platooning is handled and the server is only used to publish the messages of the client in it. And after this phase the structure of the distributed architecture is shown in the Figure 9.

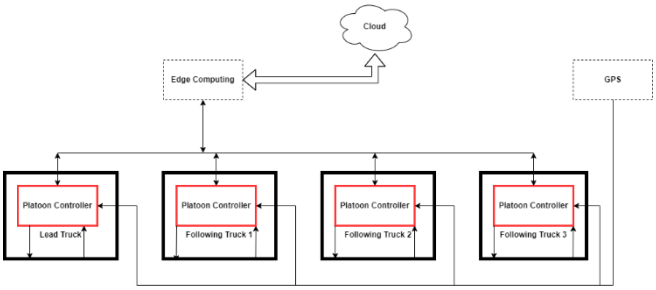


Figure 9: Platooning Distributed Architecture

### F. Platooning Primitive Requirements

The desired distance calculation is very important requirement of the platooning and lots of research has done in this measurement. And one among them is discussed in this paper [14]. It deals with stopping time and distance calculation by considering the communication delay. And calculate the condition for the safe or desired distance gap. By performing some important boundary conditions and real-life calculation, the formula derived in this paper can be used for the desired distance gap approximation.

### G. Parallel Architecture

Since platooning requires very deterministic deadline and high frequency, a designing of parallel architecture in the clients [Truck -node] can pave the way to achieve it. So, designing of parallel computing is handled in the truck's platooning control unit. This ECU gets all the data the from the LiDAR, Camera, GPS, and Data from server, process the data and calculate the acceleration or deceleration speed corresponding to the received data and update the ACC ECU accordingly. Parallel programming has multiple modals like instruction level, Data level, task level and loop level. Wherever the computing suits the parallelism, the corresponding parallel programming modal mentioned above are used. As the concrete design, all the data acquisition tasks (LiDAR, Camera, GPS, and Data from Server) are designed to follow task parallelism Fig. 10. This implementation required respective programming and as well as the hardware which supports parallelism. And this requirement makes to discuss about hardware selection in the following part.

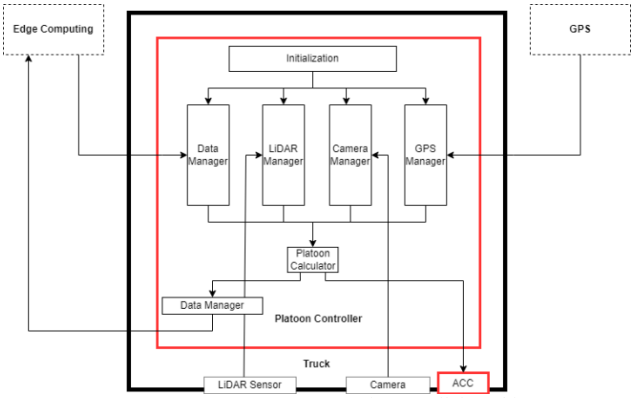


Figure 10: Platoon control unit's Parallel Architecture

### *H. Hardware Selection*

Without a proper hardware support i.e., number of resources or code or thread, implementing parallelism is not possible. So, we proposed and designed an ECU with multi-core. This multi-core ECU along with parallel programming increased the efficiency of the system double the time. And this has been proved by the real-time scheduling simulator. The simulations for the Earliest Deadline First (EDF) scheduling with single-core Fig. 11 and with multi-core Fig. 12 can be seen.

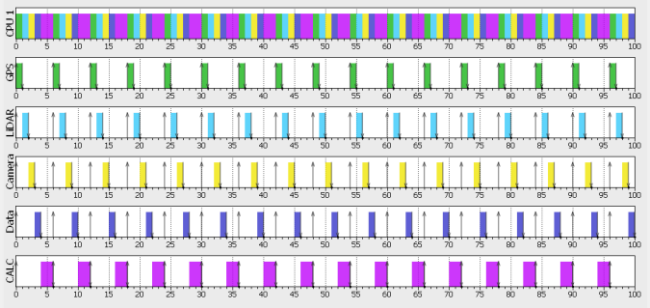


Figure 11: Single core EDF scheduling

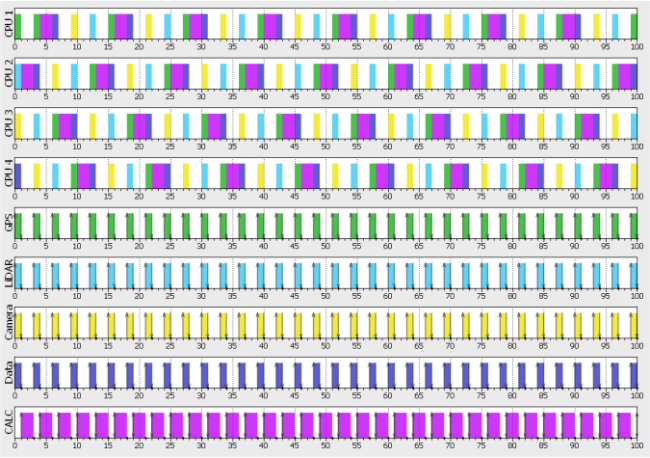


Figure 12: Multi core EDF scheduling

By choosing multi-core instead of single core the performance gets doubled with halved CPU load. Multi-core is not cost-effective when compared with single core, but when it comes to performance wise, multi-core will be the solution.

## III. Implementation Section

After the analysis and design part, the realization has started with implementation section in the following order.

### *A. Code Implementation*

To create the software, I utilized CPP as a programming language and Visual Studio 2019 as a development IDE. I have 3 classes which are going to be executed: Truck, Camera and Radar. To implement this object corresponding header and C++ files are created.

#### 1. Truck Class

Truck header and C++ file are executed to create truck object in the source file. In the figure xx there are shown the variables that we used to while creating the object. It is decided that the object will be created as a real-world truck. Because each truck has different length and starting and ending points of the are different, we created front and back coordinates for both longitude and latitude (for the sake of usefulness they are defined as x and y coordinates). Defining these coordinates in a separate way will help us to calculate distance between the trucks in an efficient way, Idle variable is used to check that weather the truck is available or not? On_move variable is utilized to check the truck is moving or not? To distinguish one truck from the another there should be unique variable. Because we cannot define the id to the all the trucks in the world, plate number is utilized. Because plate number is unique for all each truck and there is no chance of two trucks in the world have the same plate number. This plate number is utilized in the camera function to check the for intruder in the system. For ensuring the trucks is situated in the predefined order and there is truck no between the adjacent ones each truck stores the plate number of the previous truck. Truck id is assigned to each truck. OOP logic is implemented here, and all these variables are private. Writing and reading is done with getter and setter functions.

#### 2. Camera Class

It is assumed that there is a camera on each truck, and it should dynamically check for previous trucks plate number. This algorithm works as following: There are n number of trucks in the system, and we should check the intruder case for each of them. I pass the variable to the function and via using findIndex() function I can get the current index of the truck. By comparing the previous truck plate number with the one less indexed trucks plate number in the truck system I can ensure that they are joined or not. If it is equal, it means that the system working as proposed and there is no intruder. If they are not equal, there is an intruder between the trucks. Because I now the indexes of the trucks, number of the trucks is printed out to show the user that between which trucks the problem occurs.

#### 3. Radar Class

In the real world the distance between the trucks should be defined by using the radars. Because in this project we only utilized software, distance is calculated using the coordinates of the trucks. Here front and back coordinates of the trucks come into the play. To find the distance between the trucks, we subtract back coordinate of the previous truck from the front coordinates of the following trucks. Then we apply mathematical calculation and get the distance between the trucks. This distance is also calculated in repeated way to ensure that the distance is kept as desired.

```
Please insert number of trucks in the platoon:3
Please insert front x coordinate of truck 1:10
Please insert back x coordinates of truck 1:15
Please insert front y coordinate of truck 1 :10
Please insert front y coordinate of truck 1 :15
Please enter desired distance between the trucks:5
Please insert plate number of truck 1 :a
Please insert plate number of truck 2 :b
Please insert plate number of truck 3 :c
Distance between 2 and 1 is :5
Distance between 3 and 2 is :5
```

Figure 13: Radar message after calculating distance

4. Main code

Previously explained ones are the classes that is written to have template functions to utilized them in the Source.cpp. Initially in the code we need to initialize the platoon. To created it vector data structure is utilized. As it is mentioned earlier in the report, in order to be more precise, longitude and latitude variables should be updated for both front and back of the truck. In the initial case the inserted variables are the coordinates of the leading truck. To find the coordinates of the following trucks, distance is added to x coordinate. Because we are calculating the distance between the trucks, updating both longitude and latitude creates problem. While updating two variables, the distance will be greater, and we will always face with distance error. To prohibit this problem from arising, it is assumed that the trucks only going in one direction and there is no need to update the other variable. In our case we used x coordinate as a reference. By using this method all the coordinates of the trucks are defined. Plate number is insert for each individual truck and simultaneously plate number of the previous trucks is also assigned. Once all the insertions are done, platoon is created as a vector, and it is ready to use.

The figure demonstration of the platoon. Both front and back coordinates are demonstrated in the order. Only latitude variable updated via using the distance variable. And “a”, “b” and “c” is the plate numbers of the trucks. As I mentioned previously each truck will store the plate number of the previous one to make sure that there is no intruder between them. First truck does not have ancestor, so there is not previous truck data there.

```
Front latitude :12
Back latitude :12
Front longitude :15
Back longitude :15
a  ==>

Front latitude :15
Back latitude :15
Front longitude :15
Back longitude :15
b  ==> a

Front latitude :18
Back latitude :18
Front longitude :15
Back longitude :15
c  ==> b
```

Figure 14: Demonstration of platooned truck with data

5. Parallel programming

Checking for the intruder in the systems is main operation of the system and it is called repeatedly. Intruder is checked for each truck and it should be in the same time. It is done via using parallel programming and I used thread (built-in library). Firstly, I created threads vector that is equal to the size of the platoon vector. Then each thread is assigned to the function with id. Via this one parallel programming is achieved, and it is checked in the different cores.

```
Please insert number of trucks in the platoon:5
Please insert front x coordinate of truck 1:10
Please insert back x coordinates of truck 1:15
Please insert front y coordinate of truck 1 :10
Please insert front y coordinate of truck 1 :15
Please enter desired distance between the trucks:5
Please insert plate number of truck 1 :a
Please insert plate number of truck 2 :b
Please insert plate number of truck 3 :c
Please insert plate number of truck 4 :d
Please insert plate number of truck 5 :e
Turck 2 and 1 are running suffeciently Turck 5 and 4Turck 3 and 2 are running suffeciently
 are running suffeciently
Turck
4 and 3 are running suffeciently
```

Figure 15: Intruder checking with threads

6. GPU Implementation

GPGPU (General Purpose Computing on GPU) is the use of the GPU (Graphical Process Unit) together with the CPU (Central Processing Unit) to speed up computation in applications handled only by the CPU.

CPU consists of small number of cores to carry out sequential serial programming, but GPU consists of many cores to handle multiple tasks simultaneously.

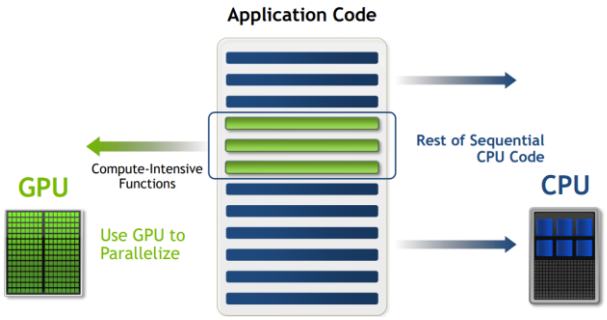


Figure 16: CPU / GPU Usage

In Truck platooning application, each truck has an initial speed. This shows the speed at which trucks join the platoon. In order to show this situation in our application, the speed value is requested for each truck created from the user. Vehicles in the Platoon System must follow each other with an average speed. This value is represented by the variableSpeed of the arrangeSpeed function running on the GPU. While the Platoon system is created with the initiliazePlatoon function, the speed value for each truck is entered from the console. In order to see the usage of the GPU more clearly, the initial speed value is adjusted to the average value, increasing or decreasing by 1 per loop cycle. The GPU is not able to see the system RAM, therefore data must be copied back and forth between the host and device. The cudaMalloc function is used for device memory management. With the help of the device pointers we created, the required memory is reserved in the GPU. cudaMalloc function reserves memory in the devices on board global memory. It takes device's pointers as the first argument and the second argument is how many bytes to reserve. After that to copy data between device and host, cudaMemcpy function is used. This function copies data to and from the device and host memory. First argument is the destination, second argument is source, third argument is the size of the variable, and fourth argument illustrates the direction. After calculation completing, data are copied from device to host with cudaMemcpy function again and finally memory on the device previously allocated by a cudaMalloc releases.

Usage of the CPU and GPU is illustrated in the following figures.

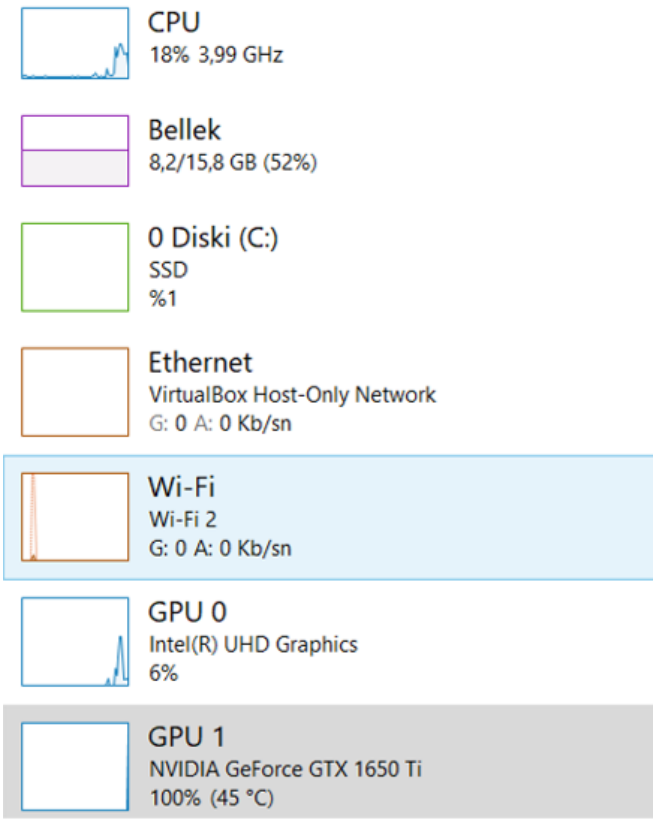


*Figure 1: CPU / GPU Usage of the Code Implementation*

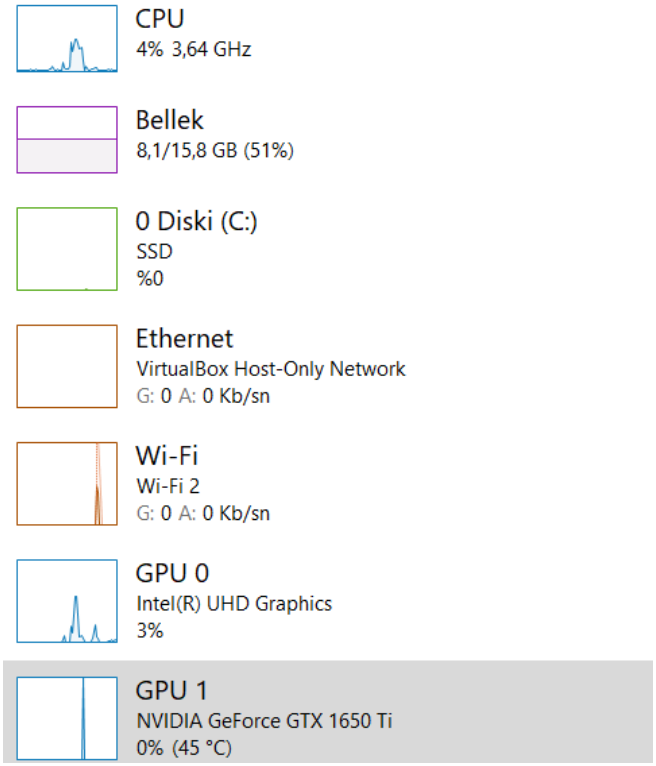


Figure 17: CPU / GPU Usage of the Code Implementation

```
Current speed of t1: 80
Speed is setting to avarage speed for t1...
Current speed of t2: 120
Speed is setting to avarage speed for t2...
Current speed of t3: 150
Speed is setting to avarage speed for t3...
Current speed 81.000000
Current speed 82.000000
Current speed 83.000000
Current speed 84.000000
Current speed 85.000000
Current speed 86.000000
Current speed 87.000000
Current speed 88.000000
Current speed 89.000000
Current speed 90.000000
Current speed 91.000000
Current speed 92.000000
Current speed 93.000000
Current speed 94.000000
Current speed 95.000000
Current speed 96.000000
Current speed 97.000000
Current speed 98.000000
Current speed 99.000000
Current speed 100.000000
Avarage speed achieved...
Current speed 119.000000
Current speed 118.000000
Current speed 117.000000
Current speed 116.000000
Current speed 115.000000
Current speed 114.000000
Current speed 113.000000
Current speed 112.000000
Current speed 111.000000
Current speed 110.000000
Current speed 109.000000
Current speed 108.000000
Current speed 107.000000
Current speed 106.000000
Current speed 105.000000
Current speed 104.000000
Current speed 103.000000
Current speed 102.000000
Current speed 101.000000
Current speed 100.000000
Avarage speed achieved...
Current speed 149.000000
```

Figure 18: Output of Function Running on GPU

## IV. Simulation

Simulation is done by CPP code. Once the corresponding data is inserted platooned vector is created. With this vector all the checking and disinserting functionalities are done. Initially, gap between trucks is checked to make sure that they are running with desired distance. Right after that, checking for intruder function is running. If there is no intruder, it is giving all the neighbor trucks are running sufficiently. Finally, to show deleting functions is working correctly, the platoon is shown then platoon after deleting one element is demonstrated. The last element in each truck is shown the current plate number and plate number of the ancestor. Because leading truck does not have any ancestor there is no corresponding previous plate number in this case. It is obvious from the printing of the platoon after removing one element, there is no collision between the data and platoon is ready to run with one less element.

```
Checking for the distance between the trucks :
Distance between 2 and 1 is :5
Distance between 3 and 2 is :5

Checking for the intruder for all trucks :
Turck 2 and 1 are running suffeciently
Turck 3 and 2 are running suffeciently

Printing platoon before removing element:
Truck 1
Front latitude :10
Back latitude :10
Front longitude :15
Back longitude :15
10aa100  ==>

Truck 2
Front latitude :15
Back latitude :15
Front longitude :15
Back longitude :15
10aa101  ==> 10aa100

Truck 3
Front latitude :20
Back latitude :20
Front longitude :15
Back longitude :15
10aa102  ==> 10aa101


Printing platoon after removing element
Truck 1
Front latitude :10
Back latitude :10
Front longitude :15
Back longitude :15
10aa100  ==>

Truck 2
Front latitude :15
Back latitude :15
Front longitude :15
Back longitude :15
10aa101  ==> 10aa100
```

Figure 19: Simulation of the all platoon

## Conclusion

This project deals with the system designing more specific to the architecture. Distributed network helped to connect the trucks in the platooning as a single system and parallel computing increases its efficiency to achieve the overall goal. It also pushes to undergo research on hardware selection, which is very required for the respective architectures. Implementing the parallelism and increasing the performance triggered the experiences of multi-core programming like openMP, p-thread and GPU programming as OpenCL and CUDA.


## Acknowledgment

We sincerely thank Prof. Dr. Stefan Henkler, who supported us throughout the journey of this project.


## References


[1] G. F. Coulouris, Ed., Distributed systems: concepts and design, 5th ed. Boston: Addison-Wesley, 2012

[2] Intelligent Transport Systems (ITS); European Profile Standard for the Physical and Medium Access Control Layer of Intelligent Transport Systems Operating in the 5 GHz Frequency Band, ETSI ES 202 663 (V1.1.0).

[3] S. E. Li, Y. Zheng, K. Li and J. Wang, "An overview of vehicular platoon control under the four-component framework," 2015 IEEE Intelligent Vehicles Symposium (IV), 2015, pp. 286-291, doi: 10.1109/IVS.2015.7225700.

[4] Andreas Festag, Standards for vehicular communication – from IEEE 802.11p to 5G

[5] White, R.; Tomizuka, M. Autonomous following lateral control of heavy vehicles using laser scanning radar. In Proceedings of the 2001 American Control Conference, Arlington, VA, USA, 25–27 June 2001; Cat. No.01CH37148. IEEE: Piscataway, NJ, USA, 2001; pp. 2333–2338.

[6] J. Contet, F. Gechter, P. Gruer, A. Koukam, Bending virtual spring-damper: A solution to improve local platoon control, in: ICCS (1), 2009, pp. 601–610.

[7] P. Ioannou, Z. Xu, Throttle and brake control systems for automatic vehicle following, IVHS Journal 1 (4) (1994) 345 –.

[8] J. J. Moskwa, J. K. Hedrick, Nonlinear algorithms for automotive engine control, IEEE Control Systems Magazine 10 (3) (1990) 88 – 93.

[9] T. Balch, R. Arkin, Behavior-based formation control for multirobot teams, Robotics and Automation, IEEE Transactions on 14 (6) (1998) 926 –939.

[10] J. Lawton, R. Beard, B. Young, A decentralized approach to formation maneuvers, Robotics and Automation, IEEE Transactions on 19 (6) (2003) 933 – 941.

[11] G. F. Coulouris, Ed., Distributed systems: concepts and design, 5th ed. Boston: Addison-Wesley, 2012

[12] R. Ramakers, K. Henning, S. Gies, D. Abel and H. M. A. Max, "Electronically coupled truck platoons on German highways," 2009 IEEE International Conference on Systems, Man and Cybernetics, 2009, pp. 2409-2414, doi: 10.1109/ICSMC.2009.5346393.

[13] N. Bouchemal and J. -Y. Jun, "V2X Architecture for Autonomous Platoon Management In Urban Environment," 2021 International Conference on Computer, Information and Telecommunication Systems (CITS), 2021, pp. 1-6, doi: 10.1109/CITS52676.2021.9617911.

[14] S. Ellwanger and E. Wohlfarth, "Truck platooning application," 2017 IEEE Intelligent Vehicles Symposium (IV), 2017, pp. 966-971, doi: 10.1109/IVS.2017.7995840.

# Annex

1. SOURCE CODE
    a. Functional Behavior of Concurrent programming

```
#include<stdio.h>
#include<conio.h>
#include <omp.h>
#include <stdbool.h>

void *MeasureRvelocity();
void *MeasureDistance();
void *GetAcceleration();

int *initializePlatoon(int n,int d, int x1, int y1);

int main()
{

  /*Initializing the variables of Platoon*/
        int nTrucks;
        int dis;
        int x1, y1;
        int rVelocity;
        int accel;
        int choice;
        int nSteps = 5;
        int j = 0;
        bool platooning;

        printf("Please enter number of trucks: ");
        scanf("%d", &nTrucks);
        printf("Please enter number of distance: ");
        scanf("%d", &dis);
        printf("Please enter start of x coordinate: ");
        scanf("%d", &x1);
        printf("Please enter start of y coordinate: ");
        scanf("%d", &y1);


        int * arr;


        int arrx[nTrucks];
        int arry[nTrucks];
        arrx[0] = x1;
        arry[0] = y1;
        for(int i = 1; i < nTrucks; i++)
        {
                arrx[i] = arrx[i-1] - dis;
                arry[i] = arry[i-1] - dis;
        }

        while(true)
        {
                for(int k = j*nSteps; k < nSteps*(j+1); k++)
                {
                        for(int i = 0; i < nTrucks; i++)
                        {
                                printf("x%d:%d-y%d:%d\n",i+1,arrx[i] + k*dis,i+1,arry[i] + k*dis);
                        }
                        printf("\n");

                }

                printf("run or stop ");
                printf("\n");
                scanf("%d",&platooning);
                //printf("%d",platooning);
                switch(platooning)
                {
                        case 0:
                                goto EndWhile;
                                break;

                }
                j++;
        }
        EndWhile: ;
        return 0;

 //initializePlatoon();

/*Loop of the PLatoon*/
```

```
    /* - coordinates of the truck want to be tracked*/
    /* -*/

  // while()
   //{
     /*Error handling*/
                        /*Checking for coupling & de-coupling*/
   //}
}
```

b. Functional running in GPU

```
_global_ void arrangeSpeed(double* speed) {

        double avarageSpeed = 100.0;

        if (speed[0] < avarageSpeed)
        {
                while (speed[0] < avarageSpeed)
                {
                        speed[0]+=1.0;
                        printf("Current speed %f \n" , speed[0]);
                }
        }
        else {
                while(speed[0] > avarageSpeed)
                {
                        speed[0]-=1.0;
                        printf("Current speed %f \n", speed[0]);
                }

        }

        printf("Avarage speed achieved...\n");
}
```

c. Memory Management for GPU

```
double speed_t1 = platoon.at(0).getSpeed();
double* d_speed_t1; //device pointer
double speed_t2 = platoon.at(1).getSpeed();
double* d_speed_t2; //device pointer
double speed_t3 = platoon.at(2).getSpeed();
double* d_speed_t3; //device pointer

cudaMalloc(&d_speed_t1, sizeof(double));
cudaMalloc(&d_speed_t2, sizeof(double));
cudaMalloc(&d_speed_t3, sizeof(double));

cudaMemcpy(d_speed_t1, &speed_t1, sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_speed_t2, &speed_t2, sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(d_speed_t3, &speed_t3, sizeof(double), cudaMemcpyHostToDevice);

cout << "Current speed of t1: " << platoon.at(0).getSpeed() << endl;
cout << "Speed is setting to avarage speed for t1..." << endl;
arrangeSpeed << < 1, 1 >> > (d_speed_t1);
cout << "Current speed of t2: " << platoon.at(1).getSpeed() << endl;
cout << "Speed is setting to avarage speed for t2..." << endl;
arrangeSpeed << < 1, 1 >> > (d_speed_t2);
cout << "Current speed of t3: " << platoon.at(2).getSpeed() << endl;
cout << "Speed is setting to avarage speed for t3..." << endl;
arrangeSpeed << < 1, 1 >> > (d_speed_t3);

cudaMemcpy(&d_speed_t1, d_speed_t1, sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(&d_speed_t2, d_speed_t2, sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(&d_speed_t3, d_speed_t3, sizeof(double), cudaMemcpyDeviceToHost);

cudaFree(d_speed_t1);
cudaFree(d_speed_t2);
cudaFree(d_speed_t3);
```

2. LINE of CODE
    a. Functional behavior of concurrent programming = 88 lines
    b. GPU programming = 90 lines
    c. Node implementation = 95 lines

3. CONTRIBUTION PERCENTAGE

    Four members 25% each.

Affidavit

We Omar Mammadov, Nijat Dashdamirov, Namik Mert Tuncbilek, Vignesh Arumugam herewith declare that we have composed the present paper and work ourselves and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

Dortmund, 20.02.2022,

Omar Mammadov 

Nijat Dashdamirov 

Namik Mert Tuncbilek 

Vignesh Arumugam