

Road Network Partitioning with Distributed Threaded-Parallelism for Large-Scale SUMO Traffic Simulation

Vicente R. Tomás

`vtomas@uji.es`

Universitat Jaume I

Iván Monzón-Catalán

Universitat Jaume I

Maribel Castillo

Universitat Jaume I

Luis A. García

Universitat Jaume I

Research Article

Keywords: traffic simulation, thread-level parallelism, distributed computing, parallel simulation

Posted Date: October 14th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11161759/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Road Network Partitioning with Distributed Threaded-Parallelism for Large-Scale SUMO Traffic Simulation

Vicente R. Tomás^{1*}, Iván Monzón-Catalán¹, Maribel Castillo¹,
Luís A. García¹

^{1*}Dept. de Engineering and Computer Science, University Jaume I,
Avda. Sos Baynat, s/n, Castelló de la Plana, 12071, Castelló, Spain.

*Corresponding author(s). E-mail(s): vtomas@uji.es;

Contributing authors: imonzon@uji.es; castillo@uji.es; garcial@uji.es;

Abstract

Large-scale SUMO traffic simulation can require substantial computational resources when realistic road networks, high traffic demand, dynamic routing, and computationally intensive external processing are considered. One approach to reduce this cost is to spatially decompose the road network and process the resulting simulation domains concurrently, while also exploiting parallelism within each domain. This work follows this strategy through a two-level parallel architecture that combines thread-level parallelism enabled by free-threaded Python with distributed execution across multiple computing nodes. A central master coordinates the domain workers, inter-domain vehicle transfers, and global synchronization. The architecture is evaluated on a large-scale Rotterdam road network under multiple traffic-demand levels, accident and no-accident conditions, and different spatial decompositions. At the single-node level, the six-thread configuration achieves speedups from $3.88\times$ to $5.77\times$ over the sequential baseline. Distributed execution further increases performance, with the best configurations reaching speedups between $9.08\times$ and $13.32\times$, corresponding to an additional $1.57\times$ – $2.45\times$ acceleration over six-thread single-node execution.

Traffic-volume validation shows that all six evaluated validation configurations satisfy the predefined equivalence criterion, while all inter-domain vehicle transfers are completed without failures. The results also show that spatial decomposition affects transfer activity and distributed performance, whereas the increasing cost of global synchronization becomes the main observed limitation to further scalability.

Keywords: traffic simulation; thread-level parallelism; distributed computing; parallel simulation

1 Introduction

The increasing complexity and dynamism of modern traffic systems, driven by Connected, Cooperative, and Automated Mobility (CCAM) and the high volume of road freight transport associated with large-scale logistics operations, have significantly increased the computational demands on decision support systems designed to assist traffic managers in making timely and informed decisions.

Among the tools available to support traffic managers, traffic simulation tools play a key role, providing a controlled environment for evaluating traffic-management strategies, incident-response measures, dynamic routing policies, infrastructure modifications, and variations in traffic demand. Traffic simulators such as SUMO (Simulation of Urban MObility) [1] are widely adopted because of their flexibility and ability to represent traffic behaviour at a high level of detail.

The computational cost of microscopic simulation becomes particularly relevant in *what-if* analyses, in which traffic operators and researchers must evaluate multiple alternatives, such as accident-response strategies, routing policies, traffic-control configurations, or emission levels under different demand conditions. These studies commonly require repeated simulations over multiple scenarios and parameter configurations. Consequently, reducing execution time is important not only for processing larger road networks and vehicle populations, but also for enabling iterative experimentation and supporting decisions under time constraints [2].

Parallel computing provides different strategies for reducing these computational demands. Within a single machine, parallel execution can exploit multiple CPU cores while avoiding external network communication, although its scalability remains limited by the resources available on that machine. Distributed execution can extend the available computational resources across several cooperating machines, but introduces additional communication, synchronization, and workload-distribution costs [3–5].

These general scalability challenges are particularly relevant for SUMO. Although SUMO supports multithreaded execution for selected operations, especially route computation, its microscopic simulation core remains predominantly sequential. According to the official SUMO documentation, the parallelization of the microscopic core is still under development and does not yet provide meaningful speedup in general. Moreover, SUMO does not natively support the distributed execution of a single simulation across multiple computing nodes [6]. Exploiting distributed-memory systems therefore requires an external architecture capable of coordinating multiple SUMO executions while preserving the consistency of the overall simulation.

At the single-node level, our previous work addressed parallelism through a process-based agent architecture built around SUMO [7]. Computationally demanding external operations, including emission estimation, dynamic routing, and large-scale vehicle-data retrieval, were distributed among several concurrent processes without modifying the internal SUMO simulation engine. The proposed approach reduced the execution

time of the evaluated scenarios, but remained restricted to the computational resources of a single node and required process-level coordination and data exchange.

The present work extends our previous process-based framework in two complementary directions. First, the process-based single-node layer is redesigned to use a thread-based execution model enabled by free-threaded Python. Unlike the previous worker processes, the new local workers are implemented as threads and execute within the same address space and can directly access common data structures, reducing part of the worker-management and data-exchange overhead. Second, the framework is extended beyond a single host by spatially decomposing the road network into multiple simulation domains, each assigned to a domain worker.

Together, these extensions result in a two-level parallel architecture in which each domain worker combines an independent SUMO instance with a local parallel execution layer composed of local workers. At the distributed level, a central master coordinates the domain workers, the inter-domain vehicle transfers, and the progress of the overall simulation. Vehicle state and routing information are exchanged through persistent TCP connections when vehicles cross domain boundaries. The scalability of this architecture therefore depends on whether the reduction in local computation obtained through network decomposition compensates for the communication, synchronization, and possible workload-distribution costs introduced by distributed execution [8].

The experimental evaluation first compares the sequential, process-based, and thread-based single-node implementations to identify the best-performing local configuration. This configuration is subsequently used within each domain worker to evaluate the scalability of the complete distributed framework. Experiments are performed on a real-world road network under different traffic intensities and under both accident-free and accident conditions. The study assesses traffic-flow preservation against the original non-partitioned SUMO simulation and analyses execution time, speedup, inter-domain vehicle transfers, and synchronization overhead.

The main contributions of this work are as follows:

- A thread-based redesign of the process-based single-node parallel architecture. The new implementation replaces local worker processes with threads operating within a shared address space and provides the local parallel layer used in the distributed implementation evaluated in this work.
- A two-level parallel architecture that combines single-node parallelism with distributed execution across multiple computing nodes. The road network is spatially decomposed among multiple domain workers, each combining an independent SUMO instance with a local parallel execution layer that can be implemented using either the process-based or thread-based local execution model.
- A TCP-based coordination and vehicle-transfer mechanism that preserves vehicle state and routing information when vehicles cross domain boundaries and maintains the consistent progress of the distributed simulation.
- An experimental comparison of sequential, process-based, and thread-based single-node execution, followed by a traffic-flow preservation and distributed scalability evaluation under different traffic intensities, accident conditions, domain counts, and decomposition orientations.

The remainder of this paper is organized as follows. Section 2 reviews previous work on parallel and distributed traffic simulation and road-network partitioning. Section 3 describes the proposed two-level parallel architecture, including the single-node parallel layer, spatial decomposition of the road network, inter-domain vehicle transfers, communication, synchronization, and the distributed execution workflow. Section 4 presents the Rotterdam test-bed, traffic-demand and accident scenarios, spatial decomposition strategies, and computing environment. Section 5 describes the experimental methodology and the configurations used to evaluate single-node and distributed execution. Section 6 presents and discusses the traffic-validation, single-node performance, and distributed scalability results. Finally, Section 7 summarizes the main findings and outlines future research directions.

2 State of the Art

Traffic simulation models vary widely depending on their purpose and the level of detail they provide. In general, traffic simulators can be classified into macroscopic, microscopic, and mesoscopic approaches. Macroscopic models represent traffic collectively through aggregated variables such as density, flow, and average speed, and are therefore well suited to large-scale network analysis and strategic planning. Microscopic simulators, in contrast, reproduce the behavior, interactions, and movements of individual vehicles, providing the level of detail required for operational traffic management, vehicle-level decision making, and CCAM applications. Mesoscopic approaches provide an intermediate representation, combining individual vehicle modeling with aggregated traffic-flow dynamics in order to balance computational efficiency and behavioral detail [9].

Among microscopic traffic simulators, the open-source Simulation of Urban Mobility (SUMO) framework has received considerable attention due to its flexibility, modularity, and extensive customization capabilities [1]. SUMO is widely used for the development and evaluation of Intelligent Transportation Systems (ITS), vehicle-to-vehicle and vehicle-to-infrastructure communication, dynamic routing strategies, traffic management algorithms, and emission-estimation frameworks [10]. However, the main simulation workflow remains largely sequential, which limits scalability as the size of the road network, the number of simulated vehicles, and the amount of external processing increase. This limitation becomes particularly relevant when SUMO is coupled through the Traffic Control Interface (TraCI) with external agents that perform computationally intensive operations at every simulation step, such as state evaluation, dynamic routing, emission estimation, or communication-related processing [11]. Large realistic scenarios therefore create a tension between the microscopic level of detail required by advanced ITS and CCAM applications and the computational time required to obtain that level of detail.

Several approaches have been proposed to reduce this computational cost without abandoning microscopic traffic simulation. In [12] is proposed a multi-scale co-simulation approach that combines microscopic and mesoscopic SUMO simulation modes. The region surrounding an EGO vehicle is simulated microscopically while the remaining network is represented at a lower level of detail, producing speedups between

approximately $3\times$ and $10\times$ in the evaluated large urban scenarios. This technique is particularly suited to scenarios in which detailed microscopic information is required mainly around a specific EGO vehicle, while traffic outside its vicinity can be represented at a lower level of detail. Other work has investigated workload imbalance and stragglers in distributed spatial simulations. The approach presented in [13] proposes dynamic straggler-mitigation strategies combining flexible synchronization, decentralized dynamic load balancing, and on-demand resource allocation, evaluated using microscopic traffic simulation. Although the work is not SUMO-specific, it highlights the importance of workload balance and synchronization in distributed microscopic traffic simulation.

An alternative line of research has focused on parallelizing the computational work performed around SUMO while leaving its simulation core unmodified. In our previous work [7], a non-intrusive task-level parallelization architecture was proposed for SUMO-based multi-agent traffic simulation. Computationally intensive vehicle- and segment-level operations, including traffic-state evaluation, emission estimation, and dynamic route calculation, were distributed across a pool of worker processes implemented through Python’s `ProcessPoolExecutor`. SUMO remained responsible for the microscopic evolution of the complete traffic network and advanced only after the external parallel operations had completed, using an explicit synchronization point at every simulation step. The framework was evaluated on two real-world interurban networks, including the Rotterdam motorway network, and achieved speedups of up to approximately $4\times$ over sequential execution. However, the architecture remained confined to a single computing node and relied on multiple worker processes, each operating in an independent address space and therefore requiring data to be exchanged between the coordinator and the workers.

The choice of multiprocessing in our previous architecture was largely motivated by the Global Interpreter Lock (GIL), which traditionally prevented multiple Python threads from executing Python bytecode concurrently and therefore limited the applicability of thread-based parallelism to CPU-intensive workloads. The introduction of free-threaded Python changes this execution model by allowing the interpreter to operate without the conventional GIL restriction, making it possible to reconsider thread-level parallelism for workloads that previously required multiple processes. A preliminary version of this process-to-thread redesign was presented as a conference poster [14]. The present work provides the complete implementation and experimental evaluation of this single-node thread-based execution model, which is subsequently compared with the original process-based implementation to determine the local configuration used in the distributed evaluation.

Another line of research relies on spatial decomposition of the road network so that multiple SUMO instances can process different simulation domains concurrently. SParTSim [3] introduced an early distributed SUMO architecture in which the road network is partitioned into several sub-networks, each processed by an independent SUMO instance. This research line was subsequently continued with dSUMO [15] and later with a decentralized self-balancing architecture [4].

QarSUMO [5] also adopts spatial decomposition, assigning each network partition to a separate QarSUMO process that runs an independent sequential SUMO instance.

Communication and synchronization between partitions are implemented through OpenMPI, including the exchange of vehicle state across border edges. Although its experimental evaluation focuses on parallel execution within a single multicore server, its MPI-based design can be extended to multi-node execution. QarSUMO additionally introduces a congestion-specific optimization that modifies parts of the SUMO simulation engine to reduce computation for consecutive slow-moving vehicles.

These approaches illustrate the potential of spatial decomposition for increasing the scalability of microscopic traffic simulation.

The performance of these spatially partitioned approaches depends strongly on how the road network is partitioned. In [16], traffic-related characteristics such as signalized intersections, road-segment length, traffic density, and lane count are incorporated into the partitioning process. The authors also outline a distributed SUMO framework in which a master controller based on TraCI and MPI coordinates the resulting sub-networks. In [17] propose a parallel framework based on spectral partitioning, transforming the road network into a graph and using the eigenstructure of its Laplacian matrix to obtain balanced sub-networks while limiting communication between partitions. This formulation explicitly recognizes that distributed performance depends not only on the number of partitions but also on the number and location of connections between them. More recently, [18] propose a dynamic partitioning framework based on KLeiden community detection, where the selection of the network decomposition is formulated as an optimization problem and can be adapted as origin–destination demand evolves during the simulation.

Road-network partitioning has also been studied independently from the runtime architecture used to execute the resulting sub-networks. The benchmark presented in [8] systematically compares static road-network division methods according to criteria such as workload balance, the number of border edges, and partitioning time. This type of analysis is important because increasing the number of partitions generally reduces the amount of network processed by each worker but simultaneously increases the number of boundaries at which inter-partition interactions may occur. However, a balanced static decomposition of the road graph does not necessarily imply a balanced distributed execution. The actual computational workload evolves with traffic demand and is determined by the spatial and temporal distribution of active vehicles, congestion, rerouting, and other dynamic traffic phenomena. Consequently, partitions with a similar number of road elements may exhibit substantially different runtime loads during the simulation.

The existing literature therefore shows that SUMO-based approaches have generally addressed these two levels of parallelism separately: either by parallelizing the external computation associated with a single SUMO instance or by spatially decomposing the road network across multiple simulator instances. Although shared-memory and distributed parallelism have been combined in other microscopic traffic simulators [19], their integration within a SUMO-based execution framework remains comparatively unexplored.

The present work addresses this gap by combining a thread-based local parallel layer with distributed execution across spatially decomposed simulation domains. Rather than proposing a new graph-partitioning algorithm, the study focuses on how

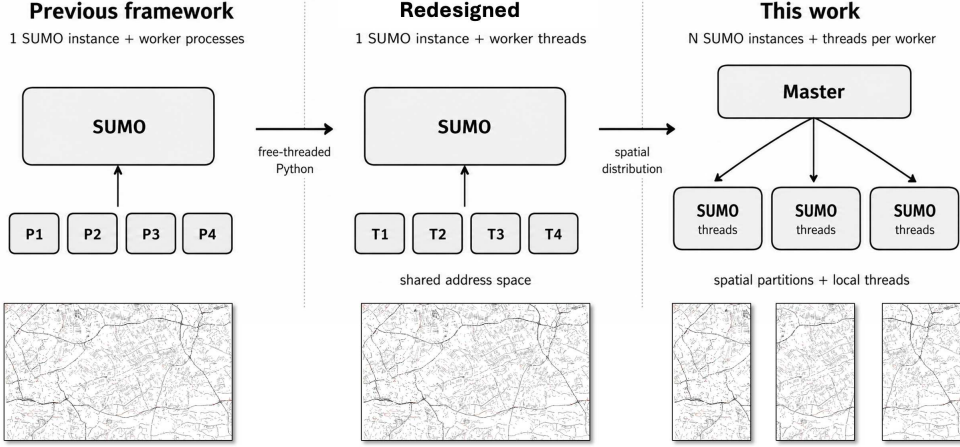


Fig. 1: Evolution of the parallel simulation framework from process-based single-node execution to thread-based single-node execution and, finally, to distributed execution with multiple domain workers coordinated by a master.

alternative spatial decomposition orientations affect inter-domain vehicle transfers, traffic-volume preservation, and distributed scalability.

3 System Architecture and Execution Model

The proposed framework follows a two-level parallel architecture. At the single-node level, computationally intensive operations associated with one SUMO instance are executed concurrently through a local parallel layer. This layer can be implemented using either local worker processes or local worker threads, as described in Section 3.1. At the distributed level, the road network is spatially decomposed into multiple simulation domains, each assigned to a domain worker combining an independent SUMO instance with a local parallel execution layer.

Figure 1 summarizes the evolution from the original process-based implementation to the thread-based single-node redesign and finally to the distributed architecture evaluated in this work. In all three designs, SUMO remains responsible for the traffic simulation and its internal simulation engine is left unmodified. The main difference lies in the execution model used around SUMO and, in the distributed architecture, in the use of multiple coordinated simulator instances instead of a single one.

3.1 Single-Node Parallel Architecture

At the single-node level, parallelism is introduced outside the SUMO simulation core through the external agent layer and its interaction with SUMO via TraCI. Computationally intensive vehicle- and segment-level operations, including traffic-state evaluation, emission estimation, dynamic route computation, and other agent-level processing, can be executed concurrently once the information required for the current

simulation step has been obtained. SUMO itself remains unmodified and is responsible for the evolution of the traffic network.

In the implementation [7], these parallel tasks were distributed through Python’s `ProcessPoolExecutor`. At each simulation step, the coordinator retrieves the information required by the external agents, constructs the corresponding tasks, and distributes them among a pool of worker processes. The simulation advances only after the tasks associated with the current step have completed and the required results have been collected. This execution model can therefore be represented as one SUMO instance plus k local worker processes.

The thread-based redesign preserves the same simulation workflow but replaces the pool of independent processes with a `ThreadPoolExecutor` running on free-threaded Python. The local workers execute as threads within the same Python process and consequently operate over a common address space. Simulation-state and agent-related data can therefore be shared among the worker threads instead of being maintained in separate process memories. The resulting execution model can be represented as one SUMO instance plus k local worker threads.

These two execution models therefore define alternative implementations of the local parallel layer used by the distributed architecture described next.

3.2 Distributed Architecture

The distributed architecture introduces execution across multiple computing nodes. To distribute the simulation among these nodes, the original SUMO road network is spatially decomposed into N simulation domains. Each simulation domain is assigned to one domain worker running an independent SUMO instance. Each domain worker also incorporates the local parallel execution layer described in Section 3.1.

The distributed organization is independent of whether the local layer is implemented using worker processes or worker threads; both execution models described in Section 3.1 are compatible with the architecture.

The spatial decomposition is defined at the level of SUMO edges. A SUMO edge represents a directed road segment connecting two junctions. Let E denote the set of edges of the original road network and let $W = \{w_1, \dots, w_N\}$ denote the set of domain workers. The decomposition partitions E into N disjoint domains such that

$$E = \bigcup_{i=1}^N D_i, \quad D_i \cap D_j = \emptyset \quad \text{for } i \neq j.$$

Each domain D_i is assigned to one domain worker w_i . Equivalently, edge ownership is represented by the mapping

$$f : E \rightarrow W, \quad f(e) = w_i \quad \text{if } e \in D_i.$$

A *domain worker* therefore denotes the distributed entity responsible for one simulation domain and its associated SUMO instance, whereas a *local worker* denotes a process or thread executing parallel operations within that domain worker.

Thus, every SUMO edge is associated with exactly one simulation domain and domain worker. The decomposition is established before execution begins and remains fixed throughout the simulation.

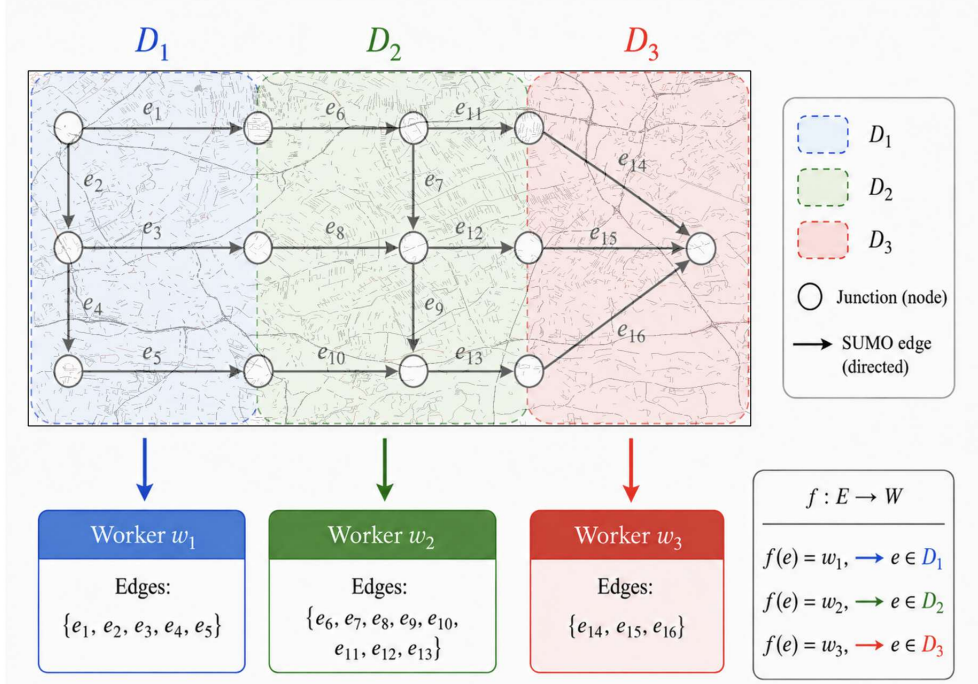


Fig. 2: Example of spatial decomposition and edge-to-domain-worker mapping. Each simulation domain is assigned to one domain worker, and every SUMO edge belongs to exactly one domain according to the mapping $f: E \rightarrow W$.

Figure 2 illustrates this organization using a simplified road network partitioned into three simulation domains. The directed edges belonging to each domain are assigned to the corresponding domain worker, providing the edge-to-domain-worker mapping used throughout the distributed execution.

Each domain worker is responsible for advancing the traffic state within its assigned domain. A central master coordinates the distributed execution but does not simulate an additional traffic domain. Instead, it maintains the global ownership mapping required to coordinate the independent SUMO instances. When the continuation of a vehicle route requires an edge belonging to another domain, this mapping allows the master to identify the corresponding destination domain worker.

The architecture therefore separates local traffic simulation from global coordination. The mechanisms used to transfer vehicle state between domain workers and to maintain a consistent simulation time across the independent SUMO instances are described in Section 3.3.

3.3 Inter-Domain Communication and Synchronization

Spatial decomposition requires explicit state transfer when a vehicle leaves the simulation domain managed by one domain worker and continues its route on an edge

assigned to another domain. In a non-partitioned SUMO execution, this transition is handled internally by the same simulator instance. In the distributed architecture, however, the vehicle state must be transferred from the source domain worker to the destination domain worker. We refer to this operation as *inter-domain vehicle transfer*.

When a domain worker detects that the continuation of a vehicle route belongs to another domain, it extracts the information required to continue the simulation in the destination SUMO instance. The transferred state includes the vehicle identifier, route information, current position, and speed. The transfer follows the centralized path

$$w_s \rightarrow M \rightarrow w_d,$$

where w_s and w_d denote the source and destination domain workers, respectively, and M denotes the master. The destination domain worker is identified using the ownership mapping defined in Section 3.2. The master forwards the vehicle state to the corresponding domain worker, which incorporates it into its local SUMO instance. This centralized organization avoids requiring direct connections between every pair of domain workers and keeps the information required to route inter-domain transfers at the master.

Communication between the master and the domain workers is implemented through persistent TCP connections. Since the exchanged information directly affects the subsequent evolution of the distributed simulation, reliable and ordered delivery is required. TCP provides these properties [20], avoiding the need to implement equivalent retransmission and message-ordering mechanisms in the simulation layer.

Reliable state transfer alone is not sufficient to preserve a consistent distributed simulation. The independent SUMO instances must also advance according to a common logical simulation time. The framework therefore uses an explicit lockstep synchronization scheme. All domain workers must complete the operations associated with timestep t , including any required inter-domain vehicle transfers, before the synchronization barrier is released and execution proceeds to timestep $t + 1$. This ensures that state generated during one timestep is incorporated into the appropriate destination domain before the following timestep begins.

3.4 Distributed Execution Workflow

Figure 3 summarizes the complete distributed execution workflow. The upper part illustrates the inter-domain vehicle-transfer path through the master, whereas the lower part shows how local execution, vehicle transfers, and global synchronization are integrated within each simulation timestep. The mechanisms introduced in the previous sections are integrated into the distributed execution workflow shown in

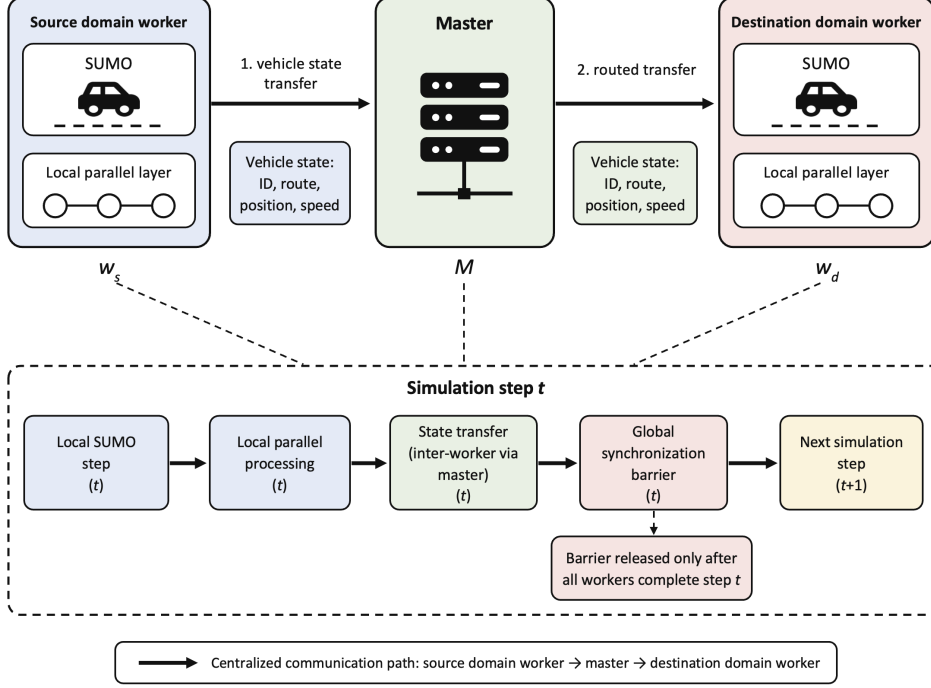


Fig. 3: Distributed execution workflow. Vehicle state is transferred from the source domain worker to the destination domain worker through the master, while all domain workers synchronize before advancing to the next simulation timestep.

Algorithm 1 formalizes this workflow and makes explicit the order in which these operations are performed. At each timestep, domain workers advance their local SUMO instances and execute the local parallel operations; vehicles requiring a domain change are then transferred through the master and incorporated into the corresponding destination domain before the global synchronization barrier is reached. The algorithm is independent of the particular implementation of the local parallel layer, which can use either of the execution models described in Section 3.1.

This execution cycle is repeated until the predefined simulation horizon is reached.

4 Test-bed and Experimental Setup

The experimental test-bed is based on a large-scale interurban road network located in the Rotterdam region of the Netherlands. The scenario was selected because it combines a dense motorway infrastructure, peri-urban and non urban, and major freight corridors connecting the Port of Rotterdam with the Dutch national road network and the European hinterland. These characteristics provide a suitable large-scale scenario for evaluating the proposed distributed architecture under different

Algorithm 1 Distributed simulation execution workflow

Require: Simulation domains $D_1, \dots, D_N$, domain workers $W = \{w_1, \dots, w_N\}$, master M , and edge-to-domain-worker mapping $f : E \rightarrow W$

- 1: Start one SUMO instance for each simulation domain D_i
- 2: Establish persistent TCP connections between M and all w_i
- 3: **for** each simulation timestep t **do**
- 4: **parallel for each** $w_i \in W$
- 5: Advance the local SUMO instance by one simulation step
- 6: Execute the local parallel operations
- 7: **for all** vehicles v with a next route edge **do**
- 8: Obtain the next route edge e_{next}
- 9: **if** $f(e_{\text{next}}) \neq w_i$ **then**
- 10: Extract vehicle state $s_v \leftarrow \langle id_v, route_v, position_v, speed_v \rangle$
- 11: Send (e_{next}, s_v) to M
- 12: **end if**
- 13: **end for**
- 14: **end parallel for**
- 15: **for all** vehicle transfers (e_{next}, s_v) received by M **do**
- 16: Determine the destination domain worker $w_d \leftarrow f(e_{\text{next}})$
- 17: Forward s_v to w_d
- 18: **end for**
- 19: **parallel for each** $w_i \in W$
- 20: **for all** vehicle states s_v received from M **do**
- 21: Incorporate s_v into the local SUMO instance
- 22: **end for**
- 23: **end parallel for**
- 24: Synchronize all domain workers at the global barrier
- 25: **end for**

traffic intensities and spatial decompositions. The same underlying network, traffic-generation procedure, vehicle behavior, and simulation parameters are maintained throughout the experimental study, while the traffic demand, incident condition, and spatial decomposition into simulation domains are systematically varied.

4.1 Rotterdam Road Network

The simulated road network represents the main motorway and supporting road infrastructure surrounding Rotterdam. This region contains several of the principal corridors connecting the Port of Rotterdam with the rest of the Netherlands and with international freight routes towards Belgium and Germany. The modeled infrastructure includes, among others, the A-15, A-27, A-2, A-12 and A-20, which are important motorway corridors in the Netherlands.

The A15 constitutes one of the main east–west freight corridors of the region and connects the Rotterdam port and industrial area with the European hinterland [21]. Towards the east, the A15–A12 connection provides continuity from the Rotterdam

area towards Germany [22]. The A20 represents another major axis of the network, connecting Rotterdam with the central Netherlands and providing an important connection towards Utrecht [23]. The resulting scenario therefore contains both strong longitudinal traffic corridors and transverse connections between the main motorway axes.

The road network was originally extracted from OpenStreetMap and processed before being imported into SUMO. The preprocessing stage retains the motorway infrastructure together with the secondary road elements required to maintain connectivity between the main corridors. After this initial extraction, a strongly connected component filtering stage is applied to remove disconnected and dangling elements that cannot participate in valid routes across the simulated network. The final SUMO model contains 22,332 edges. These edges constitute the set E subsequently decomposed into the simulation domains defined in Section 3.2. Of these, 2,739 correspond to the main motorway infrastructure and 19,593 to the supporting secondary road network. The complete modeled area extends approximately 67×42 km. Figure 4 shows the road network used in the experimental study.

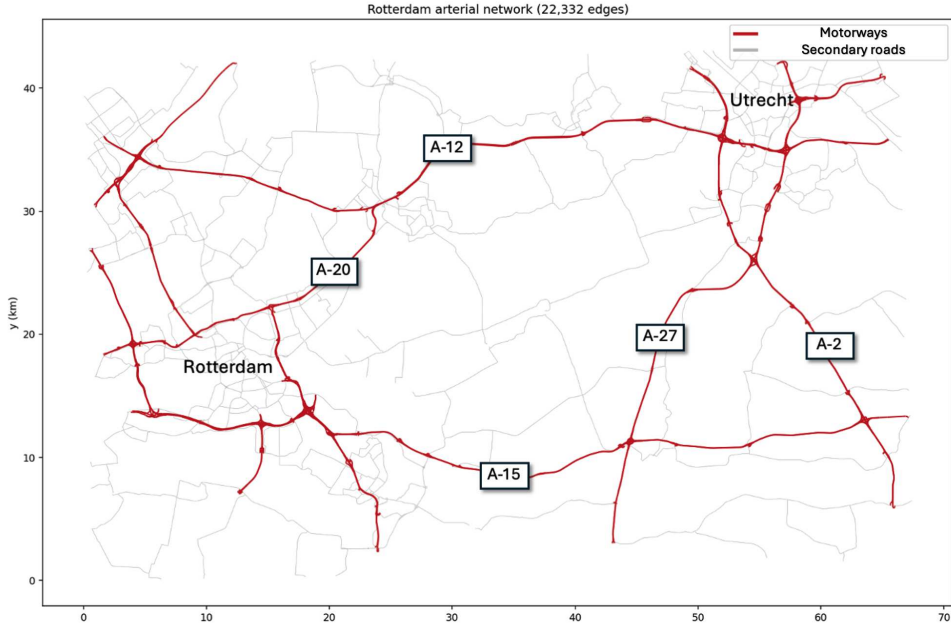


Fig. 4: Rotterdam road network used in the experimental test-bed. The model contains the principal motorway corridors together with the supporting secondary road infrastructure retained in the SUMO network.

The main characteristics of the resulting network are summarized in Table 1.

Characteristic	Rotterdam network
Region	Rotterdam, Netherlands
Main motorway axes	A15, A27, A2, A12, A20
Approximate extent	67×42 km
Motorway edges	2,739
Secondary road edges	19,593
Total SUMO edges	22,332
Maximum motorway speed	120 km/h

Table 1: Main characteristics of the Rotterdam road-network model.

4.2 Traffic Demand and Vehicle Configuration

Three traffic-demand levels are considered in the experimental study. The configured vehicle populations are $V \in \{15,000, 20,000, 30,000\}$ representing low, valley and peak traffic density according to [24]. These values correspond to the number of vehicles introduced as traffic demand and should not be interpreted as the number of vehicles that necessarily complete their trip before the end of the simulation. Since all experiments use a fixed simulation horizon, vehicles can remain active within the network when the final simulation step is reached.

For a given traffic-demand level, equivalent experimental configurations use the same vehicle population, origin–destination definition, departure schedule, routing parameters, and simulation horizon. Consequently, modifying the spatial decomposition does not modify the traffic input supplied to the simulator.

The simulated traffic represents the heterogeneous composition of an interurban network strongly influenced by the activity of the Port of Rotterdam. The traffic model therefore includes both passenger vehicles and heavy goods vehicles, which are particularly relevant along the main motorway corridors [7].

Vehicles are initially assigned a valid route between their origin and destination. Vehicles configured for dynamic routing can reconsider their itinerary during the simulation according to the available traffic state. This routing mechanism remains active under all evaluated spatial decompositions and is also used in the incident scenarios described below.

4.3 Normal and Accident Scenarios

Each of the three traffic-demand levels is evaluated under two traffic conditions: a reference condition without incidents and a condition in which an accident is introduced into the network. The resulting experimental scenarios are therefore 15k-N, 20k-N, 30k-N, 15k-A, 20k-A, and 30k-A, where N denotes a no-accident scenario and A denotes the corresponding accident scenario as shown in Table 2.

The incident is introduced on the A15 motorway in the direction away from Rotterdam, on a two-lane road section. During the incident period, one lane is completely closed to traffic, while the maximum allowed speed on the remaining lane is reduced

Scenario	Configured vehicles	Traffic condition
15k-N	15,000	No accident
20k-N	20,000	No accident
30k-N	30,000	No accident
15k-A	15,000	Accident
20k-A	20,000	Accident
30k-A	30,000	Accident

Table 2: Traffic-demand and incident scenarios included in the experimental test-bed.

by 50%. This configuration creates a severe local bottleneck and substantially reduces the effective capacity of the affected section.

The incident is configured to activate the dynamic-routing mechanism and induce vehicles to seek alternative routes through the surrounding motorway network, particularly via the A20, A12, and A27 corridors. This allows the distributed simulation to be evaluated under a traffic condition in which vehicle trajectories and the spatial distribution of traffic differ substantially from the corresponding no-accident scenario.

All simulations use a step length of one second and a fixed horizon of 14,400 simulation steps, corresponding to four hours of simulated time. The accident begins at simulation step 7,200 and remains active until step 10,800. The total incident duration is therefore 3,600 s.

The same accident location, duration, and configuration are maintained for the three traffic-demand levels. The main temporal parameters are summarized in Table 3.

Parameter	Value	Unit
Simulation step length	1	s
Simulation horizon	14,400	steps
Departure distribution	Uniform	–
Accident start	7,200	step
Accident end	10,800	step
Accident duration	3,600	s
Maximum motorway speed	120	km/h
TraCI command frequency	Every step	–

Table 3: Main SUMO and traffic-scenario parameters.

4.4 Spatial Decomposition of the Rotterdam Network

The static spatial decomposition introduced in Section 3 is instantiated for the Rotterdam road network using two complementary decomposition strategies. The first is a Traffic Control Centre (TCC)-oriented decomposition, while the second is a corridor-oriented decomposition. For conciseness, these strategies are hereafter denoted as X-oriented and Y-oriented, respectively.

The X-oriented decomposition provides a west–east spatial division that is consistent with the geographical organization of traffic management in the study area, whereas the Y-oriented decomposition provides a complementary south–north division that intersects the main long-distance motorway and freight corridors.

In both cases, the SUMO edges of the original network are divided into N disjoint simulation domains, with each domain assigned to one domain worker. The resulting decomposition remains fixed throughout each simulation run.

The two orientations differ in the spatial direction used to divide the road network. The X-oriented decomposition produces predominantly vertical domains, whereas the Y-oriented decomposition produces predominantly horizontal domains. Consequently, the two decompositions place domain boundaries differently with respect to the road network and vehicle trajectories, allowing the sensitivity of the distributed architecture to the spatial decomposition geometry to be evaluated.

For both orientations, the number of simulation domains is varied as

$$N \in \{2, 3, 4, 6, 8, 10, 12\},$$

while the underlying road network, traffic demand, and simulation parameters remain unchanged. Therefore, changing N controls the degree of spatial fragmentation, whereas changing the decomposition orientation modifies the geometry and location of the boundaries between simulation domains.

Figure 5 illustrates the two decomposition orientations using the four-domain configurations.

X-oriented decomposition.

The X-oriented decomposition divides the network into predominantly vertical spatial domains. The domains are indexed according to their position along the x coordinate, producing a west–east progression across the Rotterdam study area. This orientation is consistent with the operational structure of road traffic management in the region. Dutch motorway traffic is coordinated nationally by Verkeerscentrum Nederland (VCNL), together with five regional traffic centres operated by Rijkswaterstaat [24, 25]. Within the geographical area represented by the Rotterdam test bed, the most relevant regional centres are VC Zuidwest-Nederland, located in Rhon, and VC Midden-Nederland, located in Utrecht [26, 27]. These centres provide geographical reference points for the western and eastern parts of the modeled network, respectively, with the Gouda/A12 area lying between them.

The X-oriented decomposition follows this geographical organization. At low domain counts, a domain covers a relatively large portion of the network, whereas increasing N progressively subdivides the same west–east structure into smaller simulation domains.

For example, the four-domain X-oriented decomposition shown in Figure 5(a) provides a coarse west–east division of the network into four simulation domains.

The purpose of this decomposition is not to reproduce the legal or administrative boundaries of the Rijkswaterstaat traffic centres. Rather, it provides a reproducible spatial decomposition whose orientation is consistent with the regional traffic-management structure of the study area.

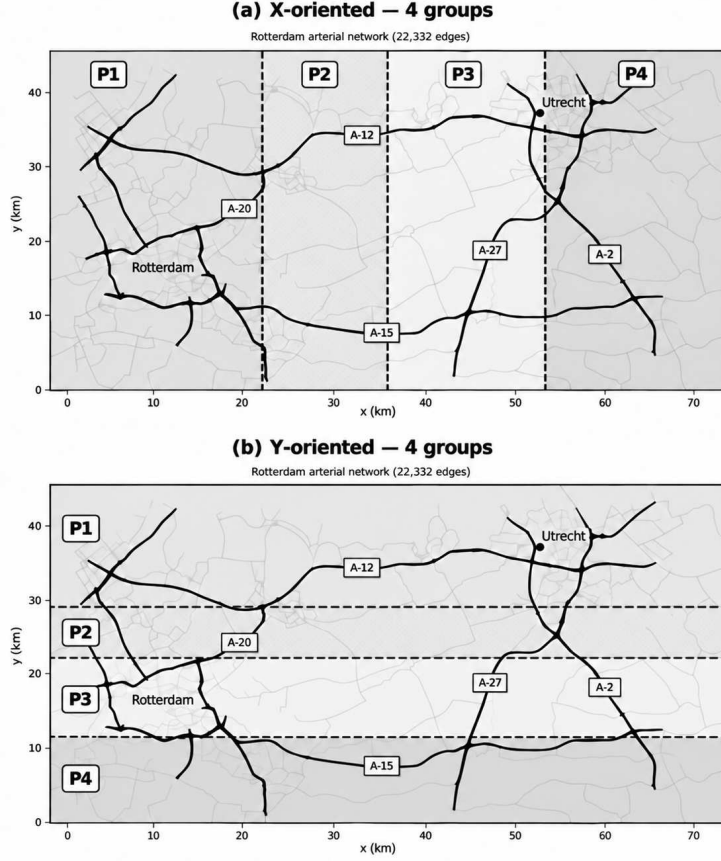


Fig. 5: Spatial decompositions of the Rotterdam network. Panel (a) illustrates the X-oriented decomposition for four simulation domains, whereas panel (b) shows the corresponding Y-oriented decomposition. The four-domain case is included for illustration purposes; the experimental evaluation also considers $N = 2, 3, 6, 8, 10$, and 12 .

This makes the X-oriented decomposition particularly useful for evaluating how inter-domain vehicle transfers across neighboring simulation domains can also be interpreted from an operational traffic-management perspective.

Y-oriented decomposition.

The Y-oriented decomposition divides the network into predominantly horizontal spatial domains. The domains are indexed according to their position along the y coordinate, producing a south–north progression across the Rotterdam study area.

Unlike the X-oriented decomposition, these domains are not intended to reflect the geographical organization of regional traffic management. Instead, this orientation provides a complementary spatial decomposition for studying long-distance motorway and freight movements across the Rotterdam region.

Rotterdam constitutes one of the principal European freight gateways, and the modeled network connects the port and metropolitan area with major Dutch motorway corridors and, through them, with the wider European transport network. This interpretation is consistent with the role of road links connecting ports, urban nodes and international transport corridors within the Trans-European Transport Network (TEN-T) [28].

The Y-oriented decomposition therefore creates horizontal bands that intersect different parts of the long-distance road system. In the modeled area, these bands intersect important motorway structures such as the A12, A20, A15 and A16 corridors, which connect Rotterdam with the Randstad, the inland Dutch network and international freight directions.

For example, the four-domain Y-oriented decomposition shown in Figure 5(b) provides a relatively coarse division of the network into four horizontal simulation domains.

This orientation provides a deliberately different distributed execution pattern from the X-oriented decomposition. A long-distance vehicle trajectory may remain within the same simulation domain for a significant part of its path or cross several horizontal domains, depending on its direction. Consequently, Y-oriented is useful for evaluating the sensitivity of the distributed architecture to the spatial fragmentation of long-distance traffic corridors.

4.5 Computing Environment

The simulations are executed using SUMO 1.27.0, while the parallel framework uses Python 3.14 with its free-threaded runtime. The computing machines are equipped with AMD Ryzen 5 5500 processors, six physical CPU cores, and approximately 94 GiB of main memory, and run Ubuntu 22.04.3 LTS. Simultaneous multithreading is disabled. The machines participating in the distributed experiments are interconnected through a Gigabit Ethernet local network.

During distributed execution, each domain worker is assigned to a separate computing machine and has access to the six physical CPU cores of that machine. The master is executed on an additional dedicated machine and is responsible exclusively for coordination, inter-domain vehicle transfers, and global synchronization. Consequently, a distributed configuration with N simulation domains uses N worker machines plus one additional machine for the master.

The main software and hardware characteristics are summarized in Table 4.

5 Experimental Methodology

This section describes the methodology used to conduct the experimental evaluation, including the common experimental procedure and the configurations used to evaluate single-node and distributed parallelism.

Component	Configuration
CPU	AMD Ryzen 5 5500
physical CPU cores per machine	6
SMT	Disabled
Main memory	~94 GiB
Operating system	Ubuntu 22.04.3 LTS
Python	3.14, free-threaded build
SUMO	1.27.0
Interconnection	Gigabit Ethernet
Domain worker placement	One worker per machine
Master placement	Dedicated machine
Distributed deployment	N worker machines + 1 master machine

Table 4: Hardware and software configuration used in the experimental test-bed.

5.1 Traffic Realizations and Repetition Procedure

Traffic variability is explicitly incorporated into the experimental methodology. For each traffic-demand level V , ten independent traffic files are generated using ten different random seeds, $\mathcal{F}_V = \{F_{V,1}, F_{V,2}, \dots, F_{V,10}\}$, where $F_{V,s}$ denotes the traffic realization generated for demand level V using seed s . Each experimental configuration is evaluated through ten simulation runs, one for each of the ten traffic files. Changing the random seed modifies the stochastic realization of the traffic input while preserving the parameters that define the corresponding experimental scenario. In particular, the configured traffic demand, vehicle-generation rules, simulation horizon, routing configuration, and vehicle characteristics remain those defined for the same traffic level.

For each seed, the same traffic realization is reused across all configurations being compared. Therefore, changing the decomposition orientation or the number of simulation domains does not regenerate the traffic demand. Likewise, the accident and no-accident variants use the same underlying traffic realization for a given seed; the only difference is the incident introduced according to the configuration described in Section 4.3.

For a particular traffic demand V , accident condition A , decomposition orientation P , and number of simulation domains N , the experimental set is $\mathcal{E}_{P,N,V,A} = \{E_{P,N,V,A,1}, \dots, E_{P,N,V,A,10}\}$.

Each experimental configuration is therefore evaluated through ten simulation runs, one for each traffic realization. Unless otherwise stated, the reported value for each configuration corresponds to the arithmetic mean over these ten runs.

5.2 Experiment I: Single-Node Process-to-Thread Evaluation

Experiment I compares the process-based parallel execution layer introduced in our previous work [7] with the new thread-based implementation running on free-threaded

Experimental factor	Configuration
Road network	Rotterdam
Local execution model	Selected from Experiment I
Local worker count	Selected from Experiment I
Communication protocol	TCP
Decomposition orientation	X-oriented, Y-oriented
Number of domains	2, 3, 4, 6, 8, 10, 12
Traffic demand	15k, 20k, 30k vehicles
Traffic condition	No accident, Accident
Simulation horizon	14,400 s
Simulation step	1 s
Traffic realizations	10

Table 5: The local execution configuration and TCP communication mechanism remain fixed throughout Experiment II.

Python. The two execution models are denoted as $C_{\text{proc}}(k)$ and $C_{\text{thread}}(k)$, corresponding to one SUMO instance with k worker processes or k worker threads, respectively.

The comparison considers $k \in \{1, 2, 4, 6\}$ workers. Both implementations use the same simulation inputs and external workload, so that the parallel execution mechanism is the only factor modified. The sequential implementation, in which the external workload is executed without parallel workers, is used as the reference baseline. The experiments are performed using the traffic scenarios defined in Section 4.3.

The best-performing configuration is subsequently used as the local execution configuration in Experiment II.

5.3 Experiment II: Distributed Architecture Evaluation

Experiment II evaluates the complete distributed architecture described in Section 3. The local execution model is no longer treated as an experimental variable. Let C_{local} denote the best-performing single-node configuration identified in Experiment I. The distributed configuration $C_{\text{dist}}(N)$ consists of N domain workers, each responsible for one simulation domain and using the same C_{local} configuration.

The distributed evaluation considers the two decomposition orientations defined in Section 4.4, $P \in \{\text{X-oriented}, \text{Y-oriented}\}$, with $N \in \{2, 3, 4, 6, 8, 10, 12\}$ simulation domains. The traffic scenarios are those defined in Section 4.3.

Combining the seven domain counts, two decomposition orientations, and six traffic scenarios results in $7 \times 2 \times 6 = 84$ distributed configurations.

Table 5 summarizes the configuration of Experiment II.

5.4 Baseline Executions and Data Collection

Each distributed configuration is evaluated against the corresponding non-partitioned sequential execution using the same traffic realization, as defined in Section 5.1.

For traffic realization s , $E_{\text{reference},s}$ denotes the non-partitioned sequential execution, whereas $E_{\text{distributed},s}$ denotes the corresponding distributed execution.

Traffic outputs are stored using the original SUMO edge identifiers, allowing equivalent road segments to be matched between $E_{\text{reference},s}$ and $E_{\text{distributed},s}$ for the subsequent traffic-volume analysis.

The distributed framework additionally records total execution time, inter-domain vehicle transfers, synchronization events, accumulated synchronization time, and the number of active vehicles associated with each domain worker throughout the simulation.

6 Experimental Results

The experimental results address traffic-volume preservation, single-node performance, and distributed scalability.

6.1 Traffic Level Analysis

The first analysis evaluates whether spatially decomposing the Rotterdam network alters the traffic volume reproduced by the original non-partitioned SUMO simulation.

Traffic validation is performed for the 20,000-vehicle no-accident scenario over the complete 14,400-step simulation horizon. Rather than repeating the traffic-equivalence analysis for every domain count included in the computational experiment, three representative medium-to-high domain counts are selected, $N \in \{6, 8, 12\}$. These configurations were selected because they correspond to progressively more fragmented distributed executions, increase the number of domain boundaries and inter-domain vehicle transfers, and include the most fragmented case evaluated in the distributed experiment.

Both X-oriented and Y-oriented decompositions are considered for each selected domain count, giving the six traffic-validation configurations $\{X_6, Y_6, X_8, Y_8, X_{12}, Y_{12}\}$.

The complete Rotterdam SUMO network contains 22,332 edges, as described in Section 4.1. For the traffic-volume comparison, only edges registering at least five vehicles in the reference execution are retained. Traffic volume is extracted from SUMO `edgeData` outputs. For every road segment and aggregation interval, SUMO reports the number of vehicles entering from another edge (**entered**) and the number of vehicles inserted directly into the segment (**departed**). Because a vehicle crossing a domain boundary is re-materialized in the destination domain worker, the principal traffic-volume measure is defined as

$$Q_e = \sum_t (\text{entered}_{e,t} + \text{departed}_{e,t}). \quad (1)$$

Traffic equivalence is assessed using complementary indicators. Correlation alone is not sufficient to demonstrate agreement because two sets of observations may be highly correlated while differing in location or scale [29, 30]. Therefore, the main analysis combines a Two One-Sided Tests (TOST) equivalence test [31, 32], Lin’s Concordance

Correlation Coefficient (CCC) [30], and the GEH statistic, widely used in traffic-model validation [33]. The principal traffic-volume results are reported in Table 6.

Configuration	Volume ratio	CCC	Pearson	GEH < 5	Median error	$\pm 10\%$
X-oriented-6	0.9992	0.9720	0.9725	97.4%	1.65%	91.9%
Y-oriented-6	0.9419	0.9655	0.9663	91.5%	4.18%	73.5%
X-oriented-8	0.9808	0.9541	0.9552	95.2%	1.45%	85.8%
Y-oriented-8	0.9762	0.9539	0.9547	90.9%	4.00%	76.7%
X-oriented-12	0.9903	0.9480	0.9500	92.9%	3.40%	75.7%
Y-oriented-12	0.9434	0.8890	0.8898	89.6%	3.79%	75.8%

Table 6: Traffic-volume agreement between the non-partitioned Rotterdam reference simulation and the six evaluated traffic-validation configurations.

All six evaluated traffic-validation configurations satisfy the predefined TOST equivalence criterion for the $\pm 10\%$ equivalence region. This is shown in Figure 6, where every 90% confidence interval remains between the lower and upper equivalence bounds.

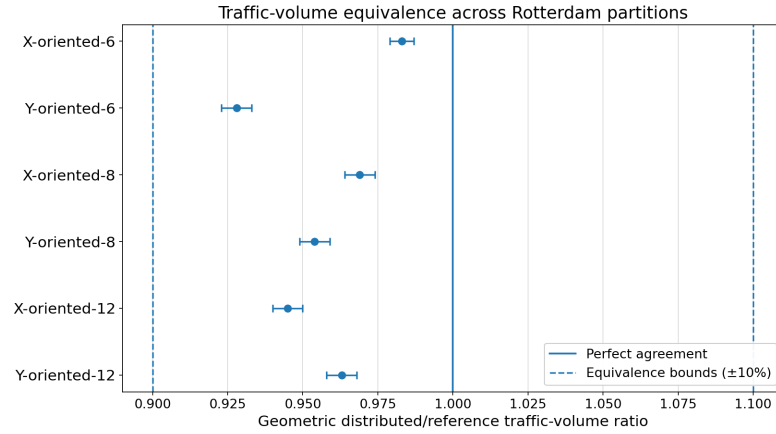


Fig. 6: Traffic-volume equivalence between the non-partitioned Rotterdam reference simulation and the six evaluated traffic-validation configurations. Points represent the geometric distributed/reference traffic-volume ratio and horizontal intervals the corresponding 90% confidence intervals. The solid vertical line indicates perfect agreement and the dashed lines indicate the predefined $\pm 10\%$ equivalence region.

The strongest agreement is obtained with X-oriented-6, which reproduces the total reference traffic volume with a ratio of 0.9992, corresponding to a difference of only 0.08% over approximately 1.59 million accumulated vehicle passages. At segment level

it reaches $CCC = 0.9720$, Pearson $r = 0.9725$, and a median relative error of 1.65%. Moreover, 91.9% of the analyzed road segments remain within $\pm 10\%$ of the reference value.

The remaining five traffic-validation configurations also preserve the network traffic volume with good agreement. Aggregate traffic-volume ratios range from 0.9419 to 0.9992, while CCC remains between 0.8890 and 0.9720. Five of the six configurations achieve Spearman rank correlation above 0.99, indicating that the spatial hierarchy of the road network is preserved: road segments that carry the highest traffic in the reference simulation remain the most heavily used segments in the distributed executions.

The GEH analysis provides a traffic-specific view of segment-level agreement. As shown in Figure 7, between 89.6% and 97.4% of the analyzed segments satisfy $GEH < 5$ across the six traffic-validation configurations, always above the 85% reference level suggested in TAG Unit M3.1 [33].

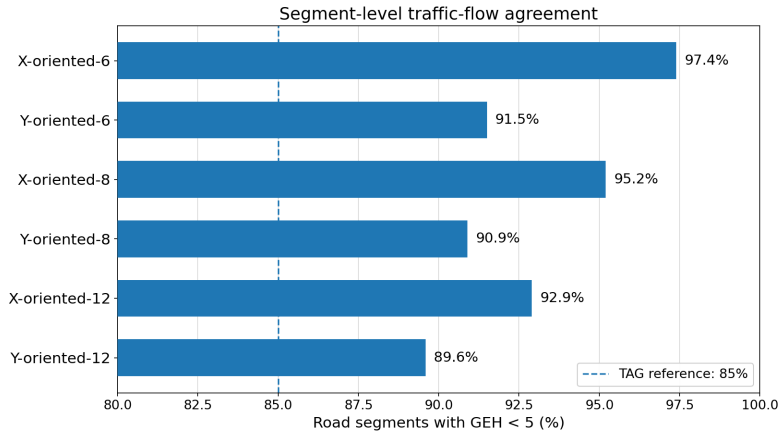


Fig. 7: Percentage of Rotterdam road segments satisfying $GEH < 5$ for each evaluated traffic-validation configuration. The dashed line indicates the 85% reference level reported in TAG Unit M3.1 [33].

X-oriented configurations generally provide the closest local agreement, although all six evaluated traffic-validation configurations remain within the selected equivalence region. To verify that the agreement is not dominated by lightly trafficked road segments, the 4,920 analyzed edges are stratified according to the accumulated reference volume into the four groups 5–50, 50–200, 200–1,000, and more than 1,000 vehicles. Figure 8 shows that the distributed/reference volume ratio remains close to one across the complete range of traffic demand, including the highest-volume stratum.

Taken together, these results indicate that, for the evaluated traffic-validation configurations, the distributed simulation preserves the traffic-volume characteristics of the non-partitioned Rotterdam simulation. The spatial decomposition therefore does

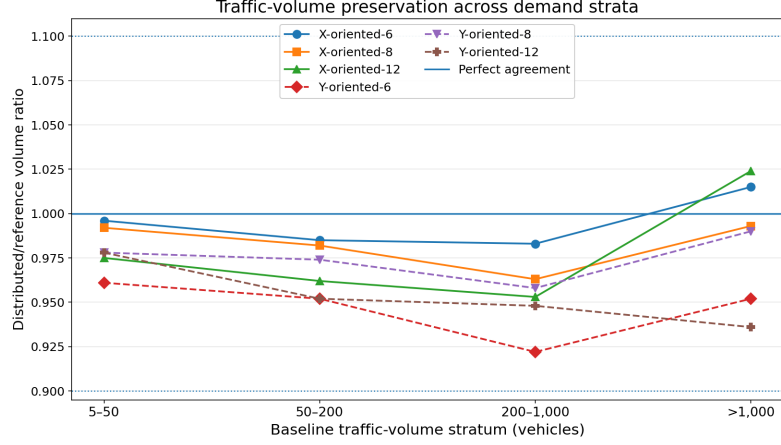


Fig. 8: Distributed/reference traffic-volume ratio according to the traffic-volume stratum of the reference road segment. The solid line at 1 indicates perfect agreement, while the dotted lines indicate $\pm 10\%$.

not introduce a systematic loss or duplication of traffic volume, which enables the distributed architecture to be analyzed next from the decomposition and computational perspectives.

6.2 Single-Node Process-to-Thread Performance

This analysis evaluates the performance of the thread-based single-node implementation against the previous process-based implementation [7].

Both implementations perform the same external vehicle- and segment-level computations, differing only in the use of worker threads instead of independent worker processes. The comparison considers $k \in \{1, 2, 4, 6\}$ local workers.

The process-based speedup relative to the original sequential execution is defined as

$$S_{\text{proc}}(k) = \frac{T_{\text{seq}}}{T_{\text{proc}}(k)}. \quad (2)$$

The additional acceleration obtained by replacing processes with worker threads is

$$S_{\text{thread/proc}}(k) = \frac{T_{\text{proc}}(k)}{T_{\text{thread}}(k)}. \quad (3)$$

where T_{seq} , $T_{\text{proc}}(k)$, and $T_{\text{thread}}(k)$ denote the total execution time of the sequential baseline and of the process-based and thread-based configurations with k local workers, respectively.

The resulting thread-based speedup relative to the original sequential execution is therefore,

$$S_{\text{thread}}(k) = S_{\text{proc}}(k) \times S_{\text{thread/proc}}(k). \quad (4)$$

Vehicles	Traffic condition	Local Workers	S_{proc}	S_{thread}	$S_{\text{thread}/\text{proc}}$	Execution time reduction threads vs. sequential (%)
15,000	No accident	1	1.00×	1.00×	1.00×	0.0%
		2	2.10×	2.96×	1.41×	66.2%
		4	2.74×	3.97×	1.45×	74.8%
		6	3.70×	5.77×	1.56×	82.7%
20,000	No accident	1	1.00×	1.00×	1.00×	0.0%
		2	2.07×	2.73×	1.32×	63.4%
		4	2.71×	3.85×	1.42×	74.0%
		6	3.54×	5.49×	1.55×	81.8%
30,000	No accident	1	1.00×	1.00×	1.00×	0.0%
		2	2.09×	2.86×	1.37×	65.1%
		4	2.61×	3.68×	1.41×	72.8%
		6	3.59×	5.35×	1.49×	81.3%
15,000	Accident	1	1.00×	1.00×	1.00×	0.0%
		2	1.80×	2.29×	1.27×	56.3%
		4	2.50×	3.53×	1.41×	71.6%
		6	3.05×	4.18×	1.37×	76.1%
20,000	Accident	1	1.00×	1.00×	1.00×	0.0%
		2	1.81×	2.21×	1.22×	54.7%
		4	2.56×	3.56×	1.39×	71.9%
		6	3.09×	4.57×	1.48×	78.1%
30,000	Accident	1	1.00×	1.00×	1.00×	0.0%
		2	1.90×	2.36×	1.24×	57.6%
		4	2.61×	3.55×	1.36×	71.8%
		6	3.01×	3.88×	1.29×	74.2%

Table 7: Single-node performance comparison between process-based and thread-based execution across the six evaluated traffic scenarios.

Table 7 summarizes the single-node results for all traffic scenarios. The table reports the process-based speedup over the sequential baseline (S_{proc}), the thread-based speedup over the sequential baseline (S_{thread}), the additional thread-over-process improvement ($S_{\text{thread}/\text{proc}}$), and the corresponding execution-time reduction of the thread-based implementation relative to the sequential baseline.

The process-based implementation reduces execution time consistently as the number of local workers increases across all evaluated traffic scenarios. With six worker processes, the speedup ranges from 3.01× to 3.70×, depending on the traffic demand and incident condition. Replacing the process-based workers with free-threaded workers provides an additional performance improvement in every configuration with more than one worker. For the six-worker configuration, the thread-over-process improvement ranges from 1.29× to 1.56×.

The thread-based implementation achieves its highest overall performance with six workers in all evaluated scenarios. The resulting speedup over the original sequential implementation ranges from 3.88× in the 30,000-vehicle accident scenario to 5.77× in the 15,000-vehicle no-accident scenario, corresponding to execution-time reductions of 74.2% and 82.7%, respectively. The no-accident scenarios generally exhibit higher speedups, reaching 5.77×, 5.49×, and 5.35× for 15,000, 20,000, and 30,000 vehicles, respectively. Under accident conditions, the corresponding speedups are 4.18×, 4.57×, and 3.88×.

Although the relative advantage of threads over processes does not increase monotonically with the number of local workers in every scenario, the six-thread configuration consistently provides the largest total acceleration. This configuration also uses the complete set of six physical CPU cores available on the experimental platform. Therefore, one SUMO instance with six free-threaded workers is selected as the local execution configuration for the distributed experiments and is kept fixed throughout the following analysis.

6.3 Distributed Execution Performance

This analysis evaluates the complete distributed execution model. Each domain worker retains the six-thread local configuration selected above, while the Rotterdam network is spatially decomposed among several coordinated SUMO instances.

All results correspond exclusively to persistent TCP communication. The number of domain workers is varied as $N \in \{2, 3, 4, 6, 8, 10, 12\}$, using both X-oriented and Y-oriented decompositions, three traffic-demand levels, and both no-accident and accident conditions.

The analysis first examines how the spatial decomposition changes the number of inter-domain vehicle transfers and then evaluates the resulting scalability and synchronization cost.

6.3.1 Spatial Decomposition and Inter-Domain Vehicle Transfers

Having established traffic-volume preservation for the selected validation configurations, the next question is how the spatial decomposition affects inter-domain vehicle transfers. In a distributed execution model, the spatial decomposition determines how often vehicles must cross from one simulation domain to another. These crossings generate the inter-domain vehicle transfers that are at the core of distributed SUMO execution [15, 34, 35].

To characterize this effect, the analysis focuses on the number of inter-domain vehicle transfers. Since the total traffic demand varies between scenarios, the raw number of transfers is normalized by the configured number of vehicles, $h = H/V$, where H is the number of outgoing vehicle transfers recorded by the domain workers and V is the configured number of vehicles in the scenario. This metric quantifies the communication activity induced by the spatial decomposition in terms of vehicle-state transfer events, rather than the time spent in TCP communication.

Across the complete set of 840 distributed simulation runs, every outgoing vehicle transfer has a corresponding incoming transfer, $H_{\text{out}} = H_{\text{in}}$, and all runs complete with zero transfer failures and zero barrier timeouts. Therefore, the communication layer preserves transfer integrity for every evaluated distributed configuration.

Figure 9 summarizes the transfer burden induced by the decomposition geometry. For each decomposition orientation and domain count, the plotted value is the mean across the six traffic scenarios, while the shaded band represents the minimum and maximum scenario values.

The main trend is clear: the transfer burden increases strongly as the network is divided into more spatial domains. For X-oriented, the mean number of vehicle

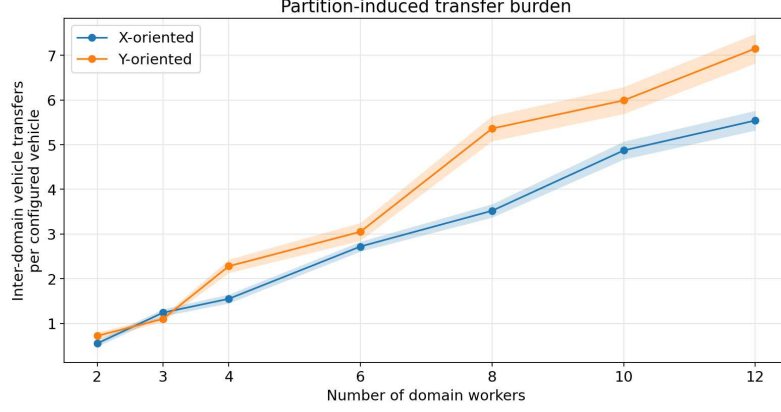


Fig. 9: Mean number of inter-domain vehicle transfers per configured vehicle as a function of the number of domain workers. The shaded bands indicate the minimum and maximum values observed across the six traffic scenarios (three demand levels, with and without accident).

transfers per vehicle rises from 0.58 with two domain workers to 5.55 with twelve domain workers. For Y-oriented, the corresponding increase is from 0.71 to 7.18.

This behavior is expected. Increasing the number of simulation domains creates more domain boundaries and shortens the average distance that a vehicle can travel before leaving its current simulation domain. As a result, a larger fraction of trips requires one or more vehicle transfers.

The decomposition orientation also matters. For two domain workers the difference between the two orientations is modest, but from four domain workers onward Y-oriented systematically produces a higher transfer burden than X-oriented. At twelve domain workers, for example, Y-oriented generates on average about 29% more vehicle transfers per vehicle than X-oriented (7.18 versus 5.55). This indicates that the Y-oriented decomposition induces more frequent inter-domain vehicle transfers for the evaluated traffic scenarios.

The decomposition results also anticipate the computational behavior discussed next. A decomposition that produces a larger number of inter-domain vehicle transfers increases the communication activity of the distributed execution and may also increase the coordination requirements among domain workers.

6.3.2 Scalability and Synchronization

This subsection evaluates the scalability of the distributed architecture using two complementary speedup metrics: the overall speedup with respect to the sequential reference execution and the additional speedup with respect to the best-performing single-node configuration. The synchronization overhead introduced as the number of domain workers increases is also analyzed.

The first performance metric is the speedup with respect to the sequential reference execution,

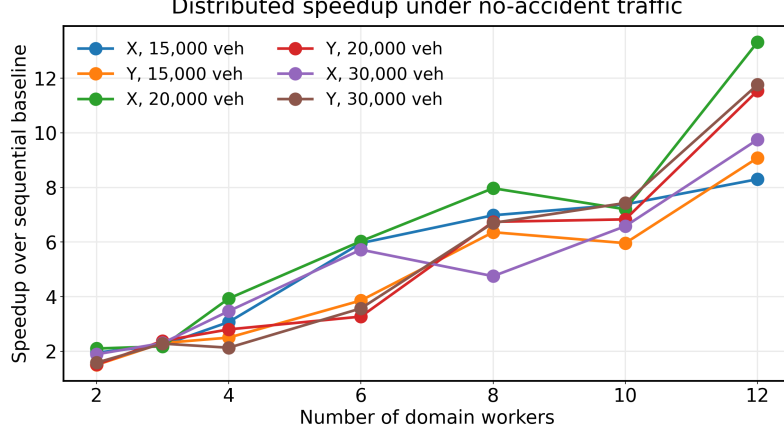


Fig. 10: Distributed speedup over the sequential baseline under no-accident traffic conditions. Each curve corresponds to one demand level and one decomposition orientation.

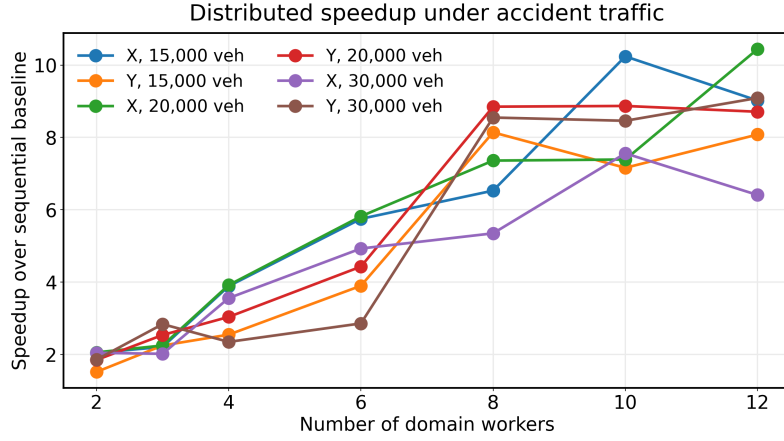


Fig. 11: Distributed speedup over the sequential baseline under accident traffic conditions. Each curve corresponds to one demand level and one decomposition orientation.

$$S(N) = \frac{T_{seq}}{T_{dist}}, \quad (5)$$

where T_{seq} and $T_{dist}(N)$ denote the total execution time of the non-partitioned sequential baseline and the distributed configuration with N domain workers, respectively. The problem size remains fixed while the number of domain workers increases.

Figures 10 and 11 show the speedup obtained for the three traffic-demand levels, with and without accident, as the number of domain workers increases from 2 to 12.

The distributed architecture provides a clear computational benefit in every evaluated configuration. At the lowest domain count ($N = 2$), speedup ranges from approximately $1.51\times$ to $2.06\times$. As the number of domain workers increases, the achieved speedup also increases and reaches values between $8.08\times$ and $13.32\times$ at $N = 12$, depending on the traffic demand, decomposition orientation, and accident condition.

The best overall result is obtained for the X-oriented decomposition with $N = 12$ in the 20,000-vehicle no-accident scenario, with a speedup of $13.320\times$. High speedups are also obtained in the accident scenarios, such as $10.435\times$ for the X-oriented decomposition with $N = 12$ and 20,000 vehicles, and $10.245\times$ for the X-oriented decomposition with $N = 10$ and 15,000 vehicles.

The effect of decomposition orientation is not uniform across all domain counts. X-oriented is consistently better for $N = 2, 4$, and 6 in all six traffic scenarios, whereas Y-oriented is consistently better at $N = 3$. For $N = 8, 10$, and 12 , neither orientation dominates universally: the best result becomes scenario-dependent. For $N = 2, 4$, and 6 , the better performance of the X-oriented decomposition coincides with its lower transfer burden. However, the different behavior observed at $N = 3$ and the scenario-dependent results at higher domain counts indicate that inter-domain vehicle transfers alone do not determine distributed performance; load balance and synchronization also play an important role.

To isolate the additional benefit of distributed execution, a second speedup metric is defined with respect to the best-performing single-node configuration,

$$S_{\text{dist/local}}(N) = \frac{T_{\text{thread},6}}{T_{\text{dist}}(N)}, \quad (6)$$

where $T_{\text{thread},6}$ denotes the total execution time of the six-thread single-node configuration for the same traffic scenario, and $T_{\text{dist}}(N)$ denotes the corresponding distributed execution time with N domain workers. This metric therefore quantifies the additional acceleration obtained by moving from the best single-node configuration to the distributed architecture, independently of the improvement already achieved by the local thread-based execution layer.

Vehicles	Traffic condition	$S_{\text{thread}(6)}$	Best configuration	$S(N)$	$S_{\text{dist/local}}(N)$
15,000	No accident	$5.77\times$	Y-12	$9.085\times$	$1.57\times$
15,000	Accident	$4.18\times$	X-10	$10.245\times$	$2.45\times$
20,000	No accident	$5.49\times$	X-12	$13.320\times$	$2.43\times$
20,000	Accident	$4.57\times$	X-12	$10.435\times$	$2.28\times$
30,000	No accident	$5.35\times$	Y-12	$11.766\times$	$2.20\times$
30,000	Accident	$3.88\times$	Y-12	$9.084\times$	$2.34\times$

Table 8: Comparison between the six-thread single-node configuration and the best-performing distributed configuration for each traffic scenario.

Table 8 shows that the best distributed configuration for each traffic scenario provides an additional speedup of $1.57\times$ to $2.45\times$ over the six-thread single-node configuration. This gain is obtained over an already accelerated single-node baseline, which achieves speedups between $3.88\times$ and $5.77\times$ over sequential execution.

The main scalability limit appears in the cost of lockstep synchronization. Figure 12 shows the fraction of the total execution time spent in the global barrier. This fraction is non-negligible even at small domain count and grows with the number of workers, as expected in a distributed lockstep execution [35, 36]. For each decomposition orientation and domain count, the plotted value is the mean across the six traffic scenarios, while the shaded band represents the minimum and maximum scenario values.

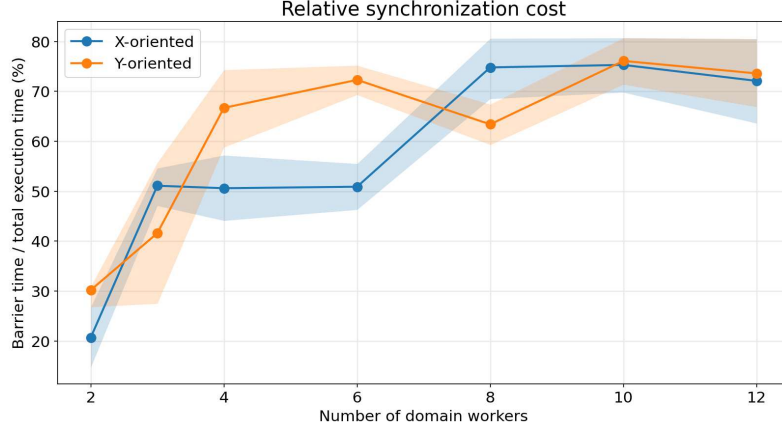


Fig. 12: Fraction of the total execution time spent in barrier synchronization as a function of the number of domain workers. The shaded bands indicate the minimum and maximum values observed across the six traffic scenarios.

For X-oriented, the mean barrier share rises from 20.8% at $N = 2$ to 72.2% at $N = 12$. For Y-oriented, it rises from 30.3% to 73.7%. Therefore, a large fraction of the total distributed execution time is ultimately spent waiting for global synchronization rather than performing local traffic computation.

Nevertheless, the distributed architecture still provides additional acceleration because the reduction in local simulation time compensates for the added synchronization cost over a substantial part of the explored configuration space. The results therefore show the expected scalability trade-off: increasing the number of domain workers reduces local computation but increases coordination overhead. In the present Rotterdam experiments, the best-performing configuration for each traffic scenario is obtained with either 10 or 12 domain workers.

Although the number of inter-domain vehicle transfers increases with the domain count, the measured synchronization time shows that coordination among domain workers becomes the dominant observed limitation to further scalability.

The proposed architecture is therefore effective: it consistently accelerates the execution of the Rotterdam scenario and achieves speedups above $10\times$ in several configurations.

7 Conclusions

This work has presented and evaluated a two-level parallel architecture for large-scale SUMO traffic simulation that combines free-threaded local parallelism with distributed spatial decomposition. Each simulation domain is assigned to a domain worker running an independent SUMO instance together with a local parallel execution layer, while a central master coordinates inter-domain vehicle transfers and synchronization. The approach extends the previous process-based single-node framework towards both shared-memory thread-level parallelism and coordinated multi-node execution without modifying the internal SUMO simulation engine.

The single-node experiments show that free-threaded execution consistently outperforms the corresponding process-based implementation when more than one local worker is used. The six-thread configuration provides the best single-node performance in all six evaluated traffic scenarios, achieving speedups between $3.88\times$ and $5.77\times$ over the sequential baseline. These results correspond to execution-time reductions ranging from 74.2% to 82.7% and support the use of six local worker threads as the execution layer within each domain worker.

The distributed experiments demonstrate that spatial decomposition can provide additional acceleration beyond the best single-node configuration. The best-performing distributed configuration for each traffic scenario uses either 10 or 12 domain workers and achieves speedups between $9.084\times$ and $13.320\times$ with respect to the sequential baseline. More importantly, when the contribution of the local six-thread execution layer is factored out, the distributed architecture still provides an additional speedup between $1.57\times$ and $2.45\times$. The maximum overall speedup, $13.320\times$, is obtained with the X-oriented 12-domain configuration for the 20,000-vehicle no-accident scenario.

The traffic-level analysis indicates that the evaluated distributed configurations preserve the traffic-volume characteristics of the original non-partitioned SUMO simulation. All six traffic-validation configurations satisfy the predefined equivalence criterion, while between 89.6% and 97.4% of the analyzed road segments satisfy $GEH < 5$. In addition, across the complete set of 840 distributed simulation runs, every outgoing vehicle transfer has a corresponding incoming transfer, with zero transfer failures and zero barrier timeouts. These results show that the computational gains are obtained without introducing systematic loss or duplication of traffic volume in the evaluated configurations.

The experiments also reveal the importance of spatial decomposition geometry. Increasing the number of domain workers reduces the amount of local simulation work performed by each worker, but also increases the number of inter-domain vehicle transfers and the associated coordination requirements. The Y-oriented decomposition generally produces a higher transfer burden than the X-oriented decomposition at medium and high domain counts. However, the best-performing orientation becomes scenario-dependent for larger numbers of domains, indicating that transfer activity

alone does not determine distributed performance and that workload distribution must also be considered.

The principal scalability limitation observed in the current implementation is global lockstep synchronization. The mean fraction of execution time spent waiting at the synchronization barrier increases from 20.8% to 72.2% for the X-oriented decomposition and from 30.3% to 73.7% for the Y-oriented decomposition when increasing from 2 to 12 domain workers. Despite this overhead, distributed execution provides clear acceleration over the sequential baseline, while the best-performing distributed configurations also outperform the best single-node implementation.

Overall, the results show that combining free-threaded local execution with distributed spatial decomposition is an effective approach for accelerating large-scale SUMO simulations while leaving the simulator core unmodified. Future work will focus on reducing synchronization waiting time, introducing dynamic load balancing among domain workers, and investigating adaptive decomposition strategies in which domain boundaries evolve according to the spatial and temporal distribution of traffic.

8 Acknowledgments

This work was carried out within the project *Dynamic and autonomous selection of safe and sustainable routes* (PID2023-152472OB-I00), funded by MICIU/AEI/10.13039/501100011033 and FEDER, EU. and by the research project PID2023-146569NB-C22 funded by MCIN/AEI/10.13039/501100011033

9 Data availability

The source codes for distributed simulations are available at <https://github.com/AiA-UJI/JOSAiA26>

References

- [1] Lopez, P.A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., Wießner, E.: Microscopic traffic simulation using SUMO. In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 2575–2582. IEEE, ??? (2018). <https://doi.org/10.1109/ITSC.2018.8569938>
- [2] Tran, D.-K., Hoang, D.-Q., Le, V.-T., Thai, M.-D.N., Do, T.-H.: Real-time traffic flow prediction using big data analytics. In: Nguyen, N.T., Dao, N.-N., Pham, Q.-D., Le, H.A. (eds.) *Intelligence of Things: Technologies and Applications (ICIT 2022)*. Lecture Notes on Data Engineering and Communications Technologies, vol. 148, pp. 398–405. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15063-0_38
- [3] Ventresque, A., Bragard, Q., Liu, E.S., Nowak, D., Murphy, L., Theodoropoulos, G., Liu, Q.: SParTSim: A space partitioning guided by road network for distributed traffic simulations. In: 2012 IEEE/ACM 16th International Symposium

- on Distributed Simulation and Real Time Applications (DS-RT), pp. 202–209. IEEE, ??? (2012). <https://doi.org/10.1109/DS-RT.2012.37>
- [4] Bragard, Q., Ventresque, A., Murphy, L.: Self-balancing decentralized distributed platform for urban traffic simulation. *IEEE Transactions on Intelligent Transportation Systems* **18**(5), 1190–1197 (2017) <https://doi.org/10.1109/TITS.2016.2603171>
 - [5] Chen, H., Yang, K., Rizzo, S.G., Vantini, G., Taylor, P., Ma, X., Chawla, S.: QarSUMO: A parallel, congestion-optimized traffic simulator. In: 2020 IEEE International Conference on Big Data (Big Data), pp. 2765–2774. IEEE, ??? (2020). <https://arxiv.org/abs/2010.03289>
 - [6] Eclipse SUMO Project: Can SUMO Be Run in Parallel on Multiple Cores or Computers? <https://eclipse.dev/sumo/docs/FAQ.html>. SUMO User Documentation. Accessed: 2026-07-22 (n.d.)
 - [7] Tomás, V.R., Castillo, M., Monzón-Catalán, I., García, L.A.: Parallel agent-based modeling for improving traffic flow simulation. *Transportation Research Interdisciplinary Perspectives* **37**, 101930 (2026) <https://doi.org/10.1016/j.trip.2026.101930>
 - [8] Potuzak, T.: Benchmark of road network division methods for distributed or parallel road traffic simulation. *The Journal of Supercomputing* **82**(6), 374 (2026) <https://doi.org/10.1007/s11227-026-08542-1>
 - [9] Barceló, J. (ed.): *Fundamentals of Traffic Simulation*. International Series in Operations Research & Management Science, vol. 145. Springer, New York, NY (2010). <https://doi.org/10.1007/978-1-4419-6142-6>
 - [10] Erdmann, J.: SUMO’s lane-changing model. In: *Modeling Mobility with Open Data*. Lecture Notes in Mobility, pp. 105–123. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15024-6_7
 - [11] Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D.: SUMO—simulation of urban MObility: An overview. In: *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pp. 63–68. ThinkMind, Barcelona, Spain (2011)
 - [12] Varga, B., Ormándi, T., Tettamanti, T.: EGO-centric, multi-scale co-simulation to tackle large urban traffic scenarios. *IEEE Access* **11**, 57437–57447 (2023) <https://doi.org/10.1109/ACCESS.2023.3284316>
 - [13] Bin Khunayn, E., Xie, H., Karunasekera, S., Ramamohanarao, K.: Dynamic straggler mitigation for large-scale spatial simulations. *ACM Transactions on Spatial Algorithms and Systems* **9**(2), 1–34 (2023) <https://doi.org/10.1145/3578933>

- [14] Monzón Catalán, I., Tomás López, V.R., Castillo, M., García, L.A.: Comparative Evaluation of Process-Based and Thread-Based Parallelization for Traffic Simulation. Poster presented at the Joint Conferences HPC 2026 and CMMSE 2026 (2026)
- [15] Bragard, Q., Ventresque, A., Murphy, L.: dSUMO: Towards a distributed SUMO. In: Proceedings of the First SUMO User Conference, Berlin, Germany (2013)
- [16] Ahmed, M.S., Hoque, M.A.: Partitioning of urban transportation networks utilizing real-world traffic parameters for distributed simulation in SUMO. In: 2016 IEEE Vehicular Networking Conference (VNC). IEEE, ??? (2016). <https://doi.org/10.1109/VNC.2016.7835942>
- [17] Liu, Z., Xie, S., Zhang, H., Zhou, D., Yang, Y.: A parallel computing framework for large-scale microscopic traffic simulation based on spectral partitioning. *Transportation Research Part E: Logistics and Transportation Review* **181**, 103368 (2024) <https://doi.org/10.1016/j.tre.2023.103368>
- [18] Zhu, H., Xie, X., Tang, K., Feng, J., Rao, W.: A novel distributed parallel simulation method with dynamic partitioning using KLeiden-based community detection. *Transportation Research Part E: Logistics and Transportation Review* **207** (2025)
- [19] Potuzak, T.: Distributed-parallel road traffic simulator for clusters of multi-core computers. In: 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications, pp. 195–201 (2012). <https://doi.org/10.1109/DS-RT.2012.36>
- [20] Eddy, W.M.: Transmission control protocol (tcp). RFC 9293, Internet Engineering Task Force (August 2022). <https://doi.org/10.17487/RFC9293> . <https://www.rfc-editor.org/info/rfc9293>
- [21] Rijkswaterstaat: Snelweg A15. Rijkswaterstaat, Ministry of Infrastructure and Water Management. Official description of the A15 motorway and its role connecting the Rotterdam port and industrial area with the European hinterland (2026)
- [22] Rijkswaterstaat: A12/A15 Ressen–Oudbroeken: doelen en resultaten. Rijkswaterstaat, Ministry of Infrastructure and Water Management. Official description of the A15–A12 connection between the Rotterdam port region and Germany (2026)
- [23] Rijkswaterstaat: A20: verbreding Nieuwerkerk aan den IJssel–Gouda. Rijkswaterstaat, Ministry of Infrastructure and Water Management. Official Rijkswaterstaat information on the A20 corridor connecting the Rotterdam region with the central Netherlands (2026)

- [24] Rijkswaterstaat: Handboek verkeerscentrales. Technical report, Rijkswaterstaat (December 2019). Traffic Control Centres Handbook, Ministry of Infrastructure and Water Management, The Netherlands
- [25] Rijkswaterstaat: Verkeerscentrum Nederland. <https://www.rijkswaterstaat.nl/wegen/wegbeheer/wegbeheer-in-nederland/verkeerscentrum-nederland>. Accessed: 2026-09-02 (2026)
- [26] Rijkswaterstaat: Verkeerscentrale Zuidwest-Nederland. <https://www.rijkswaterstaat.nl/over-ons/onze-organisatie/organisatiestructuur/verkeer-en-watermanagement/verkeercentrale-zuidwest-nederland>. Accessed: 2026-09-02 (2026)
- [27] Rijkswaterstaat: Regionale verkeerscentrale Midden-Nederland. <https://www.rijkswaterstaat.nl/over-ons/onze-organisatie/organisatiestructuur/verkeer-en-watermanagement/regionale-verkeerscentrale-midden-nederland>. Accessed: 2026-09-02 (2026)
- [28] European Commission: Trans-European Transport Network (TEN-T). https://transport.ec.europa.eu/transport-themes/infrastructure-and-investment/trans-european-transport-network-ten-t_en. Accessed: 2026-09-02 (2026)
- [29] Bland, J.M., Altman, D.G.: Statistical methods for assessing agreement between two methods of clinical measurement. *The Lancet* **1**(8476), 307–310 (1986) [https://doi.org/10.1016/S0140-6736\(86\)90837-8](https://doi.org/10.1016/S0140-6736(86)90837-8)
- [30] Lin, L.I.-K.: A concordance correlation coefficient to evaluate reproducibility. *Biometrics* **45**(1), 255–268 (1989) <https://doi.org/10.2307/2532051>
- [31] Schuirmann, D.J.: A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of Pharmacokinetics and Biopharmaceutics* **15**(6), 657–680 (1987) <https://doi.org/10.1007/BF01068419>
- [32] Lakens, D.: Equivalence tests: A practical primer for t tests, correlations, and meta-analyses. *Social Psychological and Personality Science* **8**(4), 355–362 (2017) <https://doi.org/10.1177/1948550617697177>
- [33] Department for Transport: TAG Unit M3.1: Highway Assignment Modelling. Technical report, UK Department for Transport (2026). Transport Analysis Guidance, updated 28 May 2026. <https://www.gov.uk/government/publications/tag-unit-m3-1-highway-assignment-modelling> Accessed 2026-09-02
- [34] Acosta, A., Espinosa, J., Espinosa, J.: Distributed simulation in SUMO revisited: Strategies for network partitioning and border edges management. In: Proceedings of the 4th SUMO User Conference, pp. 61–71. Deutsches Zentrum für Luft- und Raumfahrt e.V., Berlin, Germany (2016)

- [35] Arroyo, N., Acosta, A., Espinosa, J., Espinosa, J.: A new strategy for synchronizing traffic flow on a distributed simulation using SUMO. In: SUMO 2018 – Simulating Autonomous and Intermodal Transport Systems. EPiC Series in Engineering, vol. 2, pp. 152–161. EasyChair, ??? (2018). <https://doi.org/10.29007/1x48>
- [36] Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. AFIPS Conference Proceedings **30**, 483–485 (1967) <https://doi.org/10.1145/1465482.1465560>