EPiC Series in Engineering
Volume 2, 2018, Pages 152–161
SUMO 2018- Simulating Autonomous and Intermodal Transport Systems

A new strategy for synchronizing traffic flow on a distributed simulation using SUMO

Nicolás Arroyo[1], Andrés Acosta[1], Jairo Espinosa[1], and Jorge Espinosa[2]

[1] Universidad Nacional de Colombia, Medellín, Antioquia, Colombia
{jarroyo,afacosta,jespinov}@unal.edu.co
[2] Politécnico Colombiano Jaime Isaza Cadavid, Medellín, Antioquia, Colombia
jeespinosa@elpoli.edu.co

Abstract

The Project Modelling and Control of Urban Traffic in the City of Medellín (MOYCOT) has produced multiple results in modelling, simulation and control of multimodal urban traffic using the SUMO simulator. As the simulations became more complex the necessity to distribute the computational load rose. Therefore, an approach for network partitioning and border edges management was introduced. In this paper a new border edge management strategy is presented for distributed simulation with SUMO. Unlike the previous approaches, which were developed in Python programming language using the corresponding TraCI client and tools such as sumolib, the strategy presented in this work was developed in C++ using the TraCI client for this language. Additionally, this strategy involves a simplified process for network partitioning since the border edges are preserved in every partition, without the need of splitting them. In this case, neighboring partitions behave in a master-slave fashion, depending on whether the border edge is an incoming edge or an outgoing edge. Concretely, a given partition is a master for its incoming edges and a slave for its outgoing ones. Furthermore, all the vehicles are found in the master and the slave partitions, where the master partition controls its slaves through the TraCI commands **slowDown** and **moveTo** that correct the position of these vehicles. Simulation results show that this new strategy presents better precision than the previous one. The description of the new procedure for border edge management is detailed. Finally, it is compared with the previous approach and the non-distributed simulation using a free flow scenario and a scenario with queue formation is presented.

1 Introduction

One of the key features of the Simulation of Urban MObility (SUMO) package is its high capacity of integration with open-source and proprietary platforms [8]. Natively, SUMO integrates with other microscopic and macroscopic simulators such as VISUM, Vissim, OpenDRIVE and MATsim; and with information systems including Open Street Maps (OSM), greyscale height maps and Shapefiles (e.g. ArcView). Furthermore, some open-source tools can convert from other formats to those used by SUMO. For instance, one could have data in Google Earth, save it as a KMZ file and convert it to a Shapefile using QGis. On the other hand, the community has developed many other integration modules for proprietary traffic lights controllers [2], other map formats [3] and 3D rendering engines [5], among others. Additionally, the TraCI interface, with its multiple implementations for several programming languages, allow to integrate algorithms and methods for testing novel traffic management strategies and models. These characteristics made possible to have realistic large-scale and complex scenarios. Currently, SUMO has scenarios for the cities of Luxembourg, Bologna and Cologne. As the scale and complexity of the scenarios and applications grow, so does the computational capacity required for each time step, which is a particularly important aspect for monitoring and real-time applications. For this reason, there is an increasing interest in giving SUMO the ability of working in parallel. When it comes to improving the performance of applications involving several independent runs of the same simulation scenario, the required solution is almost straightforward. The DUAROUTER tool, for instance, can use several threads for finding the shortest paths from predefined origins and destinations. Non gradient-based search methods can also benefit from this approach, where the SUMO simulator is often used as a black-box model for evaluating a performance function [7]. The interest of this article is on parallel implementations of SUMO that seek to improve the performance of a single simulation instance, which is a situation commonly found in large-scale scenarios for monitoring and real-time applications. This has been achieved with the microscopic simulator Aimsun, using NVIDIA Graphical Processing Units [6]. In their work, the authors proposed a data-parallel agent-based strategy for microscopic simulation, where the microscopic models used by Aimsun 8.7 are reproduced so that every driver-vehicle unit is modeled as a state machine, using the agent-based modeling package FLAME-GPU. Simulation results showed a performance improvement in benchmarks with Manhattan scenarios, varying their size and the traffic demand. However, the work [6] does not include accuracy tests to validate their implementation of the Aimsun 8.7 microscopic models. Likewise, Potuzak obtained a performance improvement in parallel microscopic traffic simulations, this time using the distributed implementation of the Java Urban Traffic Simulator (JUTS), namely Distributed Urban Traffic Simulator (DUTS), which utilizes shared memory for communications. Two schemes were proposed by Potuzac: purely distributed and parallel/distributed. These schemes differ in that the latter exploits multiple threads, with which a performance improvement of up to 52% was achieved for a scenario with two nodes and four cores, compared with a purely distributed scheme. In the case of the SUMO simulator, although there have been several approaches to get a performance improvement, their results are not satisfactory, or have not been reported, to the best of our knowledge. A critical aspect related to performance is that subsystems, which are obtained by spatially partitioning the simulation scenario, need to exchange messages for transferring vehicles and for synchronization. From now, we will refer to this aspect as synchronization. Bragard et. al. [4] used sockets for communication among partitions. Furthermore, they developed a module for managing local communications with the SUMO instances using TraCI. The authors found that, because of this communication overhead, the performance of their distributed platform is not better than centralized SUMO. Finally, they proposed to use distributed memory schemes in order to improve the performance of distributed SUMO. In a more recent work, we proposed a parallel method for SUMO simulations using Python with the multiprocessing package [1]. In this case, local communications were managed through TraCI and interprocess communications using pipes. However, the obtained results in regards to performance were not better than centralized SUMO. In order to reduce the overhead imposed by communications relying in TCP-IP sockets and pipes, this article proposes to implement a module that wraps the TraCI interface and incorporates shared memory for interprocess communication instead of TCP-IP sockets. Additionally, a new method for improving the accuracy of the parallel simulation is presented, which is based on a master-slave scheme where a given partition acts as a master for incoming traffic and as a slave for outgoing traffic. The strategy works by correcting the position of each vehicle found in the boundary edges at each time step. This article is organized, as follows: Section 2 details the implementation of the proposed synchronization strategy. Section 3 presents simulation results. Finally, section 4 concludes the article.

2 Implementation

To parallelize the simulation, partitioning of the scenario and its routes is required, to accomplish this the method proposed in [1] was used. From this, multiple scenarios were obtained to test their behavior when parallelized. Then, the partitions must be synchronized with each other to guarantee results similar to the centralized simulation.

2.1 Synchronization

A new synchronization method was proposed following a master-slave scheme. From the partitioning, the edges that are common to neighboring partitions can be obtained. One of the nodes of these common edges has to be a "dead-end" node while the other one is connected to the rest of the network of the partition. The strategy works in such a way that vehicles in the these boundary edges are found simultaneously in the partitions connected by them. If the vehicles of an edge travel towards the "dead-end" node it means that these vehicles will finish their trip in the current partition and are unaware of the state of the next one. In the other partition, the same vehicles travel from the "dead-end" node and are constrained to the conditions of this new partition, such as traffic lights, queues, accidents, etc. A method to propagate the effects of this new partition to the previous one must be implemented.

The partition in which the vehicles travel towards the "dead-end" node acts as a slave partition in that edge because the vehicles must follow, blindly, the conditions imposed by their counterpart in the edge of the master partition. This means that position, speed and acceleration of the vehicles in the slave partition are determined by those in the edge of the master partition and the edge in the master partition is left as is. Then, the new vehicles that enter the slave edge are entered in the master edge as well. A partition can act as both, master and slave, for the different edges, e.g. a two-way road. Figure 1 is an example in free flow and Figure 2 is an example in congestion of this method.

This method lets the effects from one partition to be propagated very easily to another one which increases the precision when compared with the centralized simulation, but it comes with a high time cost since it relies heavily on TraCI's communication. To reduce this time, the TCP/IP communications of TraCI were changed to shared memory communications using the boost inter-process library. To accomplish this, some minor changes were made to the **TraCI-Server** and the **TraCIAPI** packages, only the functions **readExact** and **receiveExact** of the socket class of the SUMO's tcpip library were changed to shared memory in order to keep full functionality of the TraCIAPI. To test if shared-memory TraCI was faster than the traditional TraCI, we ran a scenario that uses TraCI heavily, consisting on the collection of pollutants information on every simulation step. The simulation time was reduced, in average, to 60% of the centralized one that uses TraCI TCP/IP.

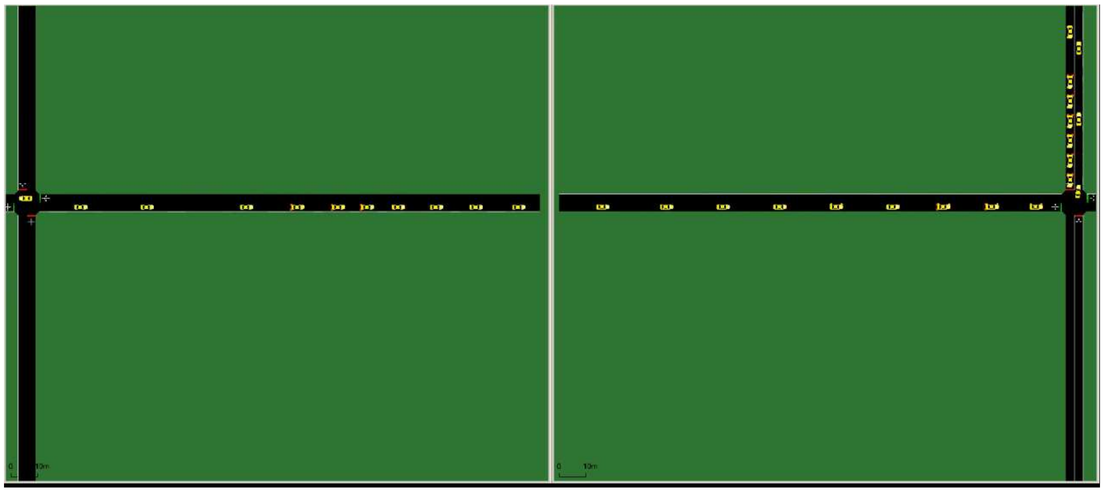


Figure 1: Vehicles in free flow, (Slave-left Master-right)

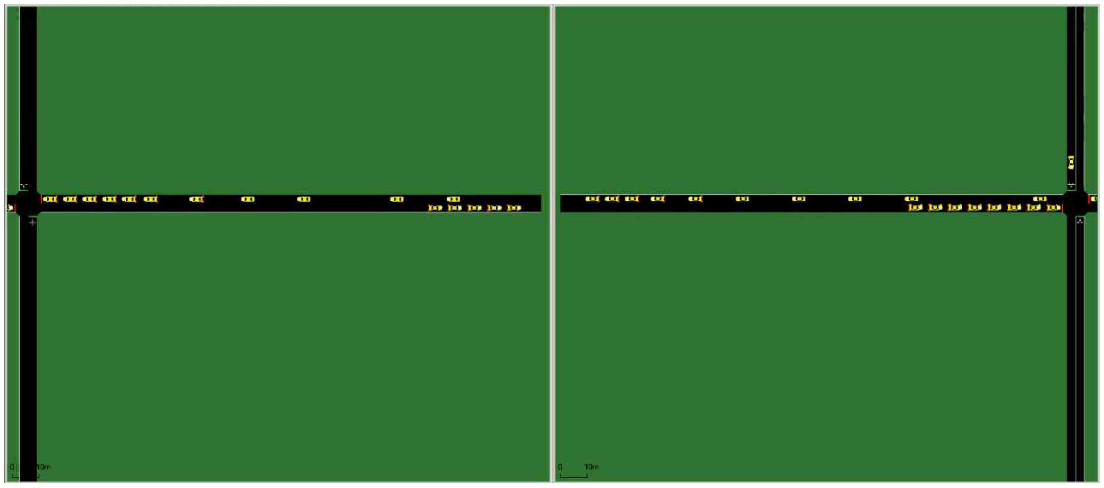


Figure 2: Propagation of queue, both partitions act as master and slave for the two edges

This algorithm was developed with the TraCIAPI in C++, in order to exploit the language's multi-threading and shared memory features. To manage the partitions, a class to collect the necessary data and send the synchronizing commands to each one of them and a class to synchronize all partitions were created. The commands used to synchronize the vehicles in slave boundary edges were **add**, **moveTo** and **slowDown** from the vehicle scope.

3 Results

Several simulations were executed with different scenarios, congestion, partitions and common edges. From these, performace and precision tests were carried out.

3.1 Scenarios

The implemented scenarios are based on a 20x20 square as seen in figure 3. Vehicles depart from the left side and the right side and their final destination is the opposite side. For this scenario, the routes of all vehicles are straight lines and the traffic demand was 3300 vehicles per hour. As it is, it can be divided into two partitions with 40 boundaries, 20 bidirectional roads, as shown in figure 4. Then, the number of boundaries and partitions were modified to create new scenarios for the analysis.

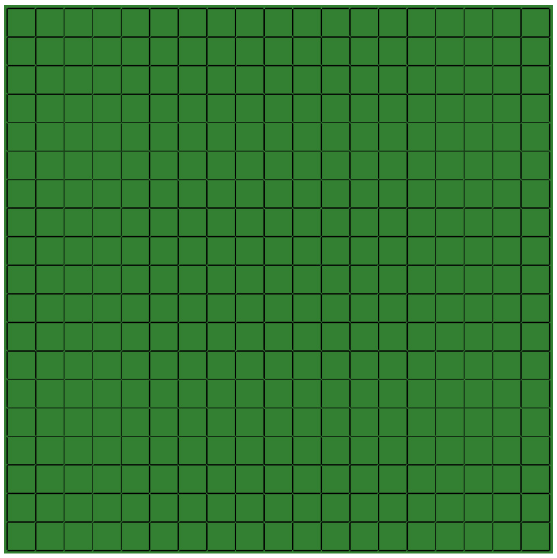


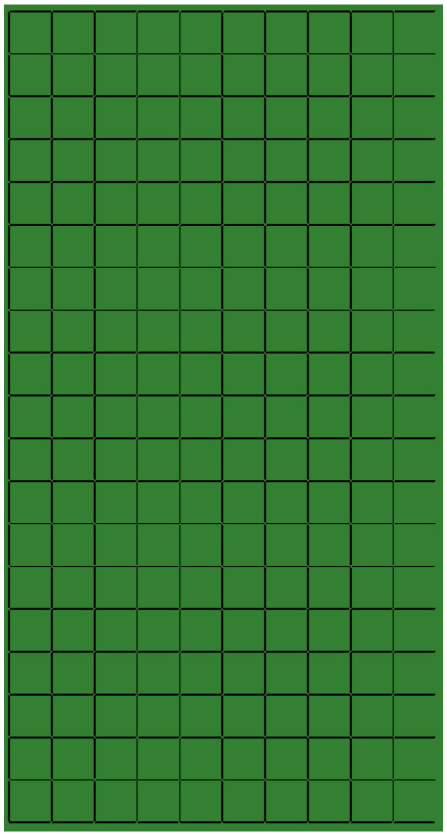
Figure 3: Base scenario for simulations

In order to analyze the effect of the number of boundaries, they were reduced. An example of a scenario with 8 boundaries can be seen in figure 5. This scenario can be divided into two partitions and the routes are not straight lines. Furthermore, the base scenario was also partitioned into 4 parts with 4 boundaries and with 8 boundaries as seen in figure 6. Finally, the base scenario was divided into 16 partitions (5x5 squares) with 10 common edges for each of the adjacent simulations.

3.2 Performace and precision

For performance, the time of the simulation was used; for precision, the distance between a vehicle in centralized and the same vehicle in parallel at the same time was used. All of the results shown use TraCI with shared memory.

The results from figure 7 show the execution time of simulations with 4, 8 and 16 boundary edges; results for 40 edges are shown in figure 8. Each of the scenarios was simulated 20 times. In general terms, the centralized simulations were always faster than the parallel ones and as the number of common edges was increased, the parallel simulation would take longer. For these centralized scenarios, TraCI was used to execute simulation steps and nothing else. This


(a) Left side of base scenario

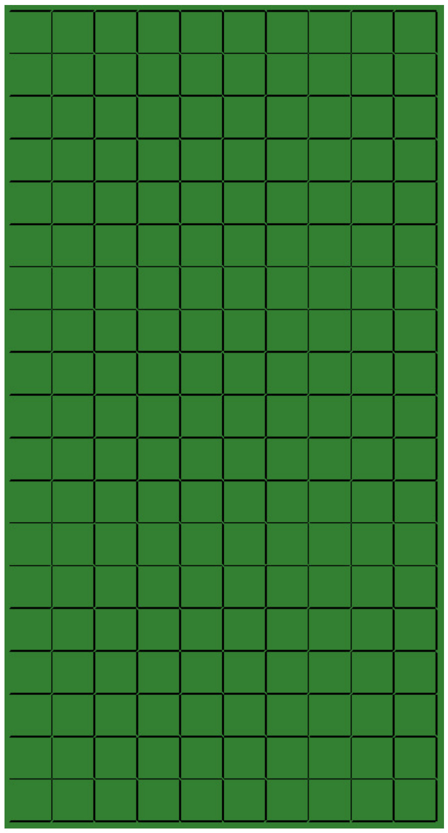

(b) Right side of base scenario

Figure 4: Base scenario on two partitions

behavior can be explained by the high usage of TraCI to synchronize the partitions which is much slower than the simulation. To prove it, performance of simulations where TraCI was heavily used was also measured. The position and name of all the vehicles were taken on every simulation step, the results are shown in tables 1 and 2.

	Free Flow		
Boundaries	Centralized Time [s]	Parallel Time [s]	Fraction [%]
4	238.3	85.2	35.8%
8	228.8	97.3	42.5%
40	196.2	121.8	62.1%

Table 1: Mean simulation times for 4 partitions and communication via TraCI in free flow

For these simulations, only 4 partitions were used, varying the number of boundary edges, and had 20 executions per scenario. The performance of the parallel one was almost a third of the time of the centralized one for many tests. Having multiple threads communicating with each of the simulations proved that most of the time of the whole process was spent in

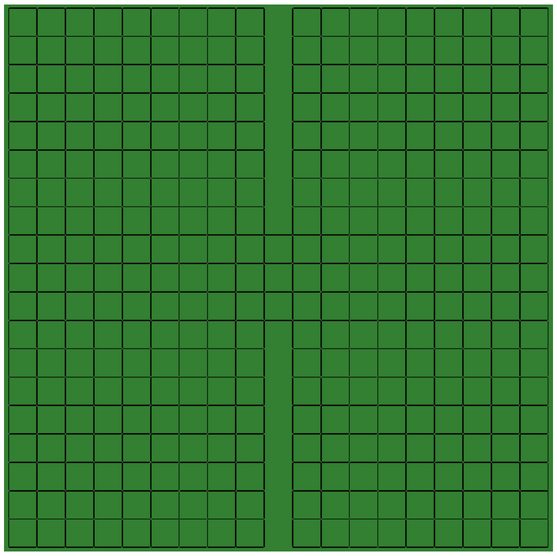
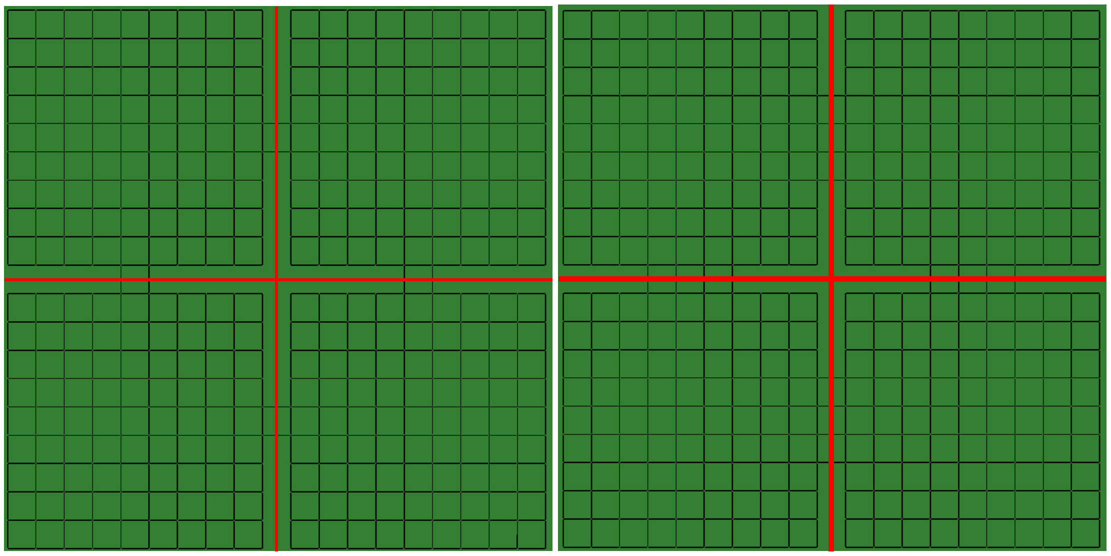


Figure 5: Base scenario with 8 boundaries

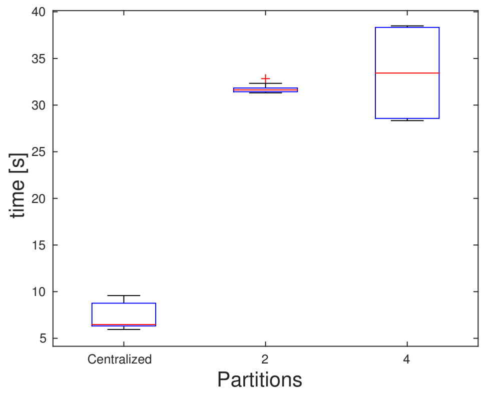


(a) Base scenario with 4 boundaries and 4 partitions (b) Base scenario with 8 boundaries and 4 partitions

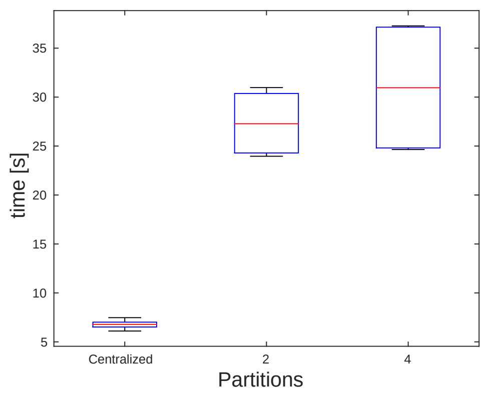
Figure 6: Base scenario with 4 partitions

communication via TraCI. Furthermore, the average utilization of the threads used for the parallel simulation did not exceed 5% of usage.

The precision for the different scenarios is shown in figure 9 The data collected for tables 2 and 1 were used but only in the 40 boundary-edge. These indices show the behavior for both congested and free flow conditions. In general, the error is around $0m$ but there are some

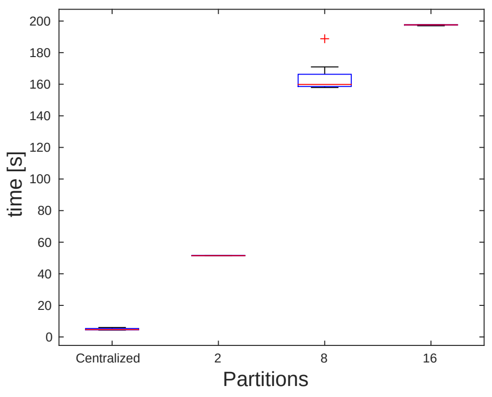


(a) Free flow

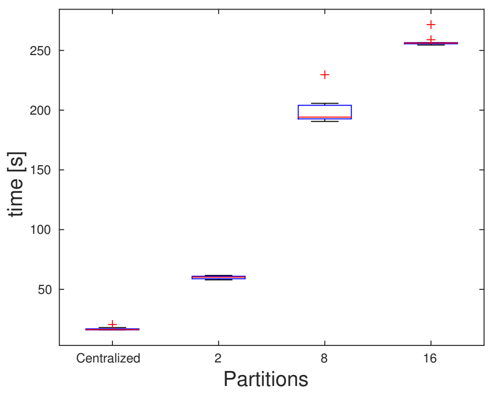


(b) Congestion

Figure 7: Simulation time for vehicles using different number of partitions



(a) Free flow



(b) Congestion

Figure 8: Simulation time for vehicles using different number of partitions and 40-edge boundary

cases where the mean error is higher. Samples are more than 3000 vehicles per simulation. As expected, the error for free flow condition is almost zero with some few exceptions, only two outliers for both 4 and 8 partitions scenarios. When the simulation encountered congestion, the outliers increased significantly but the average was still around zero and most of the errors were below 10 meters.

For this analysis, the most relevant portion of the trajectory of each vehicle is found once it enters the boundary edge, when it enters the new partition. At this moment the vehicle is being synchronized in the slave partition by means of the **moveTo** and **slowDown** TraCI commands. Note that in the congestion scenario, this implies that this congestion does propagate to the slave section but it is only taken into account when the vehicle is added to the new partition. From this point, the error is computed for each point of the obtained trajectory, for each vehicle of the simulation until each vehicle is out of the scope in this final partition. For

	Congestion		
Boundaries	Centralized Time [s]	Parallel Time [s]	Fraction [%]
4	264.8	93.0	35.1%
8	253.5	100.3	40.0%
40	868.9	318.5	36.7%

Table 2: Mean simulation times for 4 partitions and communication via TraCI in congestion conditions

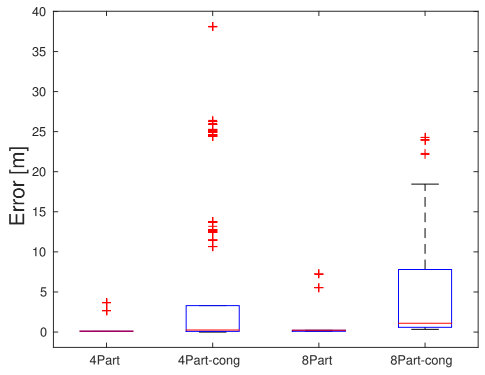


Figure 9: Precision using different partitions for a 40 boundary-edge scenario

a vehicle to be delayed or advanced one second can mean to skip or have a red light that was not present in the centralized simulation for the same vehicle. This error will be sustained until the vehicle finishes its trajectory, thus the reason of these outliers in the congestion scenarios.

4 Conclusions

In this paper, a method to parallelize the simulation of a scenario using SUMO was proposed. The results show that in terms of precision the method is acceptable but partitioning in a boundary with a traffic light will increase the error and this error is proportional to the amount of partitions with traffic lights or any other condition that is affected by a traffic light cycle. In terms of performance, a parallel simulation is slower than its centralized counterpart due to the synchronization process that is done via TraCI. When both simulations need a heavily usage of TraCI, parallel simulation is far superior to centralized. From these, it can be concluded that:

1. Although the process of partitioning a network is relatively easy to accomplish, there is not an algorithm to find an optimal partitioning of the network for the different possible configurations of a parallel simulation.

2. To accomplish a general better performance of the parallelized simulation over the centralized, and not only when TraCI is heavily used, it is necessary to synchronize and simulate the different partitions from within SUMO.

We suggest for future works to implement the strategy of synchronization directly into the source code of SUMO.

5 Acknowledgements

This work was supported by COLCIENCIAS project: Reducción de Emisiones Vehiculares Mediante el Modelado y Gestión Óptima de Tráfico en Áreas Metropolitanas - Caso Medellín - Área Metropolitana del Valle de Aburrá, código 111874558167, CT 049-2017. Universidad Nacional de Colombia. Proyecto HERMES 25374.

References

[1] A Acosta, J Espinosa, and J Espinosa. Distributed Simulation in SUMO Revisited: Strategies for Network Partitioning and Border Edges Management. In *Proceedings of the 4th SUMO User Conference*, pages 61–71, Berlin, 2016. Deutsches Zentrum für Luft- und Raumfahrt e.V.

[2] R Blokpoel. Interface between proprietary Controllers and SUMO. In *Proceedings of the 2nd SUMO User Conference*, pages 27–33, Berlin, 2014. Deutsches Zentrum für Luft- und Raumfahrt e.V.

[3] R Blokpoel. Network Conversion for SUMO Integration. In *Proceedings of the 2nd SUMO User Conference*, pages 35–44, Berlin, 2014. Deutsches Zentrum für Luft- und Raumfahrt e.V.

[4] Quentin Bragard, Anthony Ventresque, and Liam Murphy. dSUMO: towards a distributed SUMO - Simulation of Urban MObility. 2013.

[5] J Eising, F Fischer, J Flaig, O Florian, C Fritze, K Indych, M Montenegro, S Müller, M Schneider, N Vieregg, A Wegge, H Döbler, F Koller, O Menzel, S Dietzel, and B Scheuermann. PARCOURS: A SUMO-Integrated 3D Driving Simulator for Behavioral Studies. In *Proceedings of the 4th SUMO User Conference*, pages 135–146, Berlin, 2016. Deutsches Zentrum für Luft - und Raumfahrt e.V.

[6] P Heywood, S Maddock, J Casas, D Garcia, M Brackstone, and P Richmond. Data-parallel agent-based microscopic road network simulation using graphics processing units. *Simulation Modelling Practice and Theory*, 1(13), 2017.

[7] J Jin and X Ma. Stochastic optimization of signal plan and coordination using parallelized traffic simulation. In *Proceedings of the 2nd SUMO User Conference*, Berlin, 2014. Deutsches Zentrum für Luft - und Raumfahrt e.V.

[8] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.