AGH UNIVERSITY OF KRAKOW

FIELD OF SCIENCE: ENGINEERING AND TECHNOLOGY

SCIENTIFIC DISCIPLINE: INFORMATION AND COMMUNICATION TECHNOLOGY

DOCTORAL THESIS

SUPER-SCALABLE URBAN TRAFFIC SIMULATION

Author: Mateusz Najdek, MSc

Supervisor: Wojciech Turek, DSc, associate professor

Completed in:
AGH University of Krakow
Faculty of Computer Science
Institute of Computer Science

Krakow, 2025

I would like to express my deepest gratitude to my supervisor, Prof. Wojciech Turek, for your thoughtful guidance and clear direction throughout this thesis. Your critical insight and academic perspective helped me stay grounded and focused, and I greatly appreciate the time and knowledge you shared with me.

To my friends—thank you for being my anchor during this process. Your support came in many forms: encouraging messages, much-needed distractions, and the simple but powerful act of showing up when I needed it most. Whether it was a late-night chat or a quiet word of reassurance, you made a difference every step of the way.

The research presented in this work was funded by the National Science Centre, Poland, under the grant no. $2019/35/O/ST6/01806$.

I would also like to acknowledge Polish high-performance computing infrastructure PLGrid(HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. $PLG/2025/018212$.

Contents

1 Introduction

This section explores the rationale for developing effective urban traffic simulations and addresses why there is a push for creating sophisticated urban traffic simulations. It focuses on the importance of using highly detailed simulation models and explains why it is essential to create a method that allows to effectively simulate large urban traffic networks on a supercomputer. Subsequent subsections introduce the thesis and goals of this study and examine its organizational structure.

1.1. Motivation

Efficient urban transportation is crucial for the growth of modern urban areas. As the population grows and suburban regions expand, the daily volume of necessary trips increases. This leads to heightened road traffic within cities, resulting in negative outcomes such as accidents, congestion, energy waste, air pollution, ineffective spatial planning, and suboptimal transportation design. These issues not only increase travel times but also compromise safety, ultimately hindering the primary objective of road transport: quick and efficient point-to-point movement within cities.

Addressing some of these transportation issues is possible through strategic interventions [23]. For instance, the design of a safe, high-capacity traffic intersection involves analyzing the movement of all traffic participants and assessing the optimality of the design. It is also crucial to evaluate how new traffic solutions will mesh with existing structures and whether they might cause extensive congestion during peak hours. Efficient traffic simulations can play a pivotal role here, enabling the assessment of vehicular flow and facilitating the rapid testing of various scenarios.

Traffic simulations are equally valuable in optimizing public transport [32]. Through targeted research, which can be completed swiftly using simulators, public transport systems can be reorganized to enhance efficiency, cut costs, and boost passenger satisfaction. This includes facilitating transitions to different services, like suburban buses. Simulations also allow transport providers to predict and prepare for the impacts of road closures or construction work accurately.

The advent of autonomous vehicles is set to influence road safety further [123]. Traffic simulators can be utilized to analyze how these vehicles integrate with current infrastructure and manually driven cars. Simulations provide a safe environment to test the functionality of algorithms within autonomous vehicles.

Moreover, traffic simulations can aid in managing urban traffic by mitigating the effects of unforeseen road incidents [136], such as accidents and by allowing for optimalization of overall traffic flow through appropiate adjustments of traffic lights. One proposal involves creating a traffic management system capable of dynamically addressing hazardous or inconvenient road situations. Before implementing solutions (like adjusting traffic signals), the system could swiftly simulate multiple scenarios to determine the most effective approach. For such rapid analyses, the efficiency of the simulation system is paramount.

1.1.1. Classification of models in urban traffic simulation

The method and purpose of the simulation determine how traffic flow is depicted, which can vary considerably. There are three primary types of models used in traffic simulations, each differing in their method of representing movement and level of detail [138]:

- **Macroscopic models** focus on the general flow of traffic without detailing individual vehicles or infrastructure.
- **Mesoscopic models** provide aggregated data on groups of vehicles.
- **Microscopic models** offer intricate details about each vehicle, the road infrastructure, and even the driving behaviors of individual drivers.

Microscopic models incorporate the most comprehensive information, thereby delivering a highly detailed and accurate representation of reality and enabling precise analysis of traffic system dynamics. Simulations employing these models often use agent-based approaches [96], where agents—autonomous entities with specific attributes and behaviors—make decisions based on the data available to them. For instance, agents simulating drivers might possess information about their personality, driving style, destination, and chosen route. These models allow for an intricate examination of individual decisions and interactions, providing a deep understanding of complex systems such as urban traffic networks.

To effectively simulate traffic across a broad area, such as within city limits, it is essential to manage the behaviors of numerous agents. This detailed approach ensures reliable and meticulous simulation outcomes. Additionally, the interactions among these agents can lead to emergent behaviors that might not be predictable from the individual agents' rules alone, thus enhancing the realism and utility of the simulations.

Another way to categorize road traffic simulations is by how they represent time. Majority simulations operate on a discrete time model [19], dividing the simulation into distinct steps and updating model parameters at each step to reflect real-world changes. This approach is particularly useful for scenarios where time quantization simplifies the computational model without sacrificing essential dynamics. On the other hand, others use a continuous time model, processing events as they occur in a sequential order based on their creation time [149]. This approach is beneficial for capturing more fluid dynamics and can be critical in environments where timing and reaction speed are crucial, such as in high-speed traffic scenarios on major highways.

Similarly, model representation in traffic simulations can be divided into discrete and continuous models. The discrete model takes inspiration of cellular automata [119], depicting roadways as grids of cells that may contain a vehicle. As vehicles move, they shift a specific number of cells [94]. This method simplifies the modeling of traffic flow and is particularly useful for high-density scenarios like city traffic systems, where the granular control of space can help in managing complex interactions at intersections and multi-lane roads. In contrast, the continuous model [4] represents both roadways and vehicles as continuous segments, with vehicle positions changing based on calculated continuous values. This model allows for more detailed and realistic simulations but often requires greater computational resources. It excels in scenarios where the smooth flow of traffic and individual vehicle trajectories need to be accurately rendered, such as in advanced driving assistance systems (ADAS) testing or the development of autonomous vehicles.

Each of these models and methods brings unique benefits and challenges. Basically, the more complex the model, the more computationally expensive it is, but at the same time it provides more detailed information and better results. By selecting the appropriate model based on the specific requirements and constraints of a project, planners and engineers can enhance the accuracy and efficiency of traffic simulations, ultimately leading to better-informed decisions in transportation systems management and development at the expense of computational time. That is why allowing more complex and detailed models to be executed faster is crucial.

1.1.2. Leveraging a supercomputing platform

As indicated in section 1.1.1, utilizing microscopic models provides detailed and reliable simulation results. However, this technique faces a significant limitation due to the requirement to operate on large amount of map elements, each with its distinct characteristics. The number of actors needed to be depicted in these simulations is so large that it necessitates the deployment on a high-performance computing system to ensure the simulations are executed correctly and without disruptions. It necessitates the development of simulation models and algorithms that are specifically designed to operate efficiently in parallel, distributed environments. Ensuring correct and uninterrupted execution across a large number of cores demands algorithmic structures that support scalable decomposition, synchronization, and communication. This alignment between the modeling method and HPC architecture forms a foundational premise of this research.

To enhance computational efficiency, both discrete and continuous traffic models require partitioning of the simulation space for parallel execution. In discrete models, such as cellular automata, the regular structure of the space may initially appear easier to divide — for instance, actors can be assigned to specific spatial regions handled by separate compute cores. However, even in these cases, synchronization between nodes is necessary, particularly when agents interact across partition boundaries. Continuous models, which often rely on finer-grained representations of space and time, pose additional challenges for parallelization due to the complexity of numerical calculations and the need to maintain consistency in continuous variables. As a result, domain decomposition in such models is more intricate and requires specialized coordination strategies.

The operational requirements of such a detailed, microscopic, and continuous simulation system are ideally met by supercomputers. These powerful machines offer vast computational resources, which enable the execution of many complex, simultaneous tasks. This capability is particularly important for managing the dynamic interactions and real-time computations needed in road traffic simulations. In many cases, the sheer size of the model (including the number of agents, the complexity of the environment, and the volume of data being processed) exceeds the memory capacity of a single compute node, making distributed execution not only desirable but necessary. Supercomputers address this by distributing both computation and data across thousands of interconnected nodes. The ability of such systems to perform these tasks efficiently is one of the reasons they are so integral to advanced simulation efforts, as evidenced by numerous references in the literature [60, 93, 97, 116, 135, 136]. This makes them indispensable tools in the field of traffic management and urban planning, where precise and comprehensive data analysis is crucial for developing effective and scalable solutions.

Parallelizing a simulation that operates on a shared model inherently requires synchronization mechanisms to ensure consistency between distributed computations. Without careful design, such synchronization can introduce subtle inconsistencies or even errors in the simulation results. Therefore, a significant achievement of this work is the proposal of a decomposition strategy that preserves the identity of results between the sequential and parallel executions of the algorithm. This guarantees that the distributed version does not introduce artifacts or divergence, a property that is essential in high-fidelity simulations where accuracy and reproducibility are critical.

1.1.3. Scalability of the algorithm

The interpretation of scalability may differ based on particular system characteristics referenced [12]. Typically, scalability is associated with preserving or enhancing the quality of an application's performance as both the workload executed and the system's capacity for growth increase. We can distinguish 2 types of approaches for accessing the scalability described below.

Strong scalability

In this research, scalability will be characterized by improvements in program performance following an increase in the number of parallel computing nodes. Thus, a program is generally considered scalable if adding more computing cores reduces its execution time for consistent input data or the time remains almost constant when an increase in the input data is proportional to the utilized computing resources.

Scalability is commonly evaluated using a metric of program acceleration known as speedup. Commonly known strong scalability is defined as the ratio of the execution time of the sequential implementation to that of its parallel counterpart for the same input size as shown by equation 1.1. A speedup value close to the number of processing cores indicates strong scalability. Although a linear trend in the time execution graph proportional to utilized computing power (i.e., doubling the cores nearly halves the execution time) is often considered desirable, it does not necessarily represent an optimal or perfect outcome due to overhead from communication, synchronization, and non-parallelizable parts of the algorithm.

$$\text{Speedup} = \frac{\text{Execution time of the task without improvements}}{\text{Execution time after applying improvements}} \tag{1.1}$$

This definition of speedup, formulated by Amdahl in 1967 [5], presupposes a fixed quantity of input data even as the number of processors grows. It notably underscores the crucial interaction between the sequential part of the algorithm, which maintains a constant execution time irrespective of additional threading, and the parallelized section, which can be executed more swiftly due to enhancements. The formula incorporating these elements is given as:

$$S_p = \frac{1}{1 - p + \frac{p}{N}} \tag{1.2}$$

Where:

– Sp denotes the theoretical speedup of the program,

– p is the execution time of the parallelizable section by one processor,

– N is the number of processors deployed.

A distinct characteristic of the speedup graph, as per Amdahl's law, is its tendency to plateau after a certain number of threads are used. This flattening is intrinsically linked to the sequential segment of the algorithm, as the extent of its computational demand dictates when additional cores cease to enhance acceleration. Even as parallel computation times are reduced with the addition of more processors, the duration required for sequential computations remains unchanged.

Weak scalability

Gustafson further expanded upon Amdahl's discussion [45], observing that in real-world applications, the problem size tends to expand with the increase in processor count. Additionally, as problems grow larger, the proportion of time spent on sequential processing typically remains steady, thereby decreasing its relative contribution to overall computations. Conversely, the proportion of time dedicated to parallel processing increases and is distributed among more cores. Gustafson suggested that an algorithm maintaining a constant execution time despite increased problem size and number of processors would be ideally scalable under what is termed Gustafson's law.

To formalize this, Gustafson proposed an alternative definition of speedup that assumes a fixed total execution time and a workload that increases proportionally to the number of processors:

$$S_p = N - p \cdot (N - 1) \tag{1.3}$$

Where:

– S_p denotes the speedup under Gustafson's law,

– N is the number of processors,

– p is the proportion of the execution time that is inherently sequential.

Gustafson's formulation suggests that high speedup can still be achieved even when some sequential components are present, provided that the problem size scales with the computational resources. This concept of weak scalability is particularly relevant to large-scale simulations such as urban traffic modeling, where increasing the spatial scope or behavioral resolution naturally leads to larger datasets and more computational work. In this context, achieving consistent performance while increasing both problem size and processor count reflects the system's ability to maintain efficiency at scale—a key objective of the methods developed in this research.

Beyond strong and weak scalability

The distinction between strong and weak scalability provides a foundational framework for analyzing parallel performance. These discussions led to the differentiation between strong and weak scalability, corresponding to Amdahl's and Gustafson's laws, respectively. These concepts reflect different assumptions about how problem size evolves in relation to computational resources and should not be seen as competing perspectives. They highlight different use cases and system behaviors that are both relevant in high-performance computing environments. These terms should not be viewed as one being better than the other, despite their implied emotional connotations.

In practical applications, scalability is rarely characterized by a single metric. While speedup is often the primary measure, other aspects—such as how efficiently additional resources are used—can also offer insight into performance. However, due to overheads introduced by inter-process communication, synchronization, and memory access patterns, ideal behavior is difficult to achieve. Real-world systems typically exhibit diminishing returns as the number of processors increases. Efficiency is another criterion for assessing scalability. This metric could be defined as the ratio of speedup to the number of cores used to run the program. The optimal efficiency value is 1, although this is challenging to achieve. Programs that consistently exhibit high efficiency values, around 0.8, are considered well-scalable.

Interestingly, under certain conditions, parallel implementations may outperform theoretical expectations—a phenomenon known as superlinear speedup [115]. This effect can occur due to improved cache utilization, reduced contention for memory, or other architectural benefits that emerge when distributing a problem across multiple cores. Although rare, such cases underscore the complexity of performance modeling and the importance of thorough empirical analysis.

In this research, these considerations provide a broader context for evaluating the scalability of the proposed simulation methods. While speedup and resource usage are central to performance evaluation, awareness of exceptional cases and architectural effects ensures a more nuanced understanding of parallel behavior, particularly in large-scale, compute-intensive simulations such as those addressed in this work.

1.1.4. Super-scalability

The concept of super-scalability, introduced in a 2005 study [28], highlights the adaptation of scientific applications to utilize extensive computer architectures with hundreds of thousands of cores. The authors emphasize the

importance of designing efficient programs that maximize the use of available computing power by minimizing the sequential component of computations, as indicated by Amdahl's law, and establishing robust mechanisms to handle anticipated frequent failures.

For effective adaptation, parallel algorithms must exhibit two key characteristics to be deemed super-scalable: scale invariance and inherent error tolerance. Scale invariance ensures that the workload and the communication needs of a single node remain constant regardless of the overall number of nodes in the system. This includes maintaining a consistent number of neighboring nodes for communication, as well as constant message sizes and frequencies. Additionally, the computational demands and data requirements on each node should not escalate as the tasks expands. This design principle helps localize the impact of any single node's failure, thus preventing widespread disruption. Furthermore, the algorithms should be designed to handle a certain number of failures without the necessity to retrieve lost data, ensuring that accurate results are achieved despite such setbacks. While not a focus of this work, this kind of inherent fault tolerance is occurring in systems designed for large-scale distributed computing, where node failures are expected and must be handled gracefully.

From the perspective of developing an urban traffic simulator, super-scalability offers substantial benefits. Achieving scale invariance is particularly vital, as it enables efficient processing across massive computing environments, potentially speeding up calculations linearly with the addition of more computational resources. While ensuring the algorithm's resilience to errors is important, the primary focus of this endeavor should be on achieving and verifying scale invariance to harness the full potential of expansive computational architectures. This approach is essential for optimizing performance and enhancing the reliability of simulations in urban traffic systems.

1.2. Main thesis

The main thesis of the work is as follows:

It is possible to create a super-scalable, urban traffic simulation algorithm with a continuous microscopic traffic model, allowing for scalability to thousands of cores.

This general thesis entails several more specific aspects, such as:

- *It is possible to adopt existing detailed urban traffic models, like car-following or overtaking.*
- *It is possible to create automated model splitting mechanisms for complex, real-life urban road systems.*
- *It is possible to design a parallelization strategy that ensures full consistency of simulation results between parallel and sequential executions, preserving correctness despite the need for synchronization across distributed nodes.*
- *It is possible to introduce on-line auto-adaptation of the model splitting mechanisms to ensure stable performance despite variable distribution of cars in the modeled environment.*

1.3. Goals of this research

The principal goal of this research is to assess the viability of executing extensive simulations of urban traffic dynamics using a continuous model. A critical objective is to ascertain whether the algorithm adheres to super-scalability criteria, specifically scale invariance, which would facilitate achieving a nearly linear acceleration as the number of computational cores increases.

The research goals performed to confirm the thesis are:

– Scalability Assessment: Initially, the scalability of the existing SMARTS simulator was thoroughly evaluated to understand its current performance limits.

– Barrier Identification: enhancing scalability, this involved pinpointing specific aspects of the algorithm that obstruct optimal scaling and those that contradict the principles of super-scalability. This identification let to the final design of the algorithm.

– Algorithm design: Based on the insights gained from the previous steps, this phase was focused on formulating and enacting strategies to resolve the identified scalability and consistency issues.

– Tool Implementation: The newly designed algorithm and model partitioning strategies were implemented in the HiPUTS simulator, a parallel tool capable of handling large-scale microscopic traffic simulations. This implementation served as a proof of concept.

– Experimental Validation: The final phase of the research involved conducting rigorous experiments to test the improved scalability. These tests were carried out using the Ares supercomputer at the Cyfronet AGH Academic Computer Center, providing a robust platform for evaluating the enhancements made to the HiPUTS simulator.

Through these activities, the research aims to significantly advance the field of urban traffic simulation by developing more scalable and efficient computational approaches that can handle complex, large-scale urban environments.

1.4. Contributions presented in this thesis

This doctoral thesis presents several significant contributions to the field of urban traffic simulation, particularly through the development and implementation of the HiPUTS (High Performance Urban Traffic Simulator). The following are the key contributions made as part of my PhD research:

– Algorithmic Innovations: The thesis details innovative algorithmic developments that address and overcome limitations in existing traffic simulation tools. Specifically, designing new algorithms for parallel processing of this kind of computation within the simulation process allowed such scalability. A crucial aspect of these developments is that the proposed parallelization strategy preserves the correctness of the simulation results—ensuring that the outcomes of parallel executions are fully consistent with those of their sequential counterparts, despite the challenges introduced by synchronization across distributed nodes.

– Development of HiPUTS: This thesis introduces HiPUTS, the urban traffic simulation tool designed to handle large-scale simulations with high fidelity. This simulator represents a substantial advancement in the ability to model complex urban traffic patterns and interactions.

– Development of HiPUTS: As part of this thesis, the HiPUTS simulator was developed as a proof of concept for executing large-scale, high-fidelity urban traffic simulations in a parallel and distributed computing environment. HiPUTS was not intended to replace existing solutions tools but rather to demonstrate and validate the proposed algorithmic concepts, scalability strategies, and execution correctness in practice.

– Scalability Enhancements: A major focus of this research was on enhancing scalability by successfully developing and improving HiPUTS to optimize its performance and efficiency on supercomputers. This involved optimizing the algorithm to ensure scale invariance, allowing for super-linear speedups as computational resources were increased.

– Testing: The practical application of HiPUTS was tested through a series of simulations run on the Ares supercomputer at the Cyfronet AGH Academic Computer Center. These tests demonstrated the tool's enhanced capabilities and its effectiveness in simulating urban traffic scenarios.

– Publications and Collaborations: The findings and advancements achieved through this research have been shared in several conferences. As part of the work for this thesis, the collaboration with Technische Universität Berlin for 4 month internship has taken place, where parallelization of MATSim [56] took place.

Through these contributions, my thesis not only advances the technological foundations of traffic simulation through novel parallel algorithms, but also provides valuable insights and methodologies that can support urban planners and traffic engineers in improving urban environments.

1.5. Structure of this thesis

This thesis is structured into seven chapters, each of which builds upon the previous to systematically present the challenges, methodologies, implementations, and findings related to super-scalable urban traffic simulation using microscopic continuous models.

– **Chapter 1: Introduction** introduces the problem space and explains the motivation for this work. It outlines the classification of urban traffic models, discusses the computational demands of realistic simulations, and introduces the main thesis and goals of the research. It also defines key performance concepts such as scalability, speedup, and super-scalability.

– **Chapter 2: Scalability of existing urban traffic simulations** provides a comprehensive review of current traffic simulation models and techniques, with particular focus on agent-based modeling and microscopic approaches. It compares discrete and continuous models, introduces the notion of time-stepped versus event-driven simulations, and lays the theoretical groundwork for later architectural decisions.

– **Chapter 3: Parallelization of urban traffic simulation** discusses the core challenges and techniques for parallelizing traffic simulations. It focuses on model partitioning, synchronization, and dynamic load balancing. This chapter introduces essential algorithmic strategies, including a novel decomposition approach and decentralized control mechanisms that directly influence the system's scalability.

– **Chapter 4: Existing urban traffic simulators and scalability of dispersion** surveys well-known urban traffic simulation tools such as SUMO, MATSim, and SMARTS and others. It analyzes their capabilities, synchronization mechanisms, and scalability limits, highlighting gaps that the proposed HiPUTS system aims to address. This comparative analysis frames the motivations for the architectural and algorithmic decisions made in HiPUTS.

– **Chapter 5: HiPUTS (The High Performance Urban Traffic Simulator)** presents the central technical contribution of this work. It describes the conceptual design, data structures, simulation algorithm, and parallel execution architecture of HiPUTS. Detailed explanations of the components, patch-based environment model, decentralized execution scheme, and communication strategy are provided.

– **Chapter 6: Evaluation** validates the design through single-node and multi-node experiments on a high-performance cluster. Both weak (Gustafson) and strong (Amdahl) scalability are assessed on synthetic scenarios using toroidal networks. The results confirm the simulator's ability to scale to thousands of cores while maintaining computational efficiency and distribution transparency.

– **Chapter 7: Conclusions and further work** summarizes the core contributions of the research and reflects on the broader impact of the work. It also proposes a range of future research directions, including real-time simulation, integration with AI models, enhanced visualization tools, and potential extensions beyond the domain of traffic modeling.

This structure ensures a progression from foundational background to applied contributions and evaluation. Early chapters establish the theoretical and practical challenges of scalable urban traffic simulation, while the central chapters introduce and detail the novel HiPUTS architecture. The final chapters validate the method effectiveness and reflect on its broader implications. Together, these components form a cohesive narrative that demonstrates how high-fidelity, microscopic traffic simulations can be executed efficiently at HPC scale.

2 Urban traffic simulation models

This chapter provides an overview of essential concepts and definitions related to agent-based urban traffic simulations. It reviews current methods and solutions, which either underpin the research discussed in this work or illustrate the shortcomings of widely adopted techniques. Additionally, the objectives outlined in the preceding chapter are rephrased within the framework of agent-based simulations.

2.1. Agent-based modeling in microscopic traffic simulation

2.1.1. Definition of an agent

The fundamental concept underpinning the technologies discussed in this study is the *agent* [147, 122, 99]. Throughout its usage, the definition of an agent has varied, sparking numerous discussions. However, there is a general consensus on the following characteristics:

- Autonomy: The ability to operate independently and make decisions to achieve its goals.
- Interactivity: The capacity to interact with other agents.
- Perception: The capability to observe the environment and respond accordingly.
- Persistence: Continuous presence in the environment, rather than being activated on demand.

Despite this generally accepted "middle ground" definition, the concept of an agent can serve different purposes depending on the modeling context. In simulation systems, agents may take on a variety of roles. One common usage is to represent an individual from the real world—such as a person, vehicle, or device—within a modeled environment. In this role, the agent functions as a container for the attributes and behaviors assigned to that individual. These attributes may include physical characteristics, goals, or preferences, and are typically enriched with decision-making logic that imitates real-world behavior. As a result, the agent becomes a dynamic entity whose internal state evolves in response to the simulation context.

Another use of the agent concept appears in distributed simulations, where an agent may correspond to a computational process responsible for managing a portion of the simulation. These agents operate over allocated resources, maintain their own local state (e.g., a fragment of the model), and collaborate with other processes to update the global state in accordance with defined rules. They exhibit autonomy, communication capabilities, and responsiveness to environmental changes—traits that align with the general definition of software agents.

These examples highlight the versatility of the agent concept and the need to clarify its use within this work. In the context of this thesis, the primary focus is on the first type: agents as modeled individuals endowed with decision-making logic. This interpretation aligns with agent-based modeling approaches in microscopic traffic simulation, where each vehicle or driver is represented as an autonomous decision-making unit within the system.

In light of the above distinctions and for the purposes of this thesis, the following definition of an agent is adopted:

An agent is a program or part of a program that operates continuously within a specified environment, capable of observing the environment, interacting with other agents, and making autonomous decisions.

2.1.2. Agent-based modeling

While beneficial for architectural design and artificial intelligence, the agent concept has been most impactful in computer simulation. This approach, known as *agent-based modeling and simulation* (ABMS)[24, 109], has garnered significant interest since its inception. ABMS has proven valuable in diverse research areas, such as modeling economic behavior changes[54] and studying phenomena in two-party elections [72].

The strengths of ABMS are particularly evident in micro-scale models where each individual entity is represented by an agent. These models remain excellent tools for investigating discrete model based scenarios in road traffic simulation [135], pedestrian dynamics [112], urban design [104], building evacuation [102], etc. By simulating the actions and interactions of autonomous agents, researchers can gain insights into the complex behaviors and emergent phenomena that arise in these environments.

A key feature of all these applications is the need to represent diverse individuals with unique attributes, internal states, or decision-making logic. This makes it possible to account for the diversity seen in real-world populations. For example, in ecological simulations [103, 21], agents might represent different species—such as rabbits and patches of lettuce—each with distinct behaviors, movement patterns, and resource consumption strategies. This level of granularity allows the model to capture complex population dynamics and emergent behaviors that arise from local interactions between agents.

Traditional simulations based on differential equations [125, 127, 113, 81], though effective for continuous physical phenomena, often lack the expressiveness required for such applications. Differential equations typically assume a level of uniformity and predictability that does not capture the variability and adaptability of individual agents. Consequently, ABMS has become indispensable in certain research domains where such granularity is crucial.

However, employing ABMS involves considerable computational resources, as each entity in the simulation must be assigned an agent. This requirement can lead to significant increases in computational load, especially as the number of agents and the complexity of their interactions grow. The factors contributing to increased computational costs in simulations, discussed in Chapter 1, apply to ABMS as well. These factors include the need for high processing power, memory consumption, and efficient algorithms to manage and update the states of numerous agents.

As the size and complexity of agent modeling grow, exploring parallelization possibilities becomes essential. Parallel computing can distribute the computational load across multiple processors, significantly reducing simulation time and allowing for more detailed and extensive models. By leveraging advances in parallel computing, researchers can overcome some of the limitations imposed by computational resource demands, enabling more ambitious and large-scale simulations that can provide deeper insights into the systems being studied. Agent modeling offers powerful tools for simulations and understanding complex systems, it also presents challenges related to computational demands. Addressing these challenges through techniques such as parallelization is crucial for advancing the field and fully realizing the potential of agent-based simulations in various research domains.

2.2. Urban Traffic modeling

Urban traffic modeling is a critical field of study that addresses the complexities of vehicular flow within city environments. As urban populations continue to grow, the demand for efficient and effective transportation systems becomes increasingly paramount. The primary goal of urban traffic modeling is to understand, predict, and manage traffic patterns to enhance mobility, reduce congestion, improve safety, and minimize environmental impact.

Urban traffic systems are inherently complex due to the dynamic interactions between a multitude of heterogeneous vehicles, including cars, buses and other means of transport. Each of the agents operating specific means of transport exhibits unique behaviors and preferences, contributing to the overall traffic flow and influencing the efficiency of the transportation network. Traditional traffic modeling techniques, often based on macroscopic approaches such as differential equations [127, 125], can fall short in capturing the nuanced behaviors and interactions of individual agents. Consequently, more sophisticated methodologies, such as agent-based modeling and simulation (ABMS) mentioned in Section 2.1.2, have gained prominence.

Agent-based modeling and simulation provides a fine-grained approach to urban traffic modeling by representing each vehicle as an autonomous agent with specific characteristics and decision-making capabilities. This approach allows for the simulation of complex interactions and emergent phenomena that arise from the collective behavior of these agents. For instance, ABMS can effectively model scenarios such as traffic congestion, road accidents, and the impact of infrastructure changes on traffic flow [10, 137].

Despite its advantages, urban traffic modeling using ABMS also presents significant challenges. The high level of detail required for agent-based models demands substantial computational resources, especially for large-scale simulations involving thousands or millions of agents. Furthermore, accurately capturing the diverse behaviors and interactions of agents necessitates comprehensive data collection and sophisticated algorithms. Advances in parallel computing are crucial in addressing these challenges, enabling more scalable and accurate traffic simulations.

2.2.1. Classification of models in the context of detail representation

In urban traffic modeling and simulation, the level of detail represented in the models plays a crucial role in determining their applicability, performance, and accuracy. Different modeling approaches are typically classified based on the granularity of detail they incorporate—ranging from macroscopic models, which offer a broad overview, to microscopic models, which provide highly detailed representations of individual agent behaviors. As illustrated in Figure 2.1, microscopic models tend to offer the most detail but require the highest computational effort. These models often represent time and space continuously, whereas macroscopic and mesoscopic models may use discrete abstractions. Since this is a broad classification, model characteristics such as spatial and temporal resolution are treated here as tendencies rather than strict rules. Understanding this classification helps researchers and practitioners select appropriate models that align with specific research objectives and computational constraints.

1. **Macroscopic Models.** These models focus on aggregate properties of traffic flow, treating the system as a continuous fluid rather than a collection of discrete entities. They are particularly useful for analyzing large-scale traffic patterns and long-term trends, as they abstract away the behavior of individual vehicles or pedestrians. Key variables include average speed, vehicle density, and traffic flow rate. Due to their simplicity and low computational cost, macroscopic models are suitable for city-wide traffic planning, infrastructure development, and policy evaluation. Prominent examples include the Cell Transmission Model (CTM) [82], the Two-Fluid Model [52], and the Lighthill–Whitham–Richards (LWR) model [108].

2. **Mesoscopic Models.** These models provide a middle ground between macroscopic and microscopic approaches by incorporating some level of individual behavior while still grouping vehicles or road segments

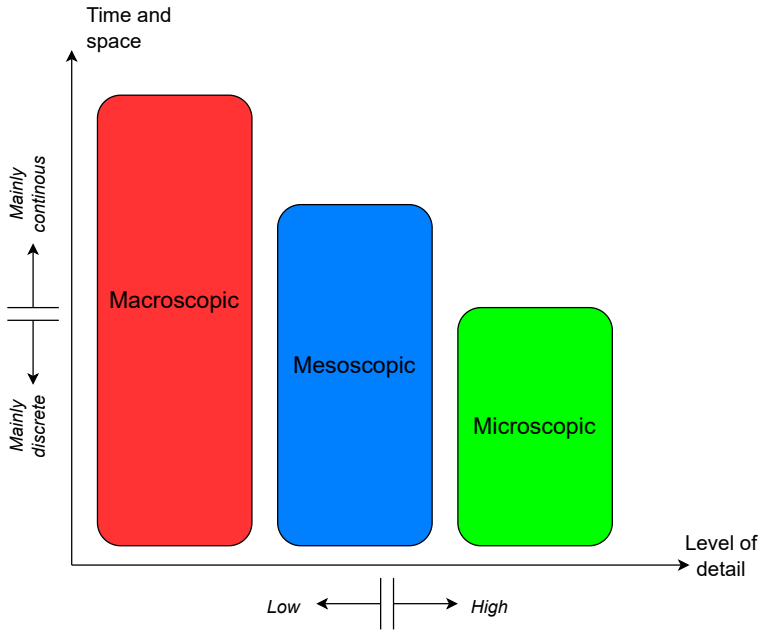


Figure 2.1: Categorization of traffic models based on their level of detail and the representation of time and space.

for efficiency. Mesoscopic models simulate interactions in traffic flow more accurately than macroscopic ones but remain computationally less demanding than full-scale microscopic models. They are particularly effective for corridor-level analysis or sub-network optimization and are commonly used in regional planning and traffic management studies. A widely used tool in this category is DYNASMART [84], Dynasmart-IP [84] or AIMSUN [8].

3. **Microscopic Models.** These models offer the highest level of detail by simulating the behaviors and interactions of individual agents such as vehicles, pedestrians, or cyclists. Each agent is represented with distinct attributes and decision-making processes, allowing the model to capture complex, emergent phenomena such as lane changes, overtaking, and responses to signals or congestion. Microscopic models are particularly valuable when accurate behavioral fidelity is essential. However, this level of detail requires significant computational resources, especially for large-scale scenarios. Applications include intersection-level traffic analysis, pedestrian flow studies, intelligent transportation systems (ITS), and testing of autonomous vehicle algorithms. A variety of simulation platforms have been developed to implement microscopic traffic models, including SUMO (Simulation of Urban MObility) [83], MATSim [56], and VISSIM [31]. These tools provide practical environments for applying microscopic modeling techniques in both research and real-world planning. Further technical aspects of microscopic models are discussed in Section 2.4.

4. **Hybrid Models.** These models combine elements from macroscopic, mesoscopic, and microscopic paradigms to create adaptive and flexible modeling frameworks [86]. Hybrid models are particularly useful in urban environments that demand both broad-scale planning and localized behavioral accuracy. They may simulate general traffic flow with macroscopic dynamics while using microscopic sub-models in areas of high complexity or interest (e.g., near intersections or evacuation zones). This approach allows for better computational efficiency while preserving fidelity where it matters most. Hybrid models are especially well-suited for scenario-based planning, emergency response simulation, and multi-scale transportation system analysis. Examples include models that integrate microscopic traffic simulation with macroscopic flow

models and multi-resolution models that dynamically adjust the level of detail based on simulation needs. A dynamic highway-focused case by Imai et al. (2024) demonstrates region-specific resolution switching—between macro and micro—validated on real merge sections [58]. Burghout et al. (2005) developed a hybrid framework combining MITSIMLab (micro) and Mezzo (meso), ensuring boundary consistency [17].

The classification of urban traffic models based on their level of detail is essential for selecting the right tool for a given task. Each model type serves a specific purpose, aligned with the goals and constraints of a given study. Macroscopic models are ideal for system-wide planning and long-term forecasting, where aggregated traffic trends are more important than individual behavior. Mesoscopic models provide a good compromise between performance and behavioral fidelity, making them suitable for corridor-level traffic analysis or regional impact studies. Microscopic models enable the study of fine-grained interactions among individual agents, offering insights into detailed traffic dynamics and local phenomena. Hybrid models offer scalable, context-sensitive flexibility, combining multiple model types to balance realism and computational efficiency across different network segments.

Among these, microscopic models offer the most comprehensive view of heterogeneous traffic systems—especially when simulating diverse agent types such as private vehicles, buses, bicycles, and pedestrians. Each agent type exhibits distinct behaviors, from driving dynamics to signal responsiveness. For example, drivers may differ in reaction time, aggressiveness, or route choice preferences, while pedestrians vary in walking speed, alertness, and decision logic. Accurately modeling this diversity is critical for producing realistic simulations but introduces significant algorithmic and computational complexity. Capturing these variations often necessitates the use of sophisticated algorithms and detailed real-world behavioral data, underscoring the value—and cost—of high-fidelity modeling.

2.2.2. Space representation and complexity

In urban traffic modeling and simulations, accurately representing the physical space of urban environments is crucial for capturing the complexities of traffic dynamics. The spatial representation forms the foundation of the model, influencing how traffic flow, interactions, and behaviors are simulated. Different methods of space representation offer varying levels of detail and computational efficiency, each suitable for different types of traffic analysis needs.

1. **Geometric Models.** They focus on the precise depiction of the physical layout of urban environments, including roads, intersections, lanes, and pedestrian pathways. These models use geometric shapes and coordinates to map out infrastructure elements, enabling detailed simulations of vehicle maneuvers such as lane changes, turns, and parking. The high level of precision in geometric models makes them ideal for studies requiring fine-grained analysis of traffic interactions and localized traffic phenomena [130].

2. **Network Models.** These models abstract the urban space into a graph-like structure where nodes represent intersections or key points, and edges represent roads or pathways connecting these points [57]. This abstraction simplifies the computational complexity by focusing on the connectivity and flow between different parts of the urban environment. Network models are particularly useful for analyzing traffic patterns over larger areas and for strategic planning, such as route optimization and traffic management at a city-wide scale.

3. **Cellular Automata Models.** These are discrete models that divide the urban space into a grid of cells, each representing a small segment of the environment [94]. They are effective for simulating the movement of vehicles and pedestrians in a rule-based, local-interaction manner. Their simplicity allows for computationally efficient simulations while still capturing emergent behaviors such as congestion formation and crowd dynamics.

4. **Hybrid Models.** These models combine elements of geometric, network, and cellular automata approaches to leverage the strengths of each [110]. For instance, a hybrid system might use detailed geometric representation for intersections and merging points while applying network abstraction for long-range connectivity and cellular automata for pedestrian zones. This multi-layered approach balances computational efficiency with the fidelity required for localized analysis, making hybrid models suitable for complex, multi-scale simulations.

Each model type serves a different analytical purpose, and the selection of an appropriate space representation method should be closely aligned with the goals and scale of the simulation. Geometric models are most appropriate when high-fidelity spatial accuracy is required, such as in detailed studies of intersections, turning behaviors, or autonomous vehicle navigation. Their precision supports complex maneuver modeling but often comes at the cost of increased computational demand.

Network models, by contrast, offer a more abstract and computationally efficient means of representing urban environments, making them highly suitable for large-scale traffic studies, strategic route planning, and infrastructure optimization. These models trade spatial granularity for the ability to simulate broader connectivity and system-wide behaviors.

Cellular automata models provide a rule-based, discretized alternative, enabling fast simulations of localized interactions with relatively low overhead. They are especially effective for simulating pedestrian movement, evacuation scenarios, and simple vehicle dynamics, where emergent behaviors such as congestion or crowd formation are of interest.

Hybrid models aim to integrate the strengths of the aforementioned approaches by combining detailed spatial fidelity where needed with abstracted or simplified representations elsewhere. For instance, high-resolution geometric models can be deployed in traffic-critical zones such as roundabouts or merging lanes, while network-based or cellular representations can be used for highway sections or pedestrian zones. This flexibility allows researchers and planners to build scalable, context-sensitive simulations that remain computationally feasible while retaining accuracy in the most relevant regions.

Ultimately, the choice of space representation has significant implications for both the performance and the interpretability of the simulation. A well-matched modeling approach ensures that the simulation accurately reflects the physical and operational dynamics of urban traffic systems while respecting resource constraints and research objectives.

2.2.3. Discrete and Continuous Models

Another highly significant classification of traffic models pertains to the nature of the independent variables used to describe the system. Traffic models fundamentally aim to capture the evolution of transportation systems by analyzing changes in state, space, and time. These independent variables determine the structure and behavior of the model and can be classified as either continuous or discrete. Continuous variables provide a smooth, uninterrupted representation, while discrete variables segment the system into distinct units or states, enabling detailed analysis at a finer level of granularity.

This classification is critical because the choice between continuous and discrete independent variables directly influences the model's accuracy, complexity, and computational requirements. As a result, the classification presented in Table 2.1 clearly distinguishes between these different approaches. It highlights the flexibility of traffic models to adapt to various scales and complexities of transportation systems.

The classification in those terms is multilayered, so for the purpose of this work below definition for simplified naming of continuous and discrete models is used:

State & Space	Time	Description
Discrete	Discrete	Cellular Automata: Traffic, pedestrians, Land use, Urban sprawl
	Continuous	Discrete Event Systems
Continuous	Discrete	Car-following models, Microscopic traffic flow models
	Continuous	Partial Differential Equation (PDE): Traffic flow models, Pedestrian models

Table 2.1: Types of Simulation in Transportation

– **Continuous models**: consider both space and state as continuous variables, treating the physical environment as a seamless domain without predefined divisions. These models use real-valued coordinates to describe locations and movements, enabling smooth transitions and unrestricted positioning within the system. In terms of space, continuous models provide a precise and fluid representation, while for state, they depict the system using variables that evolve continuously over time. This approach effectively captures gradual changes and dynamic variations in system behavior, making it ideal for analyzing processes with continuous and uninterrupted dynamics [14, 4].

– **Discrete models**: divide the physical environment into distinct, non-overlapping units, such as grid cells or zones. Each unit represents a specific location, and movement or interaction occurs between these predefined units. For example, in traffic models, roads may be divided into segments, or in pedestrian models, movement may be constrained to a lattice grid. Discrete models describe the system using a finite set of predefined states or conditions. At any given time, an entity (e.g., a vehicle or pedestrian) occupies one of these states, and changes occur as transitions between these states [94]. In many discrete models, not only spatial position but also other physical variables—such as speed or acceleration—are represented as discrete quantities, limited to a set of predefined values. This simplifies computations and aligns well with rule-based simulation frameworks, although it may reduce the precision of modeling physical dynamics.

2.3. Categorization of the factor driving simulations

Urban traffic simulation is a complex and multi-faceted field that requires robust and efficient modeling techniques to represent the dynamic nature of traffic systems accurately. Two primary paradigms dominate the simulation landscape: time-stepped simulations and event-driven simulations. Understanding the differences and implications of these approaches is crucial for selecting the most appropriate method for specific research and practical scenarios.

2.3.1. Time-Stepped Simulation

Time-stepped simulation, also known as discrete-time simulation, operates by dividing time into discrete, uniform intervals of portion of time called time steps. At each time step, the simulation progresses by updating the state of the system simultaneously for all entities according to predefined rules and equations. This approach mirrors the continuous evolution of real-world systems by sequentially processing each time step, ensuring a synchronized advancement of events across the entire simulation environment [19].

The methodology of time-stepped simulation involves:

– Initialization: Setting initial conditions and parameters for all entities and the overall system.

– Time Step Advancement: Progressing the simulation clock by a fixed time increment, usually chosen to balance accuracy and computational efficiency.

– State Update: Calculating the new state of each entity based on the current state and any interactions or dynamics prescribed by the model.

– Iteration: Repeating the time step advancement and state update until the simulation reaches a predefined endpoint or condition.

Time-stepped simulations are widely used in urban traffic studies due to their straightforward implementation and ability to model continuous processes. Typical applications in the domain of traffic simulation include Traffic Flow Analysis, which enables modeling vehicles' movement on road networks to understand congestion patterns and optimize traffic management strategies. Another application is pedestrian dynamics, where simulating pedestrian behavior and interactions in urban spaces to improve safety and infrastructure design. Last example could be usage in designing public transportation systems, where it is required to analyze the operation of buses, trams, and other public transport modes to enhance efficiency and service quality [87].

The primary advantages of time-stepped simulations lie in their simplicity and their suitability for continuous monitoring. The uniform progression of time steps simplifies the implementation and comprehension of the simulation model. Given the same initial conditions, time-stepped simulations yield repeatable and predictable results, aiding in model validation and comparison. Furthermore, this approach allows for continuous monitoring of the system state, which is useful for real-time applications and control mechanisms.

However, time-stepped simulations also face several limitations. High-resolution simulations with small time steps can be computationally expensive, particularly for large-scale urban traffic models. Managing numerous entities and interactions over many time steps can lead to a decrease in performance, affecting scalability. Additionally, the use of fixed time steps may miss critical transient events or interactions occurring between steps, potentially affecting accuracy.

2.3.2. Event-Driven Simulation

Event-driven simulation, also known as discrete-event simulation, models systems where changes in state are triggered by discrete events occurring at specific points in time. Unlike time-stepped simulations, event-driven models do not advance time uniformly. Instead, the simulation clock progresses to the time of the next event, which then triggers an update of the system state and the scheduling of subsequent events.

The methodology of event-driven simulation involves several stages. The simulation starts with the definition of initial conditions and the scheduling of the first events. The next step involves identifying and processing the next scheduled event, updating the system state accordingly. Following this, future events are determined and scheduled based on the updated state and system dynamics. This process of event processing and scheduling is repeated until the simulation reaches a termination condition or time horizon.

Event-driven simulations are particularly well-suited for scenarios where changes in the system state are sporadic and event-dependent. Key applications in the domain of traffic simulations include traffic incident management, where the impact of accidents, road closures, and other incidents on traffic flow is modeled to evaluate response strategies. Signalized intersections are another common application, with event-driven simulations used to simulate the operation of traffic lights and their interactions with vehicle queues to optimize signal timings. Adaptive traffic control systems, which implement and test adaptive traffic signal control algorithms that respond dynamically to real-time traffic conditions and events, also benefit from the event-driven approach [59].

The primary advantages of event-driven simulations include their efficiency, scalability, and dynamic flexibility. By focusing on relevant events and skipping idle periods, event-driven simulations use computational resources more effectively. They are better suited for large-scale simulations involving many interacting entities, as the computational effort is concentrated on significant events. Moreover, event-driven simulations allow for adaptive and responsive scenarios that can handle unexpected events and changes in traffic conditions.

Despite these advantages, event-driven simulations also face several challenges. Their implementation tends to be more complex due to the need for efficient event scheduling, queuing, and management of a potentially large number of events—especially in dense traffic scenarios where interactions occur frequently. Maintaining and updating the event queue becomes computationally expensive as the simulation grows in scale and resolution.

Furthermore, while event-driven simulations allow for precise modeling of asynchronous interactions, this can make the system's behavior harder to interpret or debug, especially when tracing causality across a chain of events. It is important to note that any variability or non-repeatability of results across simulation runs is typically a consequence of stochastic elements within the model itself, rather than being inherently caused by the event-driven simulation method. Both time-stepped and event-driven simulations can yield deterministic or non-deterministic results depending on the design of the underlying model.

2.3.3. Impact on performance and scalability

Performance is a critical factor in choosing between time-stepped and event-driven simulations. Time-stepped simulations can become computationally intensive, especially when small time steps are required to ensure accuracy in large-scale systems. Event-driven simulations, by contrast, optimize computational efforts by processing only the moments when significant changes occur, making them more efficient for systems with sparse and irregular state changes.

Event-driven simulations often provide higher temporal accuracy in scenarios where events are infrequent but impactful. They can precisely capture the timing and influence of individual actions, such as traffic incidents or signal changes. Time-stepped simulations, while easier to implement and analyze, may miss critical transitions occurring between discrete steps, potentially affecting simulation fidelity.

However, event-driven simulations present unique challenges when deployed in distributed or parallel computing environments. Synchronizing events across different machines becomes a non-trivial task, especially when events on one node can causally affect events on another. To ensure causal consistency, distributed event-driven systems often require conservative or optimistic synchronization protocols, such as the Chandy–Misra–Bryant algorithm [124] or Time Warp mechanism [38]. These techniques introduce communication overhead and can diminish the scalability benefits of the event-driven approach if not carefully managed. The need to maintain a consistent global event order—especially under high inter-node interaction—can offset the local efficiency gained through selective event processing.

The choice between time-stepped and event-driven simulations therefore depends not only on the temporal characteristics of the system being modeled but also on the architectural and deployment context. Time-stepped simulations are advantageous for applications requiring continuous monitoring and synchronous updates, such as dynamic traffic assignment and real-time control. Event-driven simulations excel in reactive and sparse-event scenarios, such as incident modeling or adaptive signal control, but require sophisticated synchronization mechanisms to scale effectively in distributed environments.

2.4. Microscopic Traffic Flow Models

Microscopic traffic flow models represent a fundamental component of traffic flow theory, focusing on the individual behavior of vehicles and drivers within a traffic stream. These models provide a detailed description of traffic dynamics by simulating the interactions between vehicles, enabling researchers and practitioners to study the micro-level processes that govern traffic flow. Unlike macroscopic models, which describe traffic as a continuous flow, microscopic models emphasize the discrete, agent-based interactions that contribute to traffic phenomena.

This section explores the major types of microscopic traffic flow models classified based on review literature [55, 138, 134, 9, 39, 14, 88], highlighting three commonly studied subcategories: Car-Following Models, Lane-Changing Models, and Cellular Automata (CA) Models. Each category addresses specific aspects of traffic dynamics, offering unique perspectives and methodologies for simulating driver and vehicle behavior.

2.4.1. Cellular Automata (CA) Models

Cellular Automata (CA) models, initially introduced by John von Neumann [141], represent a powerful computational framework for modeling dynamic systems. Their simplicity, adaptability, and computational efficiency have made them particularly valuable in the field of traffic flow modeling. By discretizing both space and time, CA models offer an efficient way to simulate the behavior of individual vehicles on road networks while capturing emergent macroscopic traffic phenomena such as congestion and stop-and-go waves [50].

CA models are characterized by their use of a grid-based structure, where the road is divided into discrete cells of uniform size, typically approximating the length of a single vehicle. Each cell can either be empty or occupied, and vehicle movements are governed by simple transition rules that account for acceleration, deceleration, and randomness in driver behavior. These discrete-time, discrete-space models are capable of reproducing realistic traffic dynamics with relatively low computational cost, making them ideal for large-scale simulations and real-time applications.

Cellular Automata (CA) models are built upon four core components:

1. **Physical Environment**: The road network is represented as a series of cells, each of equal length. In most applications, the cell size is approximately the length of a vehicle (e.g., 7.5 meters), ensuring that a single cell corresponds to the space occupied by one vehicle. Time is discretized into steps, typically of one second, allowing for periodic updates to the system's state.

2. **Cell States**: Each cell can exist in one of two states:

   – **Empty (0)**: No vehicle occupies the cell.

   – **Occupied (1)**: A vehicle is present in the cell.

3. **Neighborhoods**: The behavior of a vehicle is influenced by the states of some of its neighboring cells (e.g. some cells in front and next to it). These interactions determine vehicle speed and movement while ensuring safety by maintaining adequate gaps between vehicles.

4. **Local Transition Rules**: Vehicle behavior, including acceleration, deceleration, and motion, is determined by deterministic and probabilistic rules. These rules are applied iteratively to update the state of each cell.

Cellular Automata (CA) models are widely recognized for their computational efficiency [135], enabling rapid simulations of large-scale traffic networks. Their discrete structure supports scalability, allowing the simulation of thousands of vehicles without significant computational demands. Despite their simplicity, CA models effectively capture emergent traffic phenomena such as congestion patterns and traffic waves. Furthermore, their inherent flexibility allows adaptation to specific traffic conditions and the inclusion of additional features, making them suitable for various applications.

However, the simplicity of CA models also introduces limitations. The use of uniform cell sizes and discrete time steps can result in oversimplified representations of real-world traffic dynamics, failing to account for continuous vehicle movements and variations in vehicle size. Basic CA models lack the capacity to fully represent driver behavior, vehicle dynamics, and external influences such as weather or road conditions, leading to limited realism. To address these challenges, CA models often require integration with more detailed frameworks, such as

microscopic or agent-based models, to achieve greater fidelity in simulations. This balance between efficiency and realism defines their utility in traffic modeling.

Discrete Nagel and Schreckenberg Model

The Discrete Nagel and Schreckenberg (NaSch) model [94] is a cellular automaton model that simplifies the simulation of traffic flow on highways. Introduced by Kai Nagel and Michael Schreckenberg in 1992, the NaSch model represents vehicles as cells on a one-dimensional grid where each cell can either be empty or occupied by a vehicle.

Each vehicle is characterized by its discrete position (cell) and an integer-valued velocity v, which denotes the number of cells it intends to move forward during the next time step. The velocity is bounded by a predefined maximum value v_{max}, and both position and velocity are updated synchronously for all vehicles at each discrete time step. The model operates based on discrete time steps and follows a set of simple rules:

1. **Acceleration**: If the distance to the vehicle ahead is greater than the current velocity of the vehicle, the vehicle accelerates by one unit of speed, up to a maximum speed v_{max}.

2. **Deceleration**: If the distance to the vehicle ahead is less than or equal to the current velocity, the vehicle decelerates to avoid a collision, setting its speed to the gap size.

3. **Randomization**: With a probability p, the vehicle may randomly decelerate by one unit to account for driver imperfections and random behavior.

4. **Movement**: Each vehicle moves forward by its current velocity.

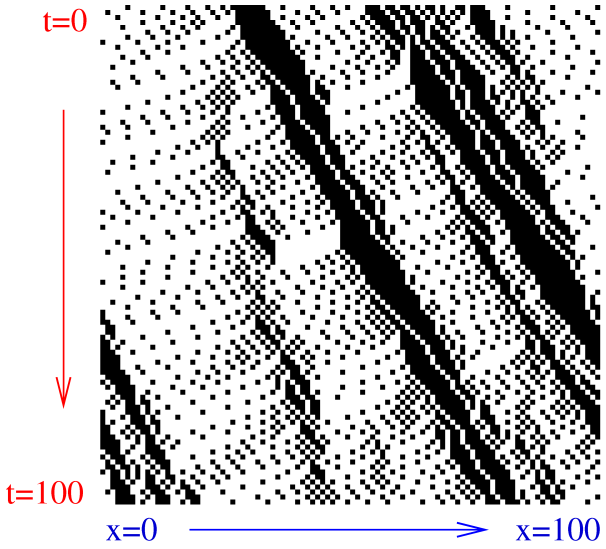


Figure 2.2: A road with jams of cars. Each horizontal line of pixels shows the state of a 100-cell road at a specific moment in time. Black pixels indicate the presence of cars, while white pixels mark empty spaces. Moving from top to bottom, each line corresponds to the road's condition at the next time step. Source: [121]

These simple rules can generate complex traffic patterns, including free flow, synchronized flow, and traffic jams as shown in the Figure 2.2. The NaSch model is particularly valued for its ability to reproduce the fundamental diagram of traffic flow, which relates traffic density to flow. The fundamental diagram shows three distinct phases: free flow, where vehicles move at maximum speed; synchronized flow, where vehicles move in clusters at reduced speed; and jammed flow, where vehicles come to a complete stop due to high density.

The model's simplicity allows for easy implementation and scalability, making it suitable for large-scale traffic simulations. However, it also has limitations, such as the inability to accurately capture individual driver behavior and interactions in multi-lane traffic. Despite these limitations, the NaSch model remains a cornerstone in traffic flow theory due to its foundational insights and ease of use.

Extensions and Applications To address its limitations and apply the model to more realistic urban scenarios, numerous extensions of the NaSch model have been proposed. These include:

- **Multi-lane and lane-changing models**: Rickert et al. (1995) extended NaSch to two-lane traffic with overtaking rules [114]. Knospe et al. (2002) further refined lane-change behavior to reproduce empirical lane usage inversion [71].
- **Traffic signals and intersection modeling**: Brockfeld et al. (2001) combined NaSch with the Biham–Middleton–Levine model on a grid and embedded traffic light controls, leading to emergent patterns and optimized signal timing strategies [16].
- **Urban network CA models**: Vranken et al. (2020) developed a CA intersection model simulating multi-lane flows, turning, and signal control with empirical validation [142].
- **Pedestrian extensions**: Adaptations of the cellular automata framework derived from NaSch principles have also been used to model pedestrian flows [155, 117].

These adaptations demonstrate the versatility of the NaSch model: by incorporating lane changes, traffic signals, multi-modal flows, and network structures, researchers can apply it to urban environments, intelligent traffic control, and large-scale simulations—making it a robust point of departure for advanced traffic modeling.

2.4.2. Car-Following Models

Car-following models describe the longitudinal dynamics of vehicles, focusing on how drivers adjust their speed and spacing based on the vehicle ahead. These models are essential for understanding traffic flow at a microscopic level, capturing driver perception, reaction, and decision-making processes.

Early models, such as Wiedemann's[53], relied on psychophysical principles, emphasizing drivers' perception thresholds. Another early model is named Follow the Leader model introduced by Gazis, Herman and Rother [40]. These models incorporate visual cues like the apparent size and changes in size of the leading vehicle, reflecting how humans process visual information. This approach provides a realistic representation of driver behavior, especially in dynamic traffic scenarios. There many car following models that can be basically divided into 4 categories: Gazis–Herman–Rothery or stimulus– response models [40, 74, 22, 69, 148], safety-distance or collision-avoidance models [73, 78, 98, 25, 43, 151], reference-signal models [51, 49, 132, 7, 77, 61, 44, 105], and models that involve human factors [11, 30, 29, 146, 37, 79, 18, 48, 126, 62].

Car-following models are critical for simulating traffic dynamics, analyzing congestion patterns, and assessing safety. They form the foundation of integrated traffic models that combine longitudinal and lateral interactions. Widely used in intelligent transportation systems and autonomous vehicle algorithms, these models are indispensable tools for developing safer and more efficient traffic systems.

Modern models, such as the Intelligent Driver Model (IDM) [132, 131] and extensions of the Helly model [80] from group of reference-signal models, include human factors like risk-taking, delayed reactions, and distractions. They also integrate principles like prospect theory, simulating drivers' responses to potential collisions under uncertainty, thereby enhancing their applicability to real-world conditions.

Among the many available car-following approaches, IDM model is discussed here in more detail due to its widespread adoption in traffic simulation tools and autonomous vehicle control systems. Its balance of realism, computational efficiency, and flexibility in capturing diverse traffic phenomena has made it particularly popular in both academic research and practical applications.

IDM - Intelligent Driver Model

The Intelligent Driver Model (IDM) is a continuous car-following model that describes the behavior of drivers in terms of acceleration and deceleration. Developed by Treiber, Hennecke, and Helbing, IDM is designed to replicate real-world driving behavior more accurately than simpler models. Unlike discrete models like NaSch described in section 2.4.1, IDM operates in continuous time and space, making it more suitable for detailed microscopic simulations.

The acceleration $a(t)$ of a vehicle in IDM is given by:

$$a(t) = a_{\text{max}} \left(1 - \left(\frac{v(t)}{v_0} \right)^{\delta} - \left(\frac{s^*(v, \Delta v)}{s(t)} \right)^2 \right) \tag{2.1}$$

where:

- $v(t)$ is the current velocity of the vehicle,
- v_0 is the desired velocity (the speed the driver wishes to travel at),
- a_{max} is the maximum acceleration,
- δ is the acceleration exponent (typically set to 4),
- $s(t)$ is the current gap to the vehicle ahead,
- $s^*(v, \Delta v)$ is the desired minimum gap, defined as:

$$s^*(v, \Delta v) = s_0 + vT + \frac{v\Delta v}{2\sqrt{a_{\text{max}} b}} \tag{2.2}$$

where:

- s_0 is the minimum distance (jam distance),
- T is the safe time headway,
- b is the comfortable deceleration,
- Δv is the relative velocity to the vehicle ahead.

IDM captures several key aspects of driving behavior, such as the tendency to maintain a safe distance from the vehicle ahead and the smooth adjustment of speed to match traffic conditions. The model's parameters can be calibrated to reflect different driving styles, from aggressive to conservative.

One of the strengths of IDM is its ability to handle a wide range of traffic scenarios, from free flow to congested conditions. It also forms the basis for more complex models that incorporate additional behaviors, such as lane changing and interaction with traffic signals. However, IDM requires more computational resources than simpler models, which can be a limitation in large-scale simulations.

2.4.3. Lane-Changing Models

Car-following models (described in the previous section 2.4.2) primarily address how vehicles regulate their speeds and positions relative to those ahead, capturing only the longitudinal dynamics of traffic flow. In contrast, lane-changing models focus on lateral interactions, describing when and why drivers shift lanes. Although both aspects are fundamental to microscopic traffic theory, historically they have been considered in isolation. Car-following behavior has been extensively researched for decades, while lane-changing received serious attention only more recently [128, 91, 156], as studies began to underscore its pronounced effects on safety and congestion.

Evidence suggests that lane-changing maneuvers can heighten driver stress, increasing the likelihood of errors and collisions. They also contribute to capacity drops at bottlenecks and the emergence of stop-and-go waves. In fact, recent research [3] shows that lane-changing can transform minor disturbances into widespread traffic oscillations. Over the last decade, efforts to model lane-changing have accelerated, producing a range of approaches. Some focus on decision-making—determining [152] when a driver opts to change lanes—while others investigate the broader impacts of these decisions on surrounding vehicles [153]. A fully integrated tool that captures both the decision-making process and the resulting effects on traffic dynamics, in conjunction with car-following behaviors, remains elusive.

Lane-changing decisions often depend on a driver's willingness to accept available gaps. This gap acceptance process, typically represented through stochastic functions, reflects whether drivers find a particular space sufficient to accommodate their maneuver. Early lane-changing models, such as Gipps's approach [43], employed deterministic rules in urban contexts, emphasizing desired speeds and correct lane positioning before turns. Subsequent models categorized maneuvers as mandatory or discretionary [154], introduced probabilistic elements to better mirror real behavior, and experimented with utility theory [120], Markov processes [106], and fuzzy logic [144] to refine the representation of these complex choices.

Yet many models overlook the consequences that one vehicle's lane change has on the traffic around it. While some research links lane-changing to key traffic phenomena—breakdowns, capacity reductions, and oscillations—there is still a need to incorporate these feedback effects more thoroughly. Ultimately, refining lane-changing models to account for both individual decision-making and broader traffic-level impacts will foster more accurate and robust microscopic traffic simulations.

MOBIL - Minimizing Overall Braking Induced by Lane Changes

Among the many proposed lane-changing models, MOBIL (Minimizing Overall Braking Induced by Lane changes) stands out as one of the most widely adopted and operationally effective. Developed by Kesting, Treiber, and Helbing [67], MOBIL has gained popularity due to its elegant balance between driver utility, safety, and traffic-wide impact, making it particularly suitable for integration into large-scale microscopic simulations. Just as the IDM model became a reference point for longitudinal vehicle behavior, MOBIL is frequently chosen as a lateral dynamics counterpart because of its generality, simplicity, and compatibility with car-following models like IDM.

Lane changing is a complex decision that involves balancing the benefits of moving to a faster lane with the potential disruption to surrounding vehicles. The MOBIL model uses a utility function to evaluate the desirability of a lane change, considering the acceleration of the ego vehicle (the vehicle considering the lane change), the following vehicle in the current lane, and the following vehicle in the target lane. A lane change is executed if it increases the overall utility, subject to safety constraints to avoid dangerous maneuvers.

The utility function U can be expressed as:

$$U = \Delta a_{\text{ego}} + p \cdot (\Delta a_{\text{oldFollower}} + \Delta a_{\text{newFollower}}) \tag{2.3}$$

where:

– Δa_{ego} is the acceleration gain of the ego vehicle after the lane change,

– p is the politeness factor, which represents the weight given to the impacts on other drivers,

– $\Delta a_{\text{oldFollower}}$ is the change in acceleration of the vehicle that was following the ego vehicle in the old lane,

– $\Delta a_{\text{newFollower}}$ is the change in acceleration of the vehicle that will follow the ego vehicle in the new lane.

The safety constraint ensures that the new acceleration of the following vehicles remains above a threshold to prevent collisions:

$$a_{\text{newFollower}} > -b_{\text{safe}} \tag{2.4}$$

where b_{safe} is a safe deceleration threshold.

The MOBIL model balances the benefits of lane changing with the potential negative impacts on surrounding traffic, promoting smoother and safer traffic flow. It considers the politeness factor, which accounts for the cooperative behavior of drivers. A higher politeness factor indicates that the driver is more considerate of the impact on other vehicles, while a lower factor suggests a more selfish driving style.

Incorporating MOBIL into traffic simulations allows for a more realistic representation of multi-lane traffic, where lane changes are a frequent and critical aspect of driving. The model can be combined with car-following models like IDM to create comprehensive simulations that account for both longitudinal and lateral vehicle dynamics.

This chapter presented a comprehensive overview of modeling approaches used in urban traffic simulation, emphasizing the agent-based paradigm and its applications across different levels of model granularity. It explored the conceptual foundations of agent-based modeling, detailed the classification of traffic models by resolution and spatial representation, and examined both discrete and continuous simulation methods. The discussion highlighted the trade-offs between model fidelity and computational efficiency, as well as the growing role of parallelization in enabling large-scale simulations. These insights lay the groundwork for selecting and adapting appropriate modeling techniques in the context of modern urban mobility challenges.

3 Parallelization of urban traffic simulation

The increasing computational demands of urban traffic simulations—driven both by performance constraints and the growing size and complexity of models that can no longer fit within the memory or processing limits of a single machine—have led researchers to explore parallelization techniques. This includes distributing computations across multiple processors or systems to improve scalability and efficiency. In this chapter, the focus is placed on microscopic traffic models represented as graphs, where the road network is abstracted as a set of nodes and edges, and each vehicle is simulated individually as an agent. Within this scope, numerous studies have proposed parallelized implementations of traffic simulation frameworks [56, 93, 63, 75, 101, 70].

These studies reveal that parallelizing detailed microscopic traffic models poses considerable challenges. A widely adopted approach across most implementations is to spatially partition the graph-based road network into disjoint fragments, with each fragment assigned to a separate computational unit. These units compute their respective sections in parallel, while synchronization mechanisms ensure consistency at the borders—where vehicles or signals might interact across partitions. This synchronization typically involves exchanging boundary state data at each simulation step to maintain the correctness of vehicle behavior and traffic flow across fragment interfaces.

However, this partitioned approach introduces its own complexities. As vehicles move through the network, they frequently cross partition boundaries, requiring data transfer and state updates between processes. This not only increases communication overhead but also leads to fluctuating computational loads, creating imbalances that demand dynamic load balancing strategies. While domain decomposition and border synchronization remain the standard, some studies have explored alternative approaches, such as functional parallelism or hybrid methods, though these are less commonly used in practice.

The simulation generates a substantial amount of data, which poses significant challenges for centralized processing and storage, particularly as the scale of the simulation increases. Furthermore, the computational load distributed to each worker is highly dynamic, shifting over time as vehicles navigate through different areas of the road network. This variability necessitates adaptive strategies to balance the workload effectively and ensure optimal performance across the computing system. Without such strategies, bottlenecks and inefficiencies can arise, further complicating the parallelization process.

This chapter discusses the challenges and existing methods for parallelizing graph-based microscopic urban traffic simulations, which require substantial computational power and efficient synchronization. The primary difficulties include managing dynamic workloads, ensuring synchronization of shared traffic states across nodes, and addressing the high data volumes generated. Graph partitioning methods, including vertex and edge partitioning, are used to divide road networks into balanced subgraphs while minimizing edge cuts. Dynamic load balancing, essential for adapting to fluctuating traffic conditions, employs techniques like area redistribution and task transfer, with approaches for stabilization such as the PID algorithm offering predictive and adaptive balancing. These strategies aim to improve scalability and accuracy while minimizing synchronization overhead, enabling reliable and efficient distributed traffic simulations.

3.1. Model Partitioning

This section will present several formal definitions from graph theory to clearly define the symbols and concepts used throughout this work. Additionally, it will address the formalization of graph partitioning and explain why the commonly used definition in related works is inadequate for modeling division in large-scale concurrent urban traffic simulations.

3.1.1. Basic definitions and notations of graph division

Definition 1: A directed graph is a pair $G = (V, E)$, where V is a finite, nonempty set of elements known as vertices, and E is a set that is a subset of the Cartesian product $E \subseteq V \times V$, with its elements referred to as edges. Edges are denoted as (u, v), where $u \in V$ and $v \in V$.

Definition 2: An edge entering a vertex $v \in V$ in the graph $G = (V, E)$ is any edge of the form $(u, v) \in E$.

Definition 3: An edge leaving a vertex $v \in V$ in the graph $G = (V, E)$ is any edge of the form $(v, u) \in E$.

Definition 4: The weight function for edges in the graph $G = (V, E)$ is defined as:

$$w : E \to \mathbb{R}^+ \cup \{0\} \tag{3.1}$$

The weight of an edge $e \in E$ is given by its value $w(e)$.

Definition 5: The weight function for vertices in the graph $G = (V, E)$ is defined as:

$$c : V \to \mathbb{R}^+ \cup \{0\} \tag{3.2}$$

The weight of a vertex $v \in V$ is given by its value $c(v)$.

Definition 6: Given an undirected graph $G = (V, E)$, where V represents the set of vertices and E represents the set of edges, along with a weight function defined as:

$$w : E \to \mathbb{R}^+ \cup \{0\} \tag{3.3}$$

A set $P = \{G_1, G_2, \ldots, G_n\}$ is called a partition of the graph G if the following conditions are met:

1. Each vertex $v \in V$ is included in exactly one subgraph G_i, meaning:

$$\forall_{v \in G}\ v \in G_i \Rightarrow \forall_{j \neq i}\ v \notin G_j \tag{3.4}$$

2. The union of all subgraphs G_i is equal to the original graph G:

$$\bigcup_{i \in \{1,\ldots,n\}} G_i = G \tag{3.5}$$

3.1.2. Classic division of a graph into subgraphs

In the literature, the problem of partitioning a graph into k subgraphs is typically approached as a minimization problem. Two additional criteria are often considered for the desired partition: minimizing the cut size and maintaining balance among the subgraphs.

Problem 1: Consider a directed graph $G = (V, E)$ with a weight function defined on the vertices c and edges w. To find a partitioning $P = \{(V_i, E_i) \mid i \in \{1, \ldots, k\}\}$ into k subgraphs we can define the set:

$$C = \{(u, v) \in E \mid u \in V_i, v \in V_j, 0 \leq i < j \leq k - 1\} \tag{3.6}$$

as the set of cut edges. The size of such a cut is given by:

$$\theta(P) = \sum_{(u,v)\in C} w(u, v) \tag{3.7}$$

The problem of partitioning into k partitions with a balancing factor ($1 \leq \epsilon < k$) can be defined as an optimization problem minimizing the function $\theta(P)$ while maintaining constraint:

$$0 < \sum_{u\in V_i} c(u) \leq \lfloor \frac{\epsilon}{k} \sum_{v\in V} c(v) \rfloor \tag{3.8}$$

3.1.3. Types of graph division

The traditional graph partitioning problem is often referred to in the literature as "vertex partitioning." In such method the goal is to allocate vertices to subgraphs such that the number of edges connecting vertices in different subgraphs is minimized. This approach, known as "edge-cut," defines the boundaries between subgraphs by edges.

A less common but relevant method for this work is known as "edge partitioning" or "vertex-cut." In this method, edges are assigned to subgraphs, and the boundaries run along vertices.

This concept is illustrated in Figure 3.1.

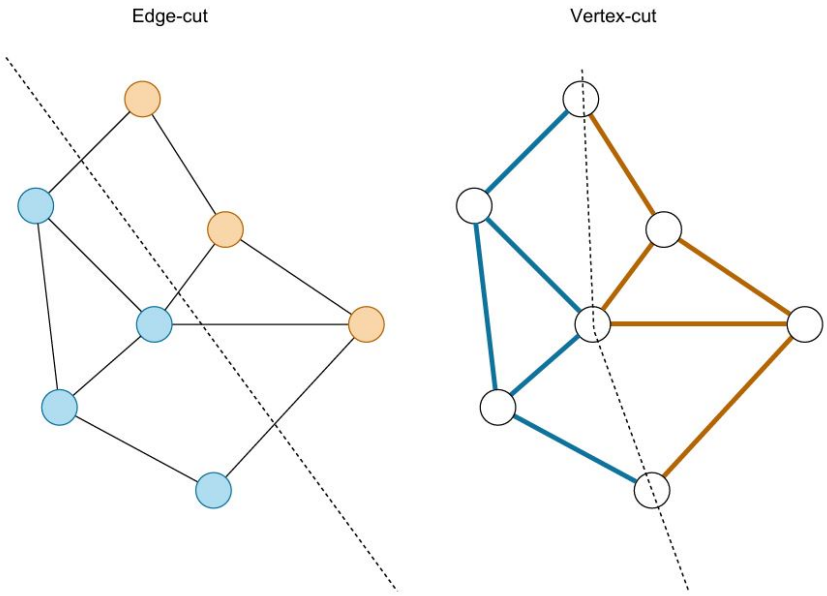


Figure 3.1: The difference between edge partitioning and vertex partitioning methods presented in graphical form

The type of partitioning discussed here is similar to "edge partitioning" but also involves assigning vertices. It can be defined as follows:

Definition 7: Given a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, and a finite set of partition identifiers P. A partition consists of two mappings:

$$\begin{aligned} f_v &: V \to P \\ f_e &: E \to P \end{aligned} \tag{3.9}$$

subject to the conditions:

$$\begin{aligned} &\forall p \in P, \exists e \in E \text{ such that } f_e(e) = p \\ &\forall v \in V, \exists e \in E \text{ such that } v \in e \text{ and } f_e(e) = f_v(v) \end{aligned} \tag{3.10}$$

Given this definition, the partitioning problem needs to be adjusted. In this work, the problem of division is defined as follows:

Problem 2: Consider a directed graph $G = (V, E)$ with vertex weight function c and edge weight function w. The aim is to find a partition $P = \{(V_i, E_i) \mid i \in 1, \ldots, k\}$ into k subgraphs. Define the set of cut edges as:

$$C = \{(u, v) \in E \mid f_v(u) = i \text{ and } f_v(v) = j,\ 0 \leq i < j \leq k - 1\} \tag{3.11}$$

The size of the cut is given by:

$$\theta(P) = \sum_{(u,v)\in C} w(u, v) \tag{3.12}$$

The problem of partitioning into k subgraphs with a balance factor can be formulated as an optimization problem that minimizes $\theta(P)$ while satisfying the constraint:

$$0 < \sum_{u\in V_i} c(u) + \sum_{e\in E_i} w(e) \leq \lfloor \frac{\epsilon}{k} \left(\sum_{u\in V} c(u) + \sum_{e\in E} w(e) \right) \rfloor \tag{3.13}$$

This formulation reflects a hybrid approach that combines elements of both vertex and edge partitioning. It ensures that each edge and its incident vertices are assigned to the same computational unit, which is essential for maintaining data locality in parallel simulations. The introduced optimization objective $\theta(P)$ quantifies the total communication cost induced by cut edges, while the balance constraint ensures that the workload is evenly distributed across partitions, accounting for both vertex and edge weights.

As a result, this formalization captures the core trade-off in graph-based traffic simulation parallelization: minimizing inter-process communication (via edge cuts) while maintaining computational balance across nodes. The structure of the problem naturally aligns with distributed agent-based simulations, where both vehicle agents (situated on edges) and infrastructure elements (nodes) must be efficiently distributed across the system.

Crucially, finding such a partitioning that satisfies the constraint described in the final equation is essential for optimizing the model decomposition. Without meeting this criterion, the resulting simulation may suffer from excessive communication overhead or uneven workload distribution, both of which degrade overall performance and scalability.

3.2. Existing methods of graph partitioning

The graph partitioning problem is commonly addressed using heuristic methods to achieve reasonable computation times. While this approach can result in reduced solution quality, the trade-off is often acceptable because the heuristic solutions are typically only marginally less accurate than exact solutions. Consequently, this chapter will explore both exact and heuristic methods, with a stronger focus on heuristic approaches.

Model	Constraints	Variable	Binary variables	Non null coefficients
A	$\|V\| + 3\|E\|K + K$	$\|V\|K + \|E\|K$	$\|V\|K$	$2\|V\|K + 7\|E\|K$
B	$\|V\| + 2\|E\|K + K$	$\|V\|K + \|E\|$	$\|V\|K$	$2\|V\|K + 6\|E\|K$
B2	$2\|E\| + K$	$\|V\| + \|E\|$	$\|V\|$	$\|V\|K + 6\|E\|$
C	$\|V\|(1 + K + K\log(K))$ $+2\|E\|\log(K) + K$	$\|V\|(K + \log(K))$ $+\|E\|$	$\|V\|\log(K)$	$\|V\|(\log(K) + 2K + 3K\log(K))$ $+6\|E\|\log(K)$
D	$\|V\|(1 + 3(K - 1 - \log(K)))$ $+2\|E\|\log(K) + 2K$	$\|V\|(K - 1)$ $+\|E\| + K$	$\|V\|\log(K)$	$\|V\|(2\log(K) + 8(K - 1 - \log(K)))$ $6\|E\|\log(K) + K(K + 1)$

Table 3.1: Reduction models. Source: [13]

3.2.1. Linear programming

While approximate solutions are generally adequate, it is still valuable to consider exact algorithms. Though exact methods partition large graphs just as effectively as heuristics, the solutions they provide can help evaluate and enhance heuristic approaches. An example of such an exact algorithm is the linear programming-based algorithm discussed in [13]. This approach proposes five methods (A, B, B2, C, D) to reduce the graph partitioning problem to a set of inequalities with unknowns in the domain of natural numbers. The number of inequalities and unknowns varies depending on the reduction method, as detailed in Table 3.1.

Model A is the basic model and serves as the foundation for the others. Model B2 is essentially a simplification of Model B for partitioning into two subgraphs; however, it allows a single vertex to belong to multiple partitions. The model most closely related to the problem under consideration is Model D, which operates in the case of partitioning into k partitions. Unfortunately, the main drawback of exact algorithms based on the specified models is their computational complexity.

The authors compared Models B and D for graphs with several dozen vertices, several dozen edges, and a few target clusters. They demonstrated that Model B performs significantly better than Model D, and for Model B, they showed that the processor time required to compute a solution grows *extremely quickly* with the number of clusters.

Even the computationally simplest of the presented models (B2) required 970 seconds to bisect a graph with 500 vertices into 10 partitions. When the number of vertices in the graph doubled, the computation time increased to 3480 seconds.

The overall study proves that it is possible to find an exact solution for relatively small graphs within an acceptable timeframe. However, there is substantial evidence that this time will very quickly become unacceptable as both the size of the graph and the number of target partitions increase.

3.2.2. Heuristic methods

The majority of heuristic graph partitioning techniques rely on two fundamental components: an algorithm for creating the initial partition, which serves as the starting solution, and an algorithm for enhancing the partition through local optimizations, which can be iteratively applied. Examples of both types of algorithms will be discussed later in this work. It is essential to emphasize these foundational elements for understanding the subsequent description of traditional heuristics.

Algorithms constructing the initial division The quality of the initial division may affect the convergence speed of the entire division algorithm, therefore the most trivial of the algorithms is not used in practical applications. However, it is impossible not to mention that it is the simplest. The method is to assign the vertices/edges of the graph to the partitions in a completely random way, e.g., by making one pass through the graph and, for each

element, selecting one of the k available partitions with a uniform distribution. Although the obtained solution will most likely be very different from the exact one, its undoubted advantage is its time complexity of $O(|V| + |E|)$.

Locally improving division algorithm. In the article [66], an algorithm for the case of graph bisection was presented as one of the possibilities for constructing the initial partition, but it can be generalized for the case of division into k partitions. The idea behind the algorithm is to search the graph in a depth-first manner and add all encountered vertices to the current partition until the number of $|V|/k$ visited vertices is exceeded. A problematic issue is the selection of the vertex from which to start the search - in [66] the author suggests performing the algorithm 10 times and selecting the initial division that generates the smallest edge-cut. The above-mentioned generalization for the problem involves performing $k - 1$ iterations of the algorithm - for each partition separately.

3.2.3. Multi-level methods.

The graphs with a small number of vertices are manageable, even with exact algorithms. Building on this observation, the method described in article [64] involves reducing the graph by merging vertices until it is small enough to be handled by an exact algorithm within a reasonable time frame. Subsequently, an iterative graph unification procedure is performed, during which local optimization algorithms can be applied at each stage. This approach is known as the multilevel graph partitioning method. Article [64] established the foundation for the METIS graph partitioning tool, which remains widely used and valued today.

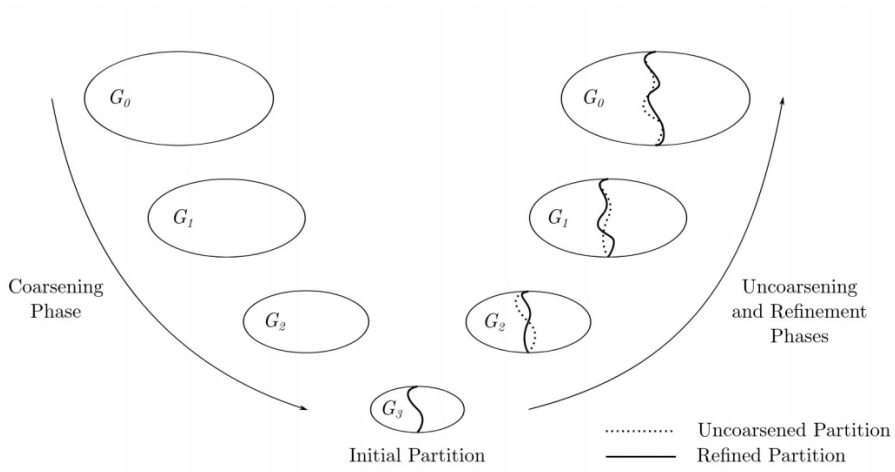


Figure 3.2: The principle of operation of multi-level methods. Source: [143]

Following the success of this method, it was generalized in article [143], which identifies four phases of the multilevel partitioning algorithm:

– Coarsening Phase

– Initial Partition

– Uncoarsening

– Local Improvements (Refinement Phases)

The latter two phases are typically performed alternately during the iterative unmerging process. Figure 3.2 illustrates the operational diagram of such an algorithm for dividing a graph into two subgraphs.

The authors of [143] suggest that new algorithms can be developed using this framework to enhance performance compared to traditional heuristic methods, provided that suitable operators for each phase can be defined.

3.2.4. Division methods profiled for the division of the road network

The earlier sections discussed general strategies for developing graph partitioning methods. However, this study focuses specifically on graphs representing road networks. The partitioning of these graphs has a practical application: to facilitate the distribution of traffic simulation computations for faster processing. As a result, algorithms designed for traffic simulators might differ somewhat from the more general methods previously described.

RGG. The first example of such an algorithm is the Neighbor Restricting Graph Growing method described in [150]. The authors observed that in parallel urban traffic simulation, the synchronization communication overhead is influenced by both the size and the number of transmitted messages. The latter can be reduced by minimizing the number of neighborhood relations in a divided map. The proposed algorithm ensures that when partitioning into k areas, there will be at most $k + 1$ neighborhood relations, meaning each area will have at most two adjacent areas. This was achieved with a simple modification of the Graph-Growing algorithm also described in [150].

The starting vertex is chosen based on the direction of the vector along which the partition will occur (provided as a parameter to the algorithm) - the vertex furthest in this direction is selected. Then, vertices are added to the current area, considering the coordinate along the specified vector, effectively partitioning the map into "strips" perpendicular to this vector. However, this partitioning method may result in overly narrow strips, which could require accessing non-adjacent areas to gather information about a vehicle's surroundings, thereby increasing communication overhead. To mitigate this, a minimum "width" for such strips could be established, although this would significantly limit the number of areas into which a map can be divided.

Recursive bisection. Another example of an algorithm focused on partitioning road networks is the one described in [145], which uses recursive bisection with vertical and horizontal lines as the cutting directions. The algorithm always selects the largest area for division and repeats the process until the desired number of areas is obtained. While this method does not ensure a balanced partition for every number of areas, the authors of the paper conducted tests using divisions into multiples of two to obscure this limitation.

Nonetheless, the authors made an insightful observation regarding the metric for assessing expected load. They demonstrated that the length of a road is not always proportional to its load and proposed a method for estimating load based on the routes vehicles will take. However, this approach assumes that these routes are known during the partitioning process, which may not be feasible for large-scale concurrent urban traffic simulations.

Geometric division. The previously presented algorithm takes advantage of the fact that the road network has an obvious geometric representation. Therefore, one should not overlook the fact that the road graph can be divided solely based on this property, as it was done by the authors of the article [70]. This division does not guarantee either the balancing of areas or the minimization of the cutting length, but it allows you to quickly divide the road graph in a time proportional to the number of vertices. The simulator presented in the mentioned article was tested for a very small number of computing nodes (maximum 4), therefore this type of algorithm was sufficient to show the possibility of accelerating calculations, which was probably the main goal of the authors.

Evolutionary approach. Given the complexity of the problem, the literature also includes attempts to utilize evolutionary algorithms for solving the partitioning issue [89]. One significant advantage of this method is the ability to perform multi-criteria optimization. However, as the authors themselves note, for road network divisions with typical conditions, it is generally not challenging to consolidate multiple quality assessment criteria into a single minimization problem. Evaluating the effectiveness of this methodology is difficult, as the referenced article does not provide comparisons with any standard algorithms beyond the evolutionary approach.

3.2.5. Grid Partitioning for Complex Environments

Simulating urban traffic involves more than balancing computational load. It requires adapting to highly irregular and fragmented environments. Urban maps often include non-navigable terrain such as buildings or water bodies, constrained passages like intersections, and varying road densities. Standard graph partitioning methods—designed for uniform domains—fall short when faced with such spatial and structural complexity.

To address these limitations, a domain-specific multi-level partitioning method was developed and presented in an earlier work [157]. This approach enhances partitioning quality in grid-based simulations where space is discretized and agent behavior is sensitive to environmental constraints. It extends traditional multilevel coarsening schemes (such as those used in METIS [66] or Party [90]) with several key innovations tailored for urban-scale traffic simulation.

The first innovation lies in the treatment of *excluded* and *indivisible areas*. These are portions of the simulation domain that either cannot be entered (e.g., buildings) or should not be divided across partitions (e.g., doorways or narrow corridors). Rather than simply assigning zero weight to such regions—a common practice that often leads to degraded partition quality—the algorithm either removes them entirely or aggregates them into atomic units to ensure integrity throughout the partitioning process.

During the coarsening phase, a modified Lightweight Approximate Matching (LAM) [107] algorithm is used, incorporating a dynamic discount factor. This factor helps prevent heavily weighted vertices (such as aggregated indivisible regions) from dominating early matches, which maintains load balance across partitions. This is particularly critical in environments with unevenly usable or fragmented space.

In the refinement phase, the algorithm applies an enhanced Helpful Set-based method [118] to incrementally restore the graph to full resolution. By introducing consistency checks and avoiding scattered or disconnected regions, the algorithm produces compact, contiguous, and communication-efficient partitions that better conform to real-world spatial layouts.

Another key feature is its hierarchical design for deployment on high-performance computing (HPC) architectures. The algorithm supports partitioning into $m \times k$ regions, where m is the number of compute nodes and k the number of cores per node. After an initial fine-grained division, these partitions are aggregated into blocks aligned with the physical layout of the computing cluster. This dramatically reduces inter-node communication, which is typically much more costly than intra-node messaging.

Empirical validation of this method on both synthetic and real-world urban maps confirms its effectiveness. The approach reliably respects spatial constraints, reduces synchronization overhead, and yields more communication-local partitions compared to conventional methods.

This grid partitioning strategy is the basis in the patch construction and distribution mechanisms within this work, which is discussed in detail in Section 5.4. Its integration ensures scalable execution of traffic models across large-scale parallel systems, while preserving the fidelity and realism of the simulation environment.

3.3. Synchronization and correctness

3.3.1. Challenges

Distributed traffic simulations face key challenges in maintaining consistent shared states, avoiding temporal mismatches, and addressing synchronization effectively. Errors at partition boundaries, asynchronous updates, and overlooked synchronization can lead to unrealistic results, while existing solutions like rollbacks are often impractical for large-scale systems.

Some of the key challenges that can be distinguish in context of synchronization are:

– Dependence on Shared State

– Temporal Inconsistencies

– Ignored or Overlooked Synchronization

Traffic simulations inherently depend on the accurate and timely representation of the state of nearby vehicles, traffic signals, and environmental conditions. These interdependencies are vital for modeling realistic traffic dynamics, such as car-following, lane-changing, and intersection control. In distributed systems, the traffic state is typically divided among computational nodes, with each node responsible for simulating a specific region or partition of the road network. While this partitioning improves scalability and reduces computational overhead per node, it introduces significant challenges in maintaining the consistency and synchronization of the overall system.

One of the most critical issues arises at the delineated boundaries where vehicles move between partitions managed by different nodes. In such cases, delays or errors in exchanging state information can result in inconsistencies. For example, a vehicle transitioning from one node to another may momentarily exist in both partitions or in neither, leading to artifacts such as non-physical accelerations, collisions, or gaps in traffic flow. These errors are compounded in dynamic traffic scenarios where the load and density vary significantly across the network, requiring frequent updates and recalculations.

Moreover, the synchronization problem extends beyond individual vehicle states. It also impacts aggregated behaviors, such as the propagation of congestion waves or the coordination of adaptive traffic control systems. For instance, a delay in updating the state of a traffic signal at a partition boundary could disrupt vehicle flow, causing artificial bottlenecks or misaligned traffic patterns. These synchronization challenges are further exacerbated in large-scale simulations involving heterogeneous traffic conditions, multimodal transportation, or interactions with pedestrian and cyclist flows.

Ensuring accurate state transitions between partitions is critical, as even minor discrepancies can propagate across the network, amplifying over time and resulting in unreliable simulation outputs. Such errors undermine the credibility of the simulation, particularly when used for traffic planning, optimization, or real-world decision-making. Addressing this issue requires robust synchronization mechanisms that not only align updates between nodes but also do so efficiently to avoid excessive communication overhead and performance degradation.

Despite its importance, synchronization between partitions is often underestimated or insufficiently addressed in distributed traffic simulations. Many approaches [129] simplify the problem by assuming low error rates or negligible delays, which can lead to unrecognized inaccuracies in the results. Advanced techniques, such as predictive modeling or asynchronous updates with eventual consistency, attempt to mitigate these issues but often introduce additional complexity or computational costs. Consequently, synchronization remains a key challenge in distributed traffic simulations, highlighting the need for innovative solutions that balance accuracy, scalability, and efficiency.

3.3.2. Implications of Problems

Failure to adequately address these challenges has significant implications for the reliability and utility of distributed traffic simulations. One of the most immediate consequences is the emergence of model artifacts. Synchronization errors, whether due to delayed updates or temporal mismatches, can propagate across the network, creating unrealistic phenomena. Phantom traffic jams, where congestion appears without a plausible cause, and unnatural gaps in traffic flow are common examples. Vehicles may behave erratically, accelerating or decelerating in ways that defy real-world dynamics. These artifacts distort the simulation's representation of traffic patterns, making it unsuitable for analyzing real-world scenarios or testing potential interventions.

Beyond technical inaccuracies, these issues also lead to reduced credibility. Traffic simulations are often used to inform critical decisions in urban planning, traffic management, and policy-making. When simulations produce inaccurate results, they can mislead decision-makers, resulting in flawed interventions such as misplaced infrastructure investments or ineffective traffic control measures. For example, a simulation may overestimate the benefits of a new signal timing strategy or underestimate the impact of a proposed road design change due to synchronization errors. Over time, such inaccuracies can erode trust in simulation tools, discouraging their use and limiting their impact on real-world decision-making.

Finally, synchronization problems contribute to inefficiency in distributed simulations. Retrospective corrections, such as rollbacks or iterative updates, are commonly used to resolve errors after they occur. However, these methods are computationally expensive and introduce significant overhead, negating the performance benefits of parallelization. Large-scale simulations, which rely on efficient distributed execution to handle complex networks, are particularly vulnerable to these inefficiencies. The additional computational load required for corrections reduces scalability and may prevent the simulation from running in real-time, limiting its applicability for time-sensitive applications like traffic monitoring or emergency response planning.

3.4. Load balancing

Load balancing can be categorized into two main types. The first type is static load balancing, where the problem is distributed among the computational nodes only at the start, with no changes to task distribution during execution. Given the nature of agent simulations, dynamic load balancing is essential. This approach involves adjusting the distribution of tasks after the initial allocation, allowing for the transfer of tasks or re-division of the area in response to changes in computational load due to the movement of agents.

3.4.1. Balancing method approach

To achieve load balancing in agent simulations, the following actions can be implemented:

– Re-divide the area.

– Transfer a fragment of the area between nodes.

Each method has its own set of advantages and disadvantages. Re-dividing the area during the simulation, as partly described in the NRGG algorithm [150], leverages the current distribution of agents to initially divide the map into lanes. This approach's main advantage is that it limits neighborhood changes and maintains area consistency during the simulation. However, it has significant drawbacks, such as potentially reassigning vastly different areas to a worker, leading to high communication overhead due to the need to transfer information about all agents on the map in the worst case.

The second method, more commonly used, involves transferring a fragment of the area between nodes. An experiment demonstrating this approach was detailed in a study [35]. Its main benefit is the relatively small number of messages required for load balancing, as only the computational cell (e.g., a street) and the agents within it need to be transferred. The major drawbacks include a potential increase in the number of neighboring nodes and the creation of additional connections between already connected computational nodes. For example, if five streets connect areas A and B, transferring an intersection might result in six or more connections. Additionally, this dynamic transfer can lead to a loss of area coherence within the computational node, increasing the number of messages transferred and diminishing the benefits of parallelism. Despite these disadvantages, this approach remains the most frequently implemented method for load balancing.

3.4.2. Determining the moment to start the balancing process

During the simulation process, it is essential to synchronize the computational nodes. This topic is explored in detail in article [42]. A simulation step cannot proceed correctly if data from adjacent cells are needed for accurate computation. Without load balancing, or with infrequent balancing, there will likely be a situation where out of N computational nodes, N-1 machines are waiting for one machine to complete its computation. It is nearly impossible to eliminate the waiting time for computational nodes when synchronization barriers are present. Figure 3.3 compares execution times with and without load balancing.

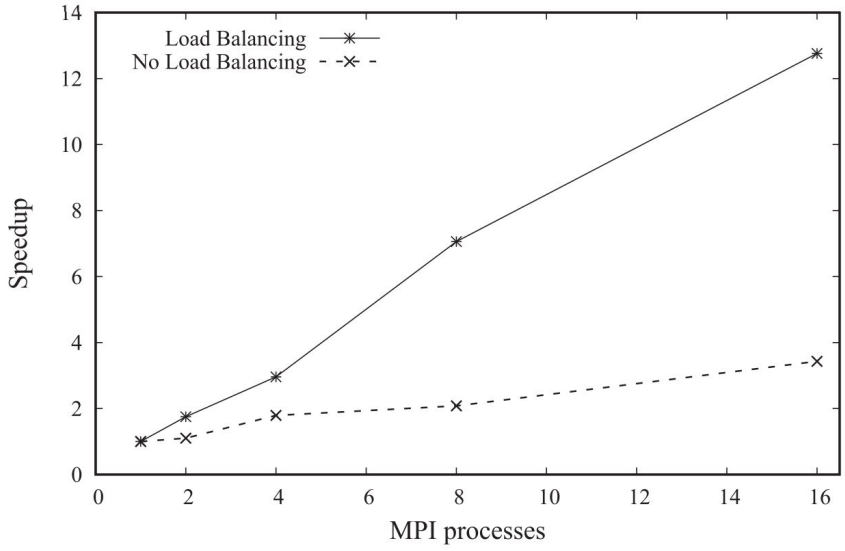


Figure 3.3: Comparison of simulation speedup with and without balancing Source: [42]

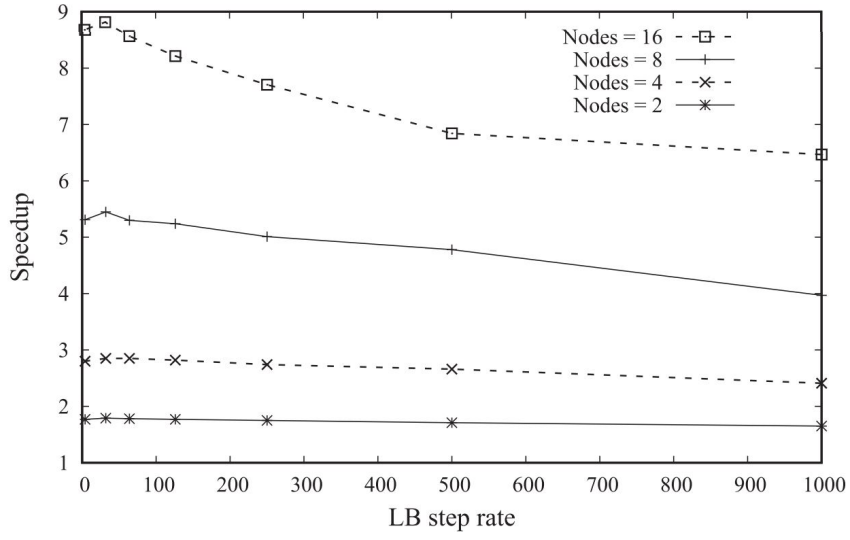


Figure 3.4: The impact of load balancing frequency on algorithm acceleration Source: [42]

The authors of article [42] also conducted an experiment to examine the effect of load balancing frequency on simulation speed. They tested frequencies of 4, 32, 64, 125, 250, 500, and 1000 iterations. The results, shown in Figure 3.4, indicate a drop in simulation performance for frequencies of 4 and 32. This decline is attributed to the excessive communication overhead caused by the negotiation process and the transfer of simulation cells

between nodes. Nonetheless, the authors assert that increased load balancing frequency generally enhances system efficiency.

Contrarily, the authors of "The Plasma Simulation Code" [41] suggest that synchronization is most effective when the imbalance exceeds at least 4%, or about every 500-1000 iterations, corresponding to a time step of about 200 ms.

Traditional load balancing methods initiate the balancing process after a specified number of iterations. This approach's advantage is that it prevents excessive cell transfers in response to temporary load increases. However, this advantage can become a drawback if balancing starts during a peak load moment.

More advanced algorithms calculate the computational cost—typically a function based on iteration execution time—and compare it at each simulation step. If the calculated value surpasses a set threshold, or if the execution time difference between the fastest and slowest nodes exceeds a certain percentage, the load balancing mechanism is activated. Although the balancing process is computationally intensive, the authors caution against frequent area exchanges [41, 42]. This solution is highly sensitive to short-term load variations, which increases the number of area transfers. The system should be more adaptable and respond more quickly to load changes. However, it must synchronize load information each time before deciding to balance, which does not eliminate the drawback of reacting to real-time changes.

3.4.3. PID algorithm

The PID (proportional-integral-derivative) algorithm is predominantly utilized in industrial applications. Initially, it was implemented to assist in ship control [46]. Today, it is used in over 90% of industrial controllers, managing variables such as pressure, flow rate, chemical composition, force, speed, and other signals. A common practical example of the PID algorithm is cruise control in vehicles, which maintains a constant speed. Another application is in drones, where it ensures stabilization in space. There have also been attempts to adapt the algorithm for simulations of particles in space.

The load balancing strategies discussed in this chapter primarily focus on the current load conditions, without considering that these might be temporary fluctuations. An approach that could address these short-term variations is the PID algorithm—proportional-integral-derivative algorithm. This algorithm, which is extensively used in industrial controllers and detailed described above and in book [6], can be represented by the following formula:

$$T_p\varepsilon + \frac{1}{T_i}\int_0^t \varepsilon dt - T_d\frac{\delta\varepsilon}{\delta t} \tag{3.14}$$

Where:

– $T_pT_iT_d$ - proportional, integral, differential gain coefficients.

– ϵ - input signal

– t - times

Each component of the presented formula serves a distinct purpose. The proportional part, - $T_p\varepsilon$ -, reacts to current changes in the simulation load, effectively replacing the previous metrics.

The differentiating part, - $\frac{\delta\varepsilon}{\delta t}$ -, helps to predict future changes based on the received signal, providing a forward-looking perspective.

The integrating part, commonly referred to as attenuation, represented by - $\int_0^t \varepsilon dt$ -, analyzes past readings. This component helps counteract temporary upward or downward trends in the load of a specific computational node. The coefficients T_p, T_i, and T_d are determined experimentally.

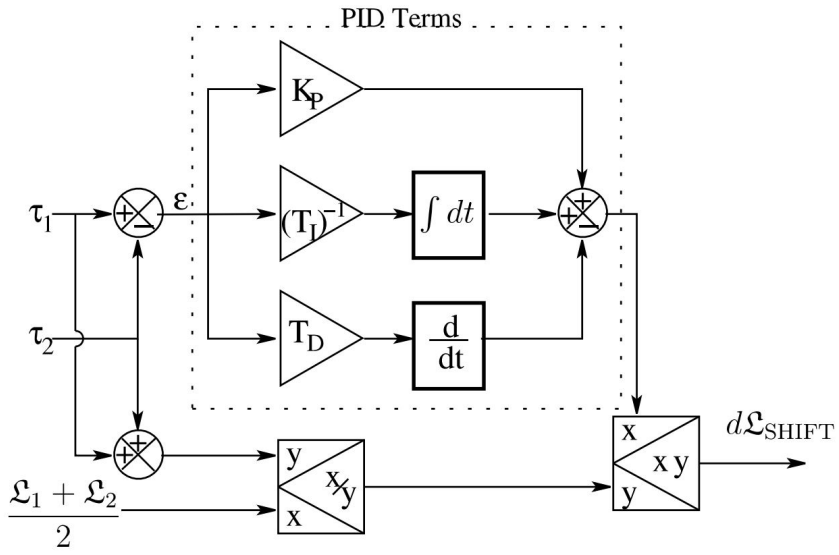


Figure 3.5: PID algorithm adaptation diagram, Source: [100]

In article [100], the authors explored adapting the PID algorithm for load balancing in particle simulations. One of the experiment's assumptions was the continuous exchange of small simulation areas to minimize system load. Figure 3.5 illustrates the algorithm's scheme. The input data includes computation time on individual processors and the location of the areas' center of gravity. The output is a new center of gravity.

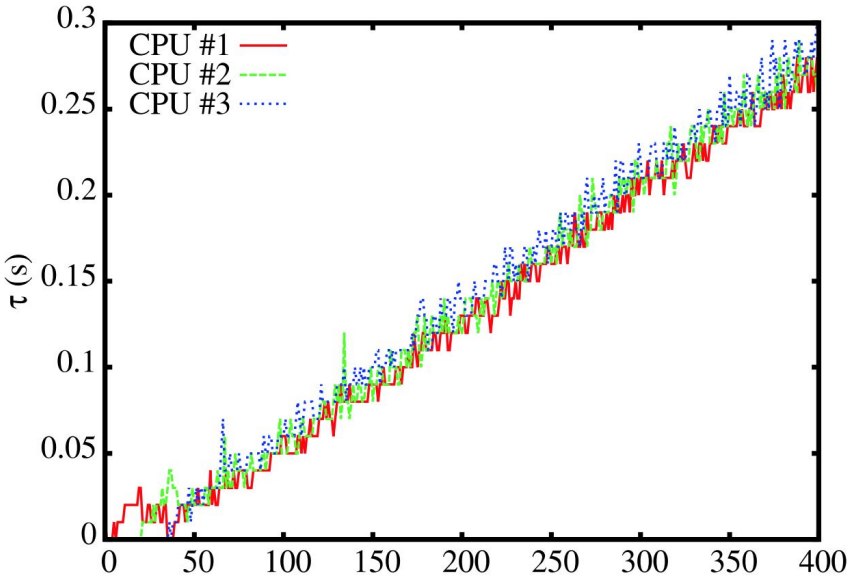


Figure 3.6: Computational time expended by each processor, Source: [100]

The load balancing results shown in article [100] are impressive (Figure 3.6). The figure ilustrates computational time τ consumed by each CPU is indeed uniform across the active processors in relation to real time. However, the article lacks a comparison with traditional load balancing algorithms. Notably absent are details on acceleration and the number of cell exchanges. It is possible that using different data as the basis for the PID algorithm could yield even better results.

In conclusion, while the PID algorithm can be applied to load balancing, the attempts made so far indicate that there is still room for further research in this area.

4 Scalability of existing Urban Traffic Simulators and their distribution

Traffic simulation tools must evolve to handle increasingly complex scenarios as simulated urban areas expand and transportation systems become more complex and detailed. Scalability, defined as the ability of a simulation system to accommodate growing numbers of vehicles and larger geographical areas without loss of performance or accuracy, is a crucial factor in this evolution. Another critical aspect is dispersion, which refers to how problem of simulation is partitioned among available computing resources. This chapter explores the capabilities and limitations of existing urban traffic simulation tools in addressing these challenges, emphasizing their approaches to scalability and dispersion modeling.

4.1. Historical Perspective on Parallel Traffic Simulation

Researchers have been exploring urban traffic simulation methods for nearly seven decades. In 1955, the "kinematic wave" [81] approach was introduced to model vehicle flow on congested roads, inspired by floodwater dynamics in rivers. Early studies primarily focused on high-density traffic movement rather than individual driver behavior, categorizing the model as macroscopic. While effective for large-scale traffic analysis, this approach lacked the granularity needed for simulating individual vehicle interactions, which microscopic models later addressed.

A significant breakthrough in microscopic traffic simulation occurred in 1990 when researchers successfully ran a simulation on a CRAY supercomputer [85]. Utilizing the SIMD (Single Instruction Multiple Data) architecture, this study focused on the feasibility of high-performance computing for traffic modeling rather than optimizing efficiency. It employed the NETSIM simulator [34], originally developed in 1978, which used a time-step model to represent road networks as directed graphs. The supercomputer enabled simulations on networks ranging from small-scale (48 nodes) to city-sized (800 nodes), vastly outperforming conventional local machines, which handled only around 100 nodes. Although the computational speedup wasn't revolutionary, the study demonstrated that large-scale microscopic simulations on supercomputers were viable.

In 1994, researchers further investigated [93] the efficiency of microscopic urban traffic simulations on supercomputers using the Nagel-Schreckenberg cellular automata model [94]. Their study explored two parallelization strategies: a vectorized, single-bit approach optimized for specific supercomputer architectures and a non-vectorized method that distributed road segments across processors. Simulations on three supercomputers, each with up to 1,024 nodes, achieved speeds roughly 1,000 times faster than prior models. A 10,000 km-long roadway simulation ran 100 times faster than real-world traffic conditions, though performance gains diminished as the problem size increased. The largest experiment, simulating a 10-million-km roadway, exceeded real-time processing limits. Rigorous testing demonstrated that the Nagel-Schreckenberg model was highly amenable to parallelization, achieving performance improvements up to 1000 times faster than alternative solutions available at the time.

This groundbreaking research focused heavily on optimizing algorithms for specific processor architectures. The most remarkable results were achieved on the single-node Nec-SX3 vector computer, showcasing the potential of optimized parallel execution.

A key performance boost came from removing centralized synchronization point and introducing asynchronous time steps. This technique allowed vehicles to advance independently if no nearby obstructions were present, reducing computational bottlenecks. However, the study noted limitations, including system customization for specific supercomputers, reduced flexibility, and a simplified road network that limited real-world applicability. Despite these constraints, the desynchronization method improved parallel efficiency, offering potential for even greater computational speedups in future simulations.

In 2001, researchers attempted to develop a parallel, distributed microscopic traffic simulator using TRANSIM [92], an agent-based model operating in discrete time and space. This tool simulated real-world road networks, including infrastructure like intersections and traffic signals, with models for lane-changing and car-following to determine vehicle speed. It also supported public transport simulation. The primary goal was to assist in road infrastructure planning, which relies on large-scale simulations predicting traffic trends over decades. Since optimal designs required multiple runs—sometimes up to 50—enhancing computational speed through parallel processing was a priority. The researchers focused on optimizing the simulation module while keeping other components, like route generation, unchanged.

To boost efficiency, the simulated area was divided into sections of equal size, each assigned to a different processor. These sections were processed independently, with data exchange occurring at boundary regions. The METIS algorithm [65] was used for partitioning, ensuring balanced computational loads. An alternative method, orthogonal recursive bisection, was tested but resulted in excessive fragmentation of major roads, increasing data exchange overhead. The study used Portland's road network, consisting of over 20,000 roads. According to the authors, a full-day traffic simulation was completed in just 36 minutes, achieving a speed 40 times faster than real-time. While impressive, such performance metrics should be interpreted with caution, as they strongly depend on model configuration—particularly the length of the simulation timestep, which directly affects computational load and simulation fidelity. Tests were conducted on 16 dual-core processors. Additionally, researchers estimated simulation time based on system architecture, accounting for processor interconnect bandwidth and network topology. The ASCII Blue Mountain Parallel Supercomputer demonstrated near-optimal speedup with up to 256 processors. The best speedup was achieved with a Gbit Ethernet-based architecture.

From a parallel computing perspective, METIS played a crucial role in ensuring balanced computational workloads and minimizing inter-process communication. The partitioning process divided roads at midpoints rather than intersections, streamlining vehicle transitions and data exchange. However, the system lacked a desynchronization mechanism, which could have further improved performance. Moreover, only a limited number of hardware-based tests were conducted, making it difficult to fully evaluate real-world effectiveness. Another framework, ParamGrid [70], introduced a distributed simulation approach across a LAN-connected cluster of personal computers. Designed for PARAMICS [20], a microscopic traffic simulator using continuous modeling, ParamGrid managed simulation partitioning, vehicle transitions, and synchronization between sections. The network was structured as a graph with intersections as nodes and roads as directed edges. However, its uniform partitioning strategy, which disregarded street density, led to imbalanced traffic distribution.

A key feature of ParamGrid was its desynchronization mechanism, allowing sections to run independently for multiple steps before synchronizing. While this improved efficiency, it also introduced errors, limiting its reliability for precise traffic modeling. The framework was tested on a California road network with over 6,000 roads, 3,000 intersections, and 14,000 vehicles, using four single-core PCs. It achieved a speedup of nearly 3×, though its efficiency remained constrained by uneven computational distribution and the lack of a robust desynchronization method.

In 2015, a novel asynchronous data synchronization method was introduced [149] to reduce simulation errors. This approach was integrated into the agent-based, microscopic, parallel, and spatially discrete SEMSim simulator, which relies on event-driven simulation dynamics. The simulation map was partitioned using METIS, with divisions placed at road segment centers, following a method similar to [92]. In this event-driven system, agents in a region generated actions stored in a timestamped queue. Each process managed a designated map segment and maintained a single event queue for its agents, handling both driver (e.g., speed adjustments, lane changes) and vehicle behaviors (e.g., battery charge updates). The algorithm allowed processes to function independently, only synchronizing when necessary with adjacent regions.

Previously, synchronization relied on global barriers, requiring all processes to pause for communication and computation. To improve efficiency, the authors introduced an asynchronous method called Mutual Appointment (MA). This approach allowed neighboring processes to schedule synchronization events—meetings—where data was exchanged, and future sync times were determined. By limiting communication to direct neighbors, this method enhanced scalability and reduced dependencies. However, in high-traffic scenarios, frequent meetings could undermine efficiency. To address this, the authors proposed Relaxed Mutual Appointment (RMA), which extended synchronization intervals based on the premise that traffic flow simulations inherently approximate real-world conditions. While this reduced computational overhead, it introduced a risk of inaccuracies due to skipped updates, making results non-deterministic. SEMSim was tested on Singapore's road network, consisting of over 43,000 intersections and 84,000 streets, with real-world vehicle route data. Simulations peaked at 75,000 vehicles and ran on a four-node cluster (each node with two eight-core processors) using up to 32 processes.

The study compared MA, RMA, and traditional global barrier synchronization. RMA reduced synchronization frequency by nearly 50% compared to MA, which itself required 1.5 times fewer synchronizations than the global barrier method. Both MA and RMA significantly reduced message exchanges compared to the global barrier approach, which required all processes to communicate. Performance was analyzed using Amdahl's Law (strong scaling) across 4, 8, 16, and 32 processes (see Fig. 4.1).

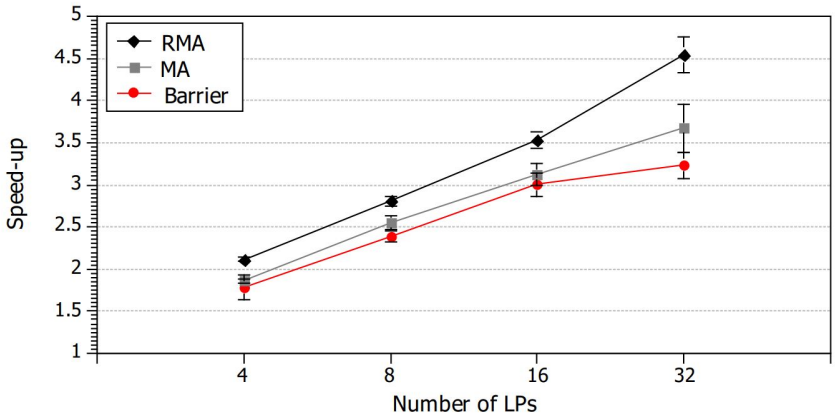


Figure 4.1: Acceleration of the simulation depending on the number of worker processes used and the synchronization strategy Source: [149]

At 32 processes, RMA achieved a speedup of 4.5, MA reached 3.6, and the global barrier method 3.25. Though the gains weren't dramatic, asynchronous synchronization showed promise for large-scale simulations by eliminating global barriers and minimizing unnecessary communication. However, the trade-off—skipping synchronization steps—remains a concern, as it can introduce inaccuracies that may not be acceptable for all applications.

The SMARTS (Scalable Microscopic Adaptive Road Traffic Simulator) [111] is likely one of the earliest distributed urban traffic simulators utilizing a continuous space model, further described in section 4.3.3. The simulator was designed for scalability, parallel processing, and microscopic traffic modeling, it integrates lane-changing

model [67] and vehicle-following dynamics model [133] while supporting OpenStreetMap data for realistic road network generation. SMARTS operates with a central server and worker processes, where the server manages task distribution, visualization, and result collection, while workers handle vehicle movement and traffic light updates. Synchronization is decentralized—each worker communicates only with adjacent processes per simulation step, ensuring efficiency and scalability.

A key limitation of the SMARTS at the time was that the system supported a maximum of 30 worker processes, which greatly constrains scalability. If one worker processes data slower than others, it delays the entire system. Since computation time depends on vehicle load, an initial balance is achieved by evenly distributing vehicles and minimizing inter-worker communication. However, static load balancing does not adjust to dynamic traffic shifts, such as congestion at major events, highlighting the need for dynamic load redistribution. SMARTS was tested in a heterogeneous environment using Melbourne's road network (150,000 nodes, 275,000 directed edges). The simulation supported up to 1 million vehicles, but performance declined at higher loads. With 100,000 vehicles and 30 workers, the simulation ran nearly five times faster than real-time (factor 4.97). However, at full capacity (1 million vehicles and 30 workers), the factor dropped to 1.14, showing that vehicle density significantly impacts computational efficiency.

The SMARTS simulator was developed as a scalable and distributed system but was not originally designed to operate in a massively parallel environment like High-Performance Computing (HPC) systems. Section 4.5 explores improvements and research made as part of work within this thesis to modify the simulator for compatibility with such machines.

4.2. Parallelizing Traffic Simulations

This section explores critical challenges in traffic simulators, focusing on synchronization mechanisms, the balance between simplified and realistic modeling, and issues related to distribution transparency. Understanding and addressing these challenges is vital for developing scalable, accurate, and efficient parallel and distributed traffic simulation.

4.2.1. Centralized vs. Asynchronous Synchronization

Synchronization mechanisms play a critical role in parallel traffic simulations, influencing both computational performance and accuracy. Early parallel simulation approaches [92, 70] frequently relied on centralized synchronization methods, wherein a master node or centralized mechanism coordinates the timing and state updates across all computational nodes. Although centralized synchronization ensures strong consistency and straightforward implementation of global simulation states, it inherently introduces scalability bottlenecks. As the size and complexity of simulations grow, the central node becomes the whole system bottleneck, being overwhelmed by communication overhead and processing load, in consequence significantly impacting overall simulation performance.

Attempts to alleviate these scalability issues led researchers to explore extending synchronization intervals. Nagel and Rickert [92] demonstrated that increasing the intervals between synchronization points could notably improve computational efficiency. However, this approach introduces trade-offs: extending intervals reduces synchronization frequency, resulting in potentially outdated or inconsistent information propagating through the simulation. Consequently, this diminished synchronization granularity leads to a reduction in simulation accuracy, particularly problematic in detailed micro-simulation contexts requiring high temporal precision.

To overcome the inherent limitations of centralized approaches, recent research has shifted towards asynchronous synchronization strategies. These decentralized techniques, exemplified by the work[149], focus on localized interactions, allowing computational nodes to synchronize primarily with their immediate neighbors or

relevant subsets of the simulation environment. By restricting synchronization to localized regions rather than globally coordinated interactions, asynchronous strategies significantly reduce communication overhead and enhance scalability, making them particularly suitable for large-scale parallel simulations.

Nevertheless, asynchronous methods introduce their own complexities and challenges. Achieving accurate and consistent state synchronization across distributed simulation nodes without a central coordinating entity can become intricate. Issues such as event causality management, deadlock avoidance, and maintaining simulation stability require sophisticated strategies, often involving complex algorithmic solutions and additional overhead for managing local synchronization conditions.

Thus, selecting between centralized and decentralized asynchronous synchronization involves trade-offs related to scalability, computational performance, implementation complexity, and simulation accuracy. While centralized synchronization may simplify some coordination mechanisms, it fundamentally limits scalability and becomes a bottleneck in large-scale or super-scalable systems. As emphasized by Engelmann and Geist [28], achieving super-scalability requires the complete avoidance of global synchronization points. Recognizing these trade-offs is therefore essential to design parallel traffic simulations capable of scaling to thousands of processing elements.

4.2.2. Simplified Models vs. Realistic Scenarios

Traffic simulations vary widely in their methodological complexity, typically falling into two broad categories: simplified models and realistic, detailed simulators. Simplified models, such as those implemented in MATSim and informed by the Nagel-Schreckenberg framework, prioritize computational efficiency and performance, enabling simulations of extensive transport networks over prolonged periods. These models generally adopt discrete time steps, cellular automata approaches, or simplified vehicle dynamics and traffic interaction description to achieve rapid computational execution. Consequently, they are particularly suitable for studying large-scale transportation scenarios and policy impacts, yet their abstraction often limits their ability to accurately replicate complex interactions and behaviors observed in real-world traffic or the impact of microscale changes and their influences on overall large environment behaviour.

Inversely, detailed simulators, exemplified by platforms such as SUMO and SMARTS, strive for greater realism by incorporating microscopic details such as individual vehicle trajectories, precise driver behavior models, and interaction dynamics at intersections and bottlenecks. While these realistic simulators can closely match empirical observations and are invaluable for detailed scenario analyses, their computational complexity significantly restricts their applicability to large-scale or city-wide scenarios due to resource-intensive calculations of detailed environment and problems with synchronization overheads.

The growing complexity of real-world mobility patterns increasingly reveals the limitations of simplified traffic models, which often fail to capture critical phenomena such as congestion propagation, local disruptions, or multimodal interactions. As a result, behaviorally rich models are becoming essential for producing meaningful simulation outcomes. However, these models are computationally demanding, which makes scalability not just a desirable feature, but a fundamental requirement. The approaches and solutions considered in this thesis address this challenge by focusing on scalable simulation methods capable of supporting detailed, realistic traffic models without compromising computational feasibility.

4.2.3. Distribution Transparency

The key factor is to ensure that the division of the simulation environment across multiple computational nodes is fully transparent from the simulation logic's perspective. In other words, the fact that the simulation is distributed across a network of machines should not influence the correctness or consistency of the simulation outcomes. This

property, referred to as *distribution transparency*, is critical to ensuring that the simulator produces identical results whether run on a single node or a highly parallel cluster of machines. It is also a critical factor for scalable traffic simulations. Effective distribution transparency ensures that distributed nodes operate coherently, replicating the behavior of a single cohesive system, which is crucial for realistic and scalable simulations.

The SMARTS simulation framework currently exhibits limitations due to insufficient distribution transparency, as identified during the course of this research. The environment model must be distributed across multiple computing nodes to achieve efficient and scalable simulation. To facilitate this, the road graph is partitioned into smaller sub-regions. These sub-regions serve as fundamental units ensuring transparency in the distribution and effective load balancing. Each computing node is responsible for managing its assigned area, along with essential information from adjacent areas and neighboring sub-regions, crucial for maintaining the accuracy and consistency of the simulation.

This distribution strategy is inspired by the stencil computation pattern but significantly differs in how the data is stored and managed. Instead of maintaining global storage for simulation data and state, this information is decentralized and distributed across individual computing units. This decentralized approach improves scalability concerning data handling and allows flexibility beyond a single-node configuration. Partitioning the simulation environment into specific segments allows nodes to concentrate only on their assigned areas, thereby accelerating computation, reducing memory overhead, and enabling the simulation of extensive, heavily congested areas.

Specifically, SMARTS divides the simulation environment into discrete segments by shadowing a virtual grid on top of simulated environment, assigning some chosen segments of the environment to separate computing processes. While such a simple division algortihm helps distribute computational load and theoretically allows simulations to scale, it is not an error-free solution. Critically, the current approach limits information exchange strictly to immediate neighboring nodes, which should be able to obtain all required knowledge to determine correct state of the simulated fragment for the next simulation step as it is a significant constraint. Ensuring accuracy and correctness in distributing the environment model is essential. It is imperative that this distribution remains entirely transparent to the simulation process itself, meaning it should not influence computation results in any way. Unfortunately, existing implementations, such as those referenced in [111], sometimes fail to address this adequately. These shortcomings may lead to scenarios where vital simulation information—like vehicle positions or states—is lost during inter-node communication, adversely affecting the correctness and reliability of the simulation outcomes.

Due to the intrinsic nature of the traffic simulation model employed by SMARTS, accurate simulation of vehicle behavior frequently necessitates knowledge from indirect neighbors—nodes beyond immediate proximity. Vehicles in complex traffic environments regularly rely on context that spans multiple segments, such as awareness of traffic conditions ahead, beyond their immediate vicinity, to make realistic and effective driving decisions. When computational nodes cannot obtain essential data from these indirect neighbors, the resulting lack of contextual awareness leads to incomplete or outdated local knowledge and wrong decisions.

Figure 4.2 illustrates a typical communication issue arising from improper information exchange between indirectly connected nodes in a distributed simulation setup. In this scenario, nodes A, B, C, and D simulate distinct parts of the environment. Although nodes A and D are not direct neighbors, the simplified boundaries separating these nodes, indicated by dashed lines, can cause problems. Each node is tasked with processing the environmental fragment it manages, exchanging information primarily with nodes responsible for neighboring fragments. However, determining optimal fragment boundaries is challenging, as boundaries must be minimized to reduce communication overhead while still ensuring all necessary information exchange.

Even when the boundaries are appropriately defined, certain arrangements of sub-regions may still necessitate data access from indirect neighbors, creating errors. As depicted in Figure 4.2, an incorrectly designed division can lead to problems. Specifically, car $C1$, simulated within fragment A, needs information about car $C2$, simulated

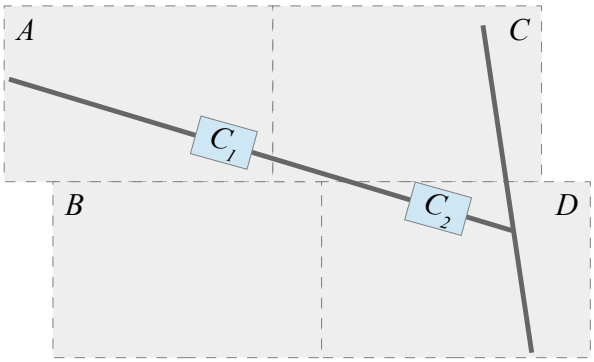


Figure 4.2: The issue of communication requirements between non-adjacent fragments.

within fragment D. However, since fragments A and D are not direct neighbors, and the connecting boundary passes through fragment C, node A does not receive essential information about car $C2$. Consequently, the simulation performed by node A lacks crucial information, causing inaccuracies and deviations in simulation outcomes.

This problem is particularly substantially visible during scenarios involving rapid traffic fluctuations, complex intersection maneuvers, or densely populated segments of road networks. In such scenarios, nodes relying only on direct neighbors become prone to generating unrealistic and unreliable behaviors due to insufficient information. Consequently, simulation inaccuracies become increasingly pronounced, manifesting as incorrect vehicle trajectories, unnatural traffic flow patterns, or unrealistic congestion formations. These inaccuracies not only compromise the realism of the simulation but also limit the reliability and applicability of the simulation outcomes for real-world decision-making and predictive analyses.

To comprehensively address these shortcomings, architectural improvements aimed at enhancing distribution transparency are essential. Such improvements might include revising the environmental segmentation strategy to ensure that relevant information can flow more freely and accurately across nodes, thus improving coherence between segments. Additionally, introducing multi-hop communication protocols could enable nodes to acquire necessary data from nodes further away, ensuring a more comprehensive situational awareness across the distributed simulation. Moreover, more sophisticated data-sharing and aggregation mechanisms, such as predictive information caching or asynchronous message passing, could reduce communication latency and improve overall system responsiveness.

Furthermore, adaptive segmentation techniques, capable of dynamically adjusting segment boundaries based on real-time traffic density or simulation demands, could further enhance transparency and efficiency. By optimizing the distribution strategy dynamically, simulation performance could significantly improve, reducing the likelihood of bottlenecks or information isolation.

Ultimately, improving distribution transparency through these combined strategies would significantly enhance the accuracy, scalability, and reliability of distributed traffic simulations within simulation frameworks like SMARTS. Such enhancements would contribute substantially toward achieving simulations that more faithfully represent real-world traffic dynamics, supporting better traffic management decisions.

4.3. Overview of contemporary Urban Traffic Simulators

Urban traffic simulators have evolved significantly over the decades, with various tools offering unique approaches to scalability and dispersion modeling. Below is a detailed examination of several major simulators and their contributions to the field.

4.3.1. MATSim (Multi-Agent Transport Simulation)

MATSim [56] is a versatile multi-agent simulation framework specifically designed to model transportation systems. It employs a highly simplified approach by representing road lanes as queues and treating vehicles as individual agents navigating through these queues. This abstraction significantly reduces the computational overhead compared to more detailed traffic simulation models, enabling MATSim to achieve exceptional computational efficiency. As a result, it has become particularly well-suited for conducting large-scale transportation simulations that can model entire metropolitan areas or national transportation systems. Notably, MATSim has an active user community and even hosts dedicated user conferences, underscoring its maturity and real-world applicability.

Over the years, MATSim's development efforts have predominantly focused on optimizing its internal architecture for single-node computations. Key techniques such as the use of parallel queues and concurrency have been implemented to maximize performance within the confines of a single machine [26]. These innovations have allowed MATSim to make effective use of modern multi-core processors. However, this reliance on vertical scaling imposes a fundamental limitation on its scalability: the framework cannot leverage computational resources beyond the maximum number of CPUs available on a single node. This bottleneck becomes particularly apparent in scenarios that demand extensive computational resources, such as simulations of densely populated urban areas with complex transportation networks or peak-hour traffic conditions.

MATSim's architecture also presents other inherent challenges. Notably, the lack of distributed computation capabilities significantly constrains its ability to handle scenarios requiring immense computational capacity. As road networks grow in size and complexity, or as the volume of traffic agents increases, the system's performance begins to degrade. Additionally, the incremental addition of numerous features and extensions over MATSim's extended development history has resulted in increased computational complexity. This, in turn, can lead to slower performance, especially when simulating highly intricate scenarios that require a more detailed representation of traffic dynamics.

While MATSim has established itself as a highly efficient tool for large-scale simulations, its emphasis on computational efficiency often comes at the expense of accuracy. The framework's simplified representation of traffic dynamics makes it less suitable for simulations requiring detailed and precise modeling of individual vehicle behaviors, such as lane-changing dynamics, intersection interactions, or the influence of driver heterogeneity. For scenarios where detailed accuracy is necessary, other simulation models may be more appropriate, despite their higher computational demands.

4.3.2. SUMO (Simulation of Urban Mobility)

SUMO (Simulation of Urban Mobility) [76] is an open-source, highly flexible, and feature-rich traffic simulation tool designed to model urban mobility systems at both macroscopic and microscopic levels. SUMO operates on continuous space and state models, allowing for detailed and realistic simulations of individual vehicle movements, lane-changing behaviors, and traffic signal interactions. It is equipped with advanced decision-making and motion modeling capabilities, making it well-suited for a wide range of applications, including traffic management, urban planning, and policy evaluation. SUMO's modular architecture enables seamless integration with real-world data, such as traffic sensor readings, GPS traces, and OpenStreetMap data, providing users with the ability to simulate diverse traffic conditions and scenarios effectively.

One of SUMO's key strengths is its comprehensive support for modeling various aspects of urban transportation, such as multi-modal traffic systems, pedestrian dynamics, public transportation, and the impact of emerging mobility solutions like connected and autonomous vehicles (CAVs). These capabilities have made SUMO an invaluable tool for researchers and policymakers seeking to evaluate the effectiveness of traffic management strategies, such as adaptive traffic signal control, congestion pricing, and vehicle routing policies. Furthermore, SUMO's

built-in visualization tools and detailed output options allow for in-depth analysis and presentation of simulation results.

Despite its extensive features, SUMO faces significant challenges when it comes to scaling simulations for large and complex networks. The tool's reliance on sequential processing for many of its core computations limits its ability to fully utilize modern distributed computing resources. Although SUMO supports parallelized simulations through techniques like multi-threading and distributed computation, these efforts have often resulted in mixed outcomes. Studies such as [2, 139, 15] have explored various approaches to enhance SUMO's scalability, including distributed computing frameworks and advanced synchronization strategies. However, achieving efficient parallel execution has proven difficult, primarily due to the complexity of maintaining accurate state synchronization across computational nodes.

The primary bottleneck in SUMO's scalability lies in its reliance on continuous space models, which require precise, real-time updates of vehicle positions and interactions. This leads to a high frequency of data exchanges between nodes during distributed simulations, often causing increased communication overhead and synchronization errors. Additionally, as network sizes grow and the number of simulated vehicles increases, the computational load per node rises, exacerbating these challenges. While innovative methods, such as partitioning road networks into smaller, independent sub-networks or leveraging approximations in synchronization, have shown promise, they often come at the cost of reduced accuracy or increased algorithmic complexity.

Given these challenges, the need for more robust and efficient parallelization techniques in SUMO remains a critical area of ongoing research. Advances in high-performance computing, such as GPU acceleration and cloud-based distributed systems, may hold potential for overcoming these limitations. Future developments in SUMO's architecture could focus on hybrid parallelization strategies that combine fine-grained threading with distributed frameworks, ensuring scalability without sacrificing accuracy or computational efficiency.

4.3.3. SMARTS (Scalable Microscopic Adaptive Road Traffic Simulator)

The first distributed urban traffic simulator known to use a continuous spatial model was the SMARTS simulator, described in detail in [111], titled the Scalable Microscopic Adaptive Road Traffic Simulator. Among its capabilities are support for distributed calculations with or without a central unit, the import of maps from the OSM format, and cross-platform portability. Additionally, the simulation operates on a continuous model where each road has a specific length $l \in \mathbb{R}$, and vehicles have fixed positions along this length within the range $(0, l)$. The SMARTS supports distributed computations across multiple nodes. These features position it as one of the most advanced simulators for large-scale traffic scenarios.

The map division algorithm is a straightforward implementation of a geometric approach. The map is initially divided into a grid of rectangles, which are then merged into areas to minimize the number of neighboring areas. The neighborhood is later determined based on the roads connecting these areas.

Furthermore, the authors assume that synchronizing the state only needs to include the roads connecting neighboring areas. Specifically, a road is considered a boundary road if it starts in area A and ends in area B, and such a road is simulated within area B. During the state synchronization process, area A must send information about vehicles entering the boundary road (vehicle transfer between areas), and area B should send information about the last vehicle on the road ending. This information is required by the vehicle decision algorithm used in the SMARTS simulation, known as IDM, described in the previous section 2.4.2.

However, SMARTS is not without its limitations. A critical issue lies in its lack of distribution transparency. While the first part of the state synchronization is well-justified, there is a concern that for very short boundary roads, the information about the vehicle visible from the preceding road may not be transmitted, leading to potential simulation errors, further discussed in section 4.2.3. The authors also discuss the need to search the map within

a given radius as a simulation parameter, demonstrating experimentally that increasing this distance increases computational costs and slows down the entire simulation. However, this aspect is not addressed during the modeling of synchronization communication.

4.3.4. CORSIM (CORridor SIMulation)

CORSIM [47] is a microscopic traffic simulation model developed by the Federal Highway Administration. It combines two simulation programs, NETSIM for surface streets and FRESIM for freeway networks, to offer integrated simulation capabilities. CORSIM's strength lies in its ability to model detailed driver behaviors, such as lane-changing and gap acceptance.

In terms of scalability, CORSIM is primarily designed for small to medium-sized networks. The lack of inherent parallelization limits its ability to handle large-scale scenarios effectively. While the integration of surface street and freeway modeling offers a comprehensive framework, its computational requirements increase significantly as network size grows, making it less suitable for extensive urban networks with high traffic volumes.

4.3.5. AIMSUN

AIMSUN (Advanced Interactive Microscopic Simulator for Urban and non-urban Networks) [33] is a versatile traffic simulation tool capable of performing microscopic, mesoscopic, and macroscopic simulations. Its modular architecture enables users to switch between simulation levels depending on the requirements of the analysis. AIMSUN supports real-time traffic management and policy evaluation, making it a popular choice for urban traffic studies.

From a scalability perspective, AIMSUN offers limited parallelization capabilities for microscopic simulations. While mesoscopic and macroscopic simulations can handle larger networks, the accuracy of traffic dispersion modeling diminishes as the level of detail decreases. Efforts to parallelize AIMSUN's microscopic engine have shown promise, but synchronization challenges remain a barrier to achieving seamless scalability for large-scale simulations.

4.3.6. PTV VISSIM

PTV VISSIM [140] is a widely used, but in comparison to previous commercial microscopic traffic simulation tool known for its high level of customization and integration with other transportation modeling software. It provides a detailed representation of vehicle and pedestrian interactions, making it suitable for analyzing complex urban environments.

VISSIM's scalability is constrained by its centralized computation framework. While it can simulate moderately large networks, its performance deteriorates significantly as the number of vehicles and intersections increases. Parallelization efforts have been limited, and the absence of distributed computation capabilities restricts its ability to handle very large-scale scenarios effectively.

4.3.7. Influences of Nagel-Schreckenberg Model

The Nagel-Schreckenberg model, while not a simulator itself, has provided significant insights into traffic flow dynamics. Its cellular automata framework is used to explore vehicle behavior, traffic jams, and the effects of simple decision-making rules on traffic patterns. Though not directly applied in complex simulation environments, the model's influence is evident in the design of asynchronous synchronization methods and simplified approaches to

scalability. These contributions have informed the development of simulators by offering a conceptual foundation for parallelized and distributed traffic modeling.

The scalability of the Nagel-Schreckenberg model as branch of cellular automata models has been a subject of extensive research [94, 92, 93], particularly in the context of parallel computing and high-performance simulations. Early implementations demonstrated that the model was well-suited for parallel execution, with performance improvements reaching up to 1000 times faster than traditional methods. However, when applied to large-scale traffic networks, challenges such as synchronization overhead, uneven workload distribution, and communication delays between computational nodes became apparent. Studies have shown that global synchronization is not necessary for accurate traffic simulation and that splitting tasks into equal parts and desynchronizing updates can significantly enhance performance. These insights have been critical in shaping modern parallel simulation frameworks.

A key breakthrough in improving the scalability of traffic simulation has been the desynchronized parallel simulation approach. This method allows individual computational spaces to update independently, with controlled delays in information exchange between neighboring spaces. In a large-scale implementation using Erlang [135], this approach demonstrated linear scalability up to 19,200 cores, simulating 11.5 million vehicles across 240,000 road segments while maintaining 160 simulation steps per second. The success of this implementation highlights the importance of asynchronous computation, minimal centralization, and efficient load balancing in achieving large-scale simulation efficiency. By adopting a distributed model where computing nodes interact locally rather than globally, researchers have successfully mitigated the bottlenecks associated with rigid synchronization, paving the way for highly scalable urban traffic simulations.

4.4. Comparative Analysis of Simulators

Simulator	Scalability	Synch. Method	Key Features	Limitations
MATSim	Single-node	Centralized	High efficiency, queue-based lanes	Limited scalability, complex architecture
SUMO	Limited	Centralized	Feature-rich, flexible modeling	Synchronization errors at large scale
SMARTS	Limited	Asynchronous	Distributed, real-world modeling	Distribution transparency issues
CORSIM	Single-node	Centralized	Detailed driver behavior modeling	Limited scalability
AIMSUN	Limited	Centralized	Versatile simulation levels	Scalability diminishes at microscopic
VISSIM	Limited	Centralized	Detailed vehicle-pedestrian interaction	Poor performance in large networks
Erlang NS	Very good	Asynchronous	Foundational insights, scalability focus	Simplified, unrealistic road grid

Table 4.1: Comparison of Traffic Simulation Tools

4.5. Improvements made in SMARTS

At the initial stage of the work described in this thesis, I explored the possibility of using the SMARTS simulator as a foundation for large-scale traffic simulation. SMARTS showed promise in terms of its architecture and agent-based approach, so significant effort [97, 158] was made to improve its scalability, computational efficiency and adaptability to demanding simulation scenarios. These improvements included architectural modifications, optimization of communication protocols, improved model distribution and streamlined data management strategies. While these enhancements extended SMARTS' capabilities to some extent, they also revealed fundamental limitations, which ultimately motivated the development of a dedicated solution. The following section outlines the attempted improvements.

As initial part of work described in this thesis SMARTS has undergone substantial enhancements to improve its scalability, computational efficiency, and adaptability to large-scale environments. These improvements among others include architectural advancements, optimization of communication protocols and streamlined data management strategies. Each of these advancements has played a crucial role in extending SMARTS' capabilities, making it more adept at handling real-world traffic simulation challenges. Below are explored these enhancements:

4.5.1. Parallel Processing and Distributed Architecture

Initial versions of SMARTS [111] relied on a centralized client-server model, where a single server process managed multiple worker processes that handled sections of the simulation. This design, while functional for smaller simulations, quickly became a bottleneck as traffic models scaled up. The heavy reliance on a central server meant that all computational processes needed to synchronize with it, leading to increased communication latency and system-wide slowdowns.

The introduction of a distributed architecture marked a transformative step in SMARTS' evolution. By allowing worker processes to operate independently on different portions of the simulation, this distributed system significantly improved parallel processing capabilities. Each worker was assigned a segment of the traffic network and processed it concurrently with others, reducing dependency on a single central entity. This enhancement enabled SMARTS to scale beyond its initial limitations, making it feasible to execute complex urban simulations involving millions of vehicles across expansive road networks.

A major issue in early iterations of SMARTS was global synchronization, where each worker had to wait for all others to complete their computations before proceeding to the next time step. This synchronous execution model severely hampered scalability, as slower workers dictated the overall simulation speed.

To address this inefficiency, SMARTS adopted asynchronous synchronization mechanisms, significantly reducing idle time among worker processes. This approach allowed each worker to operate independently, synchronizing only when necessary with adjacent workers to ensure a seamless simulation flow. By limiting synchronization to smaller localized clusters rather than across the entire simulation, this strategy markedly enhanced performance, especially for large-scale scenarios where computational loads varied across different regions.

4.5.2. Priority Synchronous Parallel (PSP) Model

Another significant enhancement to SMARTS' scalability was the Priority Synchronous Parallel (PSP) model introduced in [68]. Traditional simulation methods applied uniform processing priorities to all vehicles within the simulation. However, this approach proved inefficient when dealing with cross-boundary interactions, as vehicles transitioning between different worker domains experienced delays due to inter-process communication constraints.

The PSP model introduced a two-phase computation strategy, prioritizing vehicles near worker boundaries. These vehicles were processed first to ensure smooth transitions between partitions, significantly reducing inter-worker synchronization overhead. By frontloading critical computations and deferring less time-sensitive operations, the PSP model dramatically improved overall efficiency, ensuring a more fluid simulation process.

4.5.3. Scaling SMARTS for HPC System

The SMARTS traffic simulation system has been extended [97, 158] to support large-scale urban traffic modeling on high-performance computing (HPC) infrastructure. These extensions were developed to overcome the inherent scalability challenges of simulating continuous urban traffic, particularly for extensive road networks with realistic driver behaviors. THe part of my research, as an international collaboration between the University of Melbourne and AGH University of Science and Technology introduced several optimizations to ensure efficient parallel execution across thousands of computing cores.

Parallelized Initialization for Improved Efficiency

One of the key improvements involved eliminating inefficient sequential initialization. Previously, the server had to communicate with each worker process individually, causing a bottleneck in large-scale simulations. To address this, the initialization process was redesigned to allow parallelized data distribution across multiple worker processes. This ensured that metadata about neighboring workers, necessary for inter-process communication, was efficiently distributed without requiring a single-point synchronization at the server. Additionally, the dependency on a direct server-client data transfer model for setup was removed in favor of a shared filesystem-based configuration approach, reducing setup overhead and significantly improving initialization speed.

Thread Pooling for Efficient Message Processing

In previous implementations, incoming messages were handled using a thread-per-message model, where each new message spawned a separate processing thread. While sufficient for smaller-scale simulations, this method led to inefficiencies at larger scales, consuming excessive system resources and reducing overall performance. To address this, a thread-pooling approach was introduced, where reusable worker threads process incoming messages stored in a queue. This method significantly reduced processing overhead, minimized system resource exhaustion, and allowed the simulator to efficiently scale to a large number of workers.

Decentralized Data Collection and Serialization

A major challenge in large-scale simulations was the centralization of data collection, which resulted in significant memory bottlenecks at the server. In the extended version of SMARTS, a decentralized result collection mechanism was implemented, allowing each worker to store results independently. This approach removed the need for a master node to handle data aggregation, enabling fully parallelized data serialization. As a result, data processing and storage became more efficient, making it feasible to simulate millions of vehicles across large urban environments.

Enhancements in Data Scalability

SMARTS simulations involve processing both static and dynamic data. Static data, such as road network structures, remains constant throughout the simulation, while dynamic data, such as vehicle movements, continuously changes. As the size of the simulation increases, memory constraints become a critical factor. To improve data scalability, the static road network model was partitioned among workers, reducing memory overhead per worker. Additionally, dynamic data, such as vehicle trajectories, was managed more efficiently by introducing mechanisms to estimate and control peak memory usage, ensuring that simulations remained stable even at extreme scales.

Improved Visualization and Simulation Result Handling

One of the challenges in large-scale simulations is handling and visualizing the vast amount of data generated. The initial SMARTS implementation required the server to collect all vehicle trajectories, leading to significant memory consumption and communication overhead. The extended version introduced local aggregation of simulation metrics, allowing workers to process and store only relevant summary data, such as vehicle density and traffic flow, before sending it to the server. This greatly reduced message sizes and improved performance. For real-time visualization, only selected regions of the simulation were tracked, further minimizing overhead while maintaining detailed insights into traffic patterns.

Experimental Validation and Scalability Testing

The scalability of the extended SMARTS system was tested on the Prometheus[1] supercomputer, which consists of over 53,000 computing cores and was provided by plgrid[2] infrastructure. Two types of scalability tests were conducted: Strong Scalability Test – Keeping the total number of vehicles constant while increasing the number of workers. Results showed that performance improved linearly up to approximately 200 workers (200 cores), after which initialization overhead became a limiting factor. Weak Scalability Test – Increasing both the number of workers and the number of simulated vehicles proportionally. This test confirmed that the simulation algorithm itself was highly scalable, with execution time remaining stable even as the total number of workers increased beyond 2000 cores.

4.6. Conclusion

The development of urban traffic simulation tools has seen remarkable progress in scalability and computational efficiency. However, significant challenges remain, particularly in modeling realistic traffic dispersion across large networks. Addressing these challenges requires a careful balance of scalability, accuracy, and architectural innovation. Insights from existing simulators provide a valuable foundation for the research aimed at overcoming these limitations and advancing the state of traffic simulation technology. The advancements in SMARTS have significantly enhanced its performance, but its architectural constraints necessitate the development of a completely new HiPUTS (High Performance Urban Traffic Simulation) designed from the very begining with distributions and super-scalability as main assumptions described in this work.

[1] https://www.cyfronet.pl/en/18378,artykul,prometheus_supercomputer.html
[2] http://www.plgrid.pl/

5 | HiPUTS (The High Performance Urban Traffic Simulator)

This chapter presented the description of the method along with algorithms used in design and implementation of The High Performance Urban Traffic Simulator (**HiPUTS**) [95], a scalable urban traffic simulator based on a continuous microscopic model. The architecture was built around spatial decomposition into hexagonal patches, decentralized execution, and a local communication model, enabling distribution transparency and scale invariance. Key components such as patch management, inter-node coordination, decentralized load balancing, and the simulation algorithm were described in detail, demonstrating how the system can support large-scale, high-fidelity simulations across distributed computing environments.

HiPUTS represents a new generation of simulation platforms tailored specifically for large-scale, microscopic, and continuous urban traffic modeling in distributed high-performance computing environments. The HiPUTS Simulator is a super-scalable solution for distributed microscopic urban traffic simulation. It was conceptualized and developed with an ambitious goal: to overcome the limitations observed in legacy and even recent traffic simulation tools, which, although often powerful in their own right, fall short in terms of scalability, distribution transparency, and computational efficiency when applied to city-scale simulations or higher.

5.1. Design Goals and Primary Objectives

The development of the High Performance Urban Traffic Simulator (HiPUTS) was guided by a set of ambitious yet well-defined research goals grounded in the central hypothesis of this thesis: that it is possible to construct a super-scalable, continuous, microscopic urban traffic simulation framework capable of scaling to thousands of computational cores without sacrificing simulation fidelity. While existing platforms offer partial solutions, none simultaneously satisfy the combined criteria of spatial continuity, microscopic behavioral realism, distribution transparency, and computational scale invariance, as discussed in Chapter 4. The design of HiPUTS was therefore mainly the answer to the above questions as the scientific solution answering them and combining these characteristics into a unified, high-performance simulation platform along with filling the technological gap in this area.

The architecture design and implementation of HiPUTS were driven by four core design objectives:

1. **Support for Continuous Traffic Models and Future Extensibility**
   A central assumption is that vehicular traffic — both in movement and infrastructure — is best represented using continuous rather than discrete models. Roads, lanes, and vehicles all possess real-world properties such as length, width, speed, and acceleration that vary continuously rather than in fixed, quantized units. Embracing a continuous spatial model allows for more realistic and fluid representation of dynamic phenomena such as overtaking maneuvers, lane changes, vehicle interactions, and congestion propagation. Moreover, the architecture is explicitly designed to be modular, making it possible to incorporate new behavioral

or physical models in the future, such as hybrid microscopic–macroscopic models, adaptive cruise control algorithms, or advanced driver assistance systems (ADAS).

2. **Distribution Transparency and Consistency of Results**
   A common challenge in distributed simulation is ensuring that the results remain consistent regardless of the underlying deployment. The key factor is to exhibit *distribution transparency*, meaning that simulations produce equivalent outcomes whether executed on a single machine or across multiple computational nodes. This property is vital for both reproducibility and validation of simulation experiments. It is achieved through deterministic communication patterns, consistent agent behaviors, and synchronized state updates across the distributed environment. Consequently, researchers and practitioners can design experiments without needing to recalibrate models for different computing setups, which streamlines workflow and enhances confidence in the results.

3. **Super-Scalability through Scale-Invariant Computation and Load Balancing**
   Another hallmark is its *super-scalable* architecture. Traditional simulators often experience diminishing returns when scaled across multiple nodes due to uneven task distribution or communication overhead. Another important goal is to maintain *scale invariance* — the computational load assigned to each unit remains approximately constant, even as the simulation scenario expands. This is enabled by a fine-tuned *load balancing engine*, which dynamically partitions the simulation environment and reallocates agents to ensure that no computational node becomes a bottleneck. As a result capable of efficiently handling simulations involving tens or hundreds of thousands of vehicles across expansive urban networks, making it suitable for both real-time systems and offline analysis of smart city infrastructure.

5.2. Microscopic and Continuous Simulation Model

HiPUTS distinguishes itself not only by its engineering but also by its adherence to a specific **conceptual modeling philosophy** — namely, a *microscopic, continuous, and agent-based approach* to traffic simulation.

In this paradigm, *each vehicle is modeled as an autonomous agent* with its own decision-making logic. These agents are capable of adjusting their driving behavior based on the state of their visible environment, meaning local conditions, personal objectives, and interactions with nearby vehicles or traffic control elements. This allows the simulation to capture emergent phenomena such as stop-and-go waves, bottlenecks, and driver heterogeneity — patterns that are often lost in coarser macroscopic or mesoscopic models.

5.2.1. Environment

The road network in HiPUTS is represented as a directed graph, where intersections serve as nodes and roads are modeled as unidirectional edges. Although the current implementation focuses on single-lane configurations, there is also support for multi-lane roads with more complex traffic dynamics. This foundational structure allows for a clear and efficient abstraction of urban infrastructure.

Each road segment and intersection is grouped into spatially localized computational units called *patches*, with each patch containing a subset of the network—typically a collection of adjacent lanes and junctions. This partitioning is a critical component of HiPUTS's distributed architecture: each patch is managed by a single computational node, which reduces inter-node communication overhead and supports dynamic load balancing during simulation execution. This enables load balancing between compute nodes and reduces communication complexity within small processing regions (it is unlikely that a three-segment route will be spread across three different, transitively neighboring fragments). A single Patch contains sets of nodes, lanes (resulting from the simplification

that all roads are processed as single-lane), and require only information about other neighboring Patches. At a high level of abstraction, a Patch can be considered a single node connected to others via multiple lanes that act as bridges between them. The Patches are further described in following Section 5.4.

HiPUTS supports both custom-built networks and realistic traffic scenarios based on geographic data imported from *OpenStreetMap*. During import, the raw data is cleaned and transformed into a graph representation optimized for simulation.

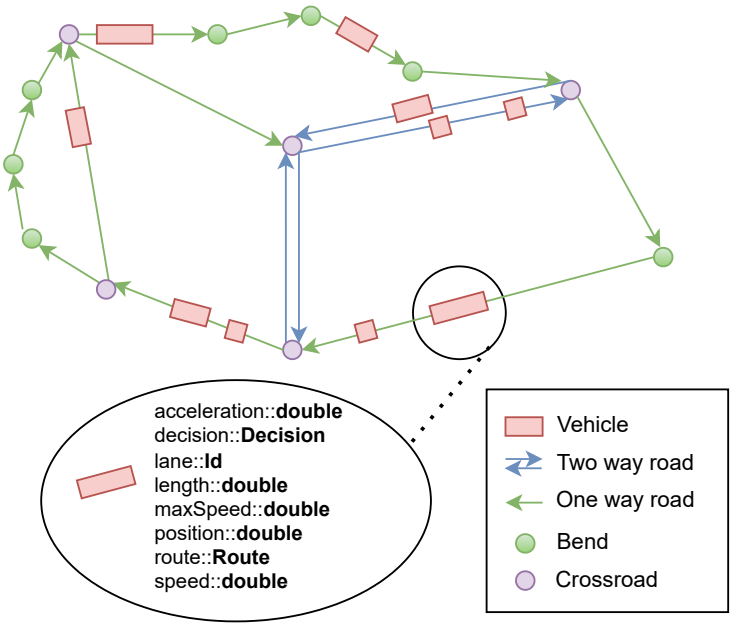


Figure 5.1: Example representation of vehicle model and graph of environment

The traffic network is modeled as a directed graph, where edges denote traffic lanes and nodes correspond to intersections or junctions, as illustrated by an example in Figure 5.1. Connections between nodes may represent either one-way (green) or two-way (blue) roads, depending on the directionality of the traffic flow. Each edge corresponds to a straight lane with a fixed length and direction, while each node represents a traffic junction—categorized as either a simple bend (connecting one or two lanes) or a more complex crossroads (connecting three or more). Each lane in the simulator includes information about its length, starting and ending nodes, and the vehicles currently on it. Additionally, it is associated with an adjacent lane going in the opposite direction.

Junctions maintain information about the ordered sequence of incoming and outgoing lanes, which facilitates routing decisions and traffic rule enforcement. In the simulator's model the lane acts as the smallest aggregation unit that is able to store vehicles, but this information about exact location on the lane is stored within the agent as the vehicle. A node in the simulator represents an intersection of several lanes or a continuation of a single lane in a different direction. It contains data on incoming and outgoing lanes, as well as its actual geographic location. The model is built based only on roads and the nodes that belong to them. Adjacent objects, such as buildings, are omitted for the sake of simplification.

5.2.2. Vehicle state

Each vehicle in the HiPUTS simulator is modeled as an autonomous agent equipped with a comprehensive set of parameters that define its physical characteristics, current state, and decision-making capabilities. These

parameters are essential for simulating realistic traffic dynamics and agent-based interactions. Some of the primary attributes associated with a vehicle are shown in the Figure 5.1 and further described below:

- **length** (`double`) – Defines the physical length of the vehicle, which affects spatial occupancy on the road and inter-vehicle spacing.
- **maxSpeed** (`double`) – Represents the maximum allowable speed the vehicle can attain under ideal conditions.
- **lane** (`Id`) – Refers to the identifier of the lane on which the vehicle is currently traveling.
- **position** (`double`) – Indicates the exact position of the vehicle along its assigned lane, expressed as a continuous scalar value.
- **route** (`Route`) – A predefined sequence of lanes and junctions that constitute the vehicle's full path through the network.
- **speed** (`double`) – Captures the current speed of the vehicle at a specific simulation timestep.
- **acceleration** (`double`) – Describes the rate of change in the vehicle's speed, which may be influenced by traffic conditions, model rules, or driver behavior.
- **decision** (`Decision`) – Encapsulates the set of behavioral choices the vehicle makes at each timestep, including actions such as adjusting acceleration, preparing for a lane change, or yielding at an intersection. This is a consequence of applying the decision models, described in the following section 5.2.3.

These attributes collectively enable each vehicle to behave in a context-sensitive manner, adapting to its environment, maintaining safe distances, and following the rules encoded in the simulation logic.

5.2.3. Decision Model

Vehicles in the simulation are autonomous agents, each maintaining a detailed state that includes its current lane, position, velocity, acceleration, physical dimensions, and a predefined route. Lanes track which vehicles occupy them, enabling local observation and vehicle-to-vehicle dynamics. Traffic behavior is governed by a modular decision framework called *deciders*. Vehicles are initialized with randomized starting positions and individualized travel goals, simulating diverse and realistic traffic patterns. Their movement is governed by the car-following models such as Intelligent Driver Model (IDM), described in section 2.4.2, to determine longitudinal behavior such as acceleration and safe following distances. At intersections, a separate decision module enforces right-of-way rules. Additional behavioral models, including MOBIL for lane changing, described in section 2.4.3, traffic light control, and overtaking on narrow roads, enrich the simulation with realistic and varied driving patterns. This enables smooth behavior and prevents collisions by encouraging safe spacing.

Beyond car-following (e.g., IDM) and lane-changing (e.g., MOBIL), vehicles interact with intersection control logic. Signalized intersections with fixed-cycle signals and and priority-based intersections where right-of-way is encoded as per-node priorities are taken into account. An approaching vehicle queries the control state (signal phase or priority relation) and constrains its longitudinal decision accordingly (e.g., enforced stopping or reduced target speed if no acceptable gap is available). This logic is applied during the *decision phase* (see Sec. 5.2.5), ensuring that all vehicles compute decisions against the same frozen world state.

This hybrid capability—combining geographic realism with modular behavioral logic and distributed computation—enables HiPUTS to simulate both theoretical scenarios and real-world urban environments at high levels of detail and computational efficiency.

The simulation advances in discrete time steps. Users can configure both the step duration and the total simulation period. For example, running 86400 time steps at one-second intervals yields one whole simulated day of traffic. This discrete temporal model enables efficient computation without compromising spatial continuity or behavioral realism.

HiPUTS is capable of modeling a wide variety of traffic entities and infrastructure components, including:

– Multi-lane roads with variable widths and speed profiles,

– Vehicles of different categories (e.g., passenger cars, buses, trucks),

– Intersections with or without traffic signals,

– Visualization and data logging modules for analyzing simulation outputs.

While these features enhance the richness of the simulation, it is the core combination of *microscopic autonomy, continuous space, and distributed scalability* that defines the simulator's contribution to the field.

5.2.4. Formal definition and characteristics of simulation model

For the purpose of building a solution for importing data from open sources, a list of requirements is defined for a valid simulation model. It specifies the properties and conditions that the model must fulfill after being loaded into the program. All of this has been described based on specific definitions.

Formal key components definition

The model ultimately used in the software is a quintuple (5 element tuple):

$$M = (N, R, L, C, I) \tag{5.1}$$

where N – the set of all nodes, R – the set of all roads, L – the set of all lanes, C – the set of all traffic signal control centers, and I – the set of all traffic signal lights. In its structure, the model resembles an extended graph, where the set of nodes can be interpreted as the set of vertices, and the set of roads as the set of edges. Additionally, the model contains information about lanes and traffic signalization.

The fundamental concepts on which this model definition is based, and which precisely describe the contents of the above sets, are:

– **road**, represented as a quadruple (4 element tuple):

$$r = (n_{\text{in}}, L_r, i, p, n_{\text{out}}) \in R \tag{5.2}$$

where n_{in} – input node, L_r – list of lanes belonging to the road, ordered according to their actual layout on the map, from left to right, in the direction consistent with traffic flow, i – traffic signal located on this road just before the output node the output node, indicating the possibility of exiting this road onto an intersection, p – information about the road's priority at the destination node, and n_{out} - the output node.

– **node**, represented as a triple:

$$n = (R_{\text{in}}, \lambda, \phi, c, R_{\text{out}}) \in N \tag{5.3}$$

where R_{in} – set of incoming roads, λ – geographical length, ϕ – geographical width, c – traffic signal control center at the node, and R_{out} – set of outgoing roads,

– **lane**, which in the model is only an object aggregating other lanes that are successors of a given lane, represented as pair:

$$l = (S, V) \in L \tag{5.4}$$

where $S = \{l_s \in L\}$ – the set containing successor lanes that may be occupied after the current lane is passed, and V – the list of vehicles currently present on the lane,

– **traffic signal control center**, which is an abstract object (i.e., one that does not exist in the real world but exists in the model), located at a node and responsible for aggregating green light groups. Its representation is a single element tuple:

$$c = (G) \in C \tag{5.5}$$

where:

$$G = \{g \subset I\} \tag{5.6}$$

is a set aggregating the mentioned green light groups. A group is a set $g \subset I$ that contains only those traffic signals $i \in I$ that can simultaneously display a green light at a given intersection. A traffic signal itself does not contain any additional information apart from the currently displayed color.

Characteristics and requirements of the model

A model defined in this way, after importing data from external sources, must satisfy the following requirements:

1. The sets N and R must form a directed graph, so for all $r_1, r_2 \in R$ such that $r_1 = (n_1, L_1, i_1, p_1, n_2)$ and $r_2 = (n_2, L_2, i_2, p_2, n_1)$, the following conditions must be met:
   - $r_1 \neq r_2$,
   - $L_1 \neq L_2$,
   - $(i_1 \neq i_2) \vee (i_1 = i_2 = \emptyset)$.
2. The sets N and R contain only those nodes and roads required for simulation in relation to the map from which the model was built. In other words, if each adopted criterion is expressed as a function $f_n : N \to \text{bool}$ and $f_r : R \to \text{bool}$, then $\forall n \in N$ satisfies $f_n(n) = \text{true}$ and $\forall r \in R$ satisfies $f_r(r) = \text{true}$. The most important criteria are:

- A node has at least one incoming or outgoing road (so it cannot be isolated), i.e. $\forall n = (R_{\text{in}}, \lambda, \phi, c, R_{\text{out}}) \in N$ we have $(R_{\text{in}} \neq \emptyset) \vee (R_{\text{out}} \neq \emptyset)$,
- A road has both a start node and an end node, i.e. $\forall r = (n_{\text{in}}, L_r, i, p, n_{\text{out}}) \in R$ satisfies $n_{\text{in}} \neq \emptyset \wedge n_{\text{out}} \neq \emptyset$,
- The connections between nodes correspond to real connections on the original map, which are actually traversed by vehicles, and any other connections visible on the map, such as boundary lines of building walls, are ignored. So if there exists a function $f_{\text{det}} : R \rightarrow$ bool that defines whether a given road is driveable, then $\forall r = (n_{\text{in}}, L_r, i, p, n_{\text{out}}) \in R$ satisfies $f_{\text{det}}(r) =$ true,
- A road is of a specific type and is not a pedestrian path, bicycle path, or private driveway. So if there exists a function $f_t : R \rightarrow$ bool that defines whether a road is a regular vehicle road, then $\forall r = (n_{\text{in}}, L_r, i, p, n_{\text{out}}) \in R$ satisfies $f_t(r) =$ true.

3. The model is informationally complete, i.e. $(N \neq \emptyset) \wedge (R \neq \emptyset) \wedge (L \neq \emptyset)$, whereas $(C = \emptyset) \wedge (I = \emptyset)$ only occur if the original map did not contain any signalized intersections. Additionally:
   - $\forall n \in N$ such that $n = (R_1, \lambda, \phi, c, R_2)$, the condition $(R_1 \subseteq R) \wedge (R_2 \subseteq R) \wedge (R_1 \cap R_2 = \emptyset)$ must hold,
   - $\forall r \in R$ such that $r = (n_{\text{in}}, L_r, i, p, n_{\text{out}})$ the condition $(n_{\text{in}} \in N) \wedge (n_{\text{out}} \in N) \wedge (n_{\text{in}} \neq n_{\text{out}}) \wedge (L_r \subseteq L) \wedge (L_r \neq \emptyset)$ must hold,
   - $\forall l \in L$ such that $l = (S, V)$, the condition $(S \subseteq L) \vee (S = \emptyset)$ must hold,
   - $\forall c \in C : \exists n \in N$ such that $n = (R_1, \lambda, \phi, c, R_2)$,
   - $\forall i \in I : \exists r \in R$ such that $r = (n_{\text{in}}, L_r, i, n_{\text{out}})$.
4. A node $n \in N$ is considered an intersection when there are more than two real-world roads involved at that node. A road qualifies as part of an intersection if it satisfies any of the following conditions:
   - An incoming one-way road by definition, i.e., such that $r_{\text{in}} = (n_1, L_1, i_1, p_1, n) \in R_{\text{in}}$, and there does **not** exist $r_{\text{out}} = (n, L_2, i_2, p_2, n_2) \in R_{\text{out}}$ such that $n_1 = n_2$, and analogously for an outgoing road.
   - A road that is both incoming and outgoing (originally bidirectional), i.e., $\exists r_{\text{in}} = (n_1, L_1, i_1, p_1, n) \in R_{\text{in}}$ and $\exists r_{\text{out}} = (n, L_2, i_2, p_2, n_2) \in R_{\text{out}}$, such that $n_1 = n_2$.
5. The model must not contain duplicates, i.e.:
   - $\forall n_1 \in N : \nexists n_2 \in N$ such that $n_1 = n_2$,
   - $\forall r_1 \in R : \nexists r_2 \in R$ such that $r_1 = r_2$,
   - $\forall l_1 \in L : \nexists l_2 \in L$ such that $l_1 = l_2$,
   - $\forall c_1 \in C : \nexists c_2 \in C$ such that $c_1 = c_2$,
   - $\forall i_1 \in I : \nexists i_2 \in I$ such that $i_1 = i_2$.
6. The graph built from roads and nodes must be connected. If there exists a function $\mathit{path} : N \times N \rightarrow N^*$ returning a path from one node to another, then for all $n_1, n_2 \in N : (n_1 \neq n_2)$, the inequality $\mathit{path}(n_1, n_2) \neq \emptyset$ must hold.
7. The graph must be free of artificial dead-end roads resulting from map cutting during export. Therefore, $\forall n = (R_{\text{in}}, \lambda, \phi, c, R_{\text{out}}) \in N$ the inequalities $|R_{\text{in}}| \geq 1$ and $|R_{\text{out}}| \geq 1$ must hold.

8. The model also considers relationships between roads found in open data sources. Examples of such relations may include no-entry or entry requirements, which in the model should be represented as an appropriate lack of successors for lanes on specified roads. For example, for a no-entry relation, we define a function *noentry* : $R \times R \rightarrow$ *bool* with the property that $\forall r_1 = (n_{1in}, L_1, i_1, p_1, n_{out1}), r_2 = (n_{2in}, L_2, i_2, p_2, n_{out2}) \in R$ such that $r_1 \neq r_2$ and *noentry*$(r_1, r_2) =$ *true*, then the following holds: $\forall l_1 \in L_1 : \nexists l_2 \in L_2$ such that, if $l_1 = (S_1, V_1), l_2 = (S_2, V_2)$, then $l_2 \in S_1$.

9. Intersections are not overly complex, meaning that, having a distance function between nodes *dist* : $N \times N \rightarrow \mathbb{R}^+$ and a defined minimum distance $d \in \mathbb{R}^+$, one can assert that $\forall n \in N : \nexists n_x \in N$ such that $n \neq n_x$ and yet *dist*$(n, n_x) < d$.

10. Each road $r = (n_{in}, L_r, i, p, n_{out}) \in R$ has a defined set of lanes $L_r \neq \emptyset$ that belong to it. Moreover, each lane $l = (S, V) \in L_r$ has a defined non-empty successor set $S \neq \emptyset$, i.e., lanes that vehicles may occupy after completing travel on the current lane. Between successor sets and lane sets on the road, the following conditions hold:

    - $\forall r = (n_{in}, L_r, i, p, n_c) \in R : \nexists l_1 = (S_1, V_1), l_2 = (S_2, V_2) \in L_r$ such that if $l_1 \neq l_2$, then there exists a successor from one lane that belongs to another lane in the same road, i.e., $(S_1 \cap L_r = \emptyset) \wedge (S_2 \cap L_r = \emptyset)$.
    - $\forall r = (n_{in}, L_r, i, p, n_c) \in R$ and final node n_c associated with intersection $(R_{in}, \lambda, \phi, c, R_{out})$ having all outgoing roads $r_x = (n_c, L_x, i_x, p_x, n_{outx}) \in R_{out}$, the union of all successor lanes is created: $L_{out} = \bigcup L_x$. It is possible that for lanes $l_{r1} = (S_1, V_1), l_{r2} = (S_2, V_2) \in L_r$ there exist successors such that $S_1 \cap S_2 \neq \emptyset$, which means that a vehicle can exit the road onto different successor roads using the same target lane. If $\bigcap S_l = \emptyset$, then each outgoing road at the intersection is reachable from only one incoming road.

    The set of successor lanes may also include lanes from the incoming road if the road has more than one lane toward the same direction. That is, for every road $r = (n_{1in}, L_{r1}, i_1, p_1, n_{out1})$, $r_2 = (n_{2in}, L_{r2}, i_2, p_2, n_{out2}) \in R$ if indeed:

    The set of successors may be empty when the preceding road has a greater number of lanes compared to the subsequent one, i.e., $\forall r_1 = (n_{\text{in1}}, L_{r1}, i_1, p_1, n_{\text{out1}}),\ r_2 = (n_{\text{in2}}, L_{r2}, i_2, p_2, n_{\text{out2}}) \in R$ is true, if there exists a lane $l_1 = (S_1, V_1) \in L_{r1}$ such that $S_1 = \emptyset$, then it holds that $|L_{r1}| \geq |L_{r2}|$.

11. Each node $n = (R_{in}, \lambda, \phi, c, R_{out}) \in N$ should aggregate incoming roads R_{in} and outgoing roads R_{out} in an ordered manner, preferably according to some predefined physical metric. Thus, these sets should be represented as ordered lists:

$$R_{in} = [r_{in1}, r_{in2}, \ldots, r_{in|R_{in}|}], \quad R_{out} = [r_{out1}, r_{out2}, \ldots, r_{out|R_{out}|}]$$

    where $\forall i, j \in \{1 \ldots |R_{in}|\}$ if $i \leq j$ then $r_i \leq r_j$, and analogously for R_{out}. An example of such a metric might be the angle relative to a defined road (e.g., the first in the list).

12. Information about traffic signals at an intersection should be included in the model fragment only if signalization actually exists at that location. In other words, the presence of a traffic signal control center $c \in C$ at a node $n = (R_{in}, \lambda, \phi, c, R_{out})$ is optional, and the center may simply be unassigned: $c = \emptyset$. A similar case applies to roads with nonexistent traffic lights; for a road $r = (n_{in}, L, i, p, n_{out}) \in R$, signalization is optional and may be unassigned by setting $i = \emptyset$, i.e., the exit from the road to the nearest intersection is not governed by any traffic light.

13. If a node $n = (R_{in}, \lambda, \phi, c, R_{out}) \in N$ has a non-empty signal control center $c \neq \emptyset$, then all incoming roads $r = (n_{in}, L, i, p, n_{out}) \in R$ to that node must have a signal assigned, i.e., $i \neq \emptyset$. Moreover, in every existing signal control center $c = (G) \neq \emptyset$, the set of green light groups $G = \{g \subseteq I\}$ must also be non-empty. All such groups must be mutually exclusive and no signal may belong to more than one group, i.e., $\forall g_1, g_2 \in G : (g_1 \neq g_2) \Rightarrow (g_1 \cap g_2 = \emptyset)$.

14. Each road $r = (n_{in}, L, i, p, n_x) \in R$ must have an explicitly defined priority $p \in \{0, 1\}$ with respect to other roads entering the same node $n_x = (R_{in}, \lambda, \phi, c, R_{out}) \in N$ being an intersection. If the target node is not an intersection, then no road has priority and all share the same lowest level. Therefore:

    - If $n_x = (R_{in}, \lambda, \phi, c, R_{out}) \in N$ is an intersection, then $\exists r_x = (n_{in_x}, L_x, i_x, p_x, n_x) \in R_{in}$ such that $p_x = 1$, and for all other roads, $p = 0$, i.e., $\forall r = (n_{in}, L, i, p, n_x) \in R$ such that $r \neq r_x$, it holds that $p = 0$.
    - If $n_x = (R_{\text{in}}, \lambda, \phi, c, R_{\text{out}}) \in N$ is not an intersection, then $\forall r = (n_{\text{in}}, L, i, p, n_x) \in R_{\text{in}}$ it holds that $p = 0$.

The requirements defined above ensure that the imported simulation model preserves both structural and semantic correctness. The graph representation must remain connected, complete, and free of duplicates or artificial artifacts. Roads and nodes are constrained to reflect only driveable and relevant network elements, while intersections are consistently characterized with priorities, signalization rules, and lane-successor relations. These constraints collectively guarantee that the model remains internally coherent, topologically valid, and directly usable in microscopic traffic simulation. In particular, they prevent inconsistencies that could arise from incomplete map exports, misclassified road types, or ill-defined traffic signal logic. As a result, the formal model not only provides a faithful abstraction of real-world data but also establishes a robust foundation for scalable and distributed execution within HiPUTS.

5.2.5. Time Model and Decision-Update Algorithm

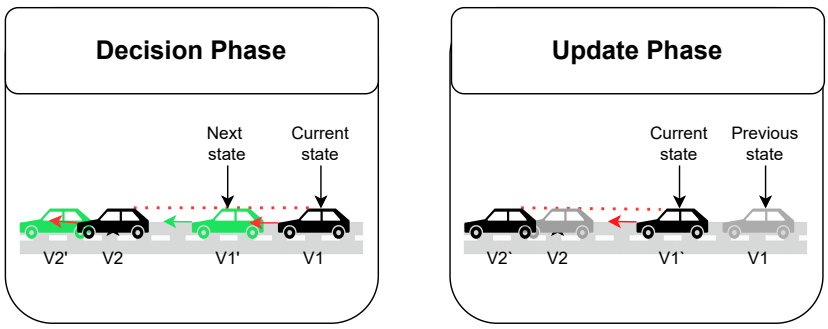


Figure 5.2: Illustration of two-stage phase execution: Decision phase (left) and Update phase (right).

The simulator advances in fixed discrete time steps Δt. Each iteration consists of two phases computed over a snapshot of the previous global state. The overview of the 2-phase Algorithm 1 within a single simulation step is presented below. Those stages are illustrated in Figure 5.2, where two vehicles are visualized on a road fragment in two states ($V1$, $V1'$ for vehicle 1 and $V2$, $V2'$ for vehicle 2). This separation of stages corresponds to two phases executed in sequence:

1. **Decision phase** (read-only): In the first phasethe vehciles invoke their behavior models (such as IDM and MOBIL) to determine next-step decisions. The output of a decision phase for vehicles includes information

about changes in desired acceleration, lane change intent, and routing choices. These decisions are stored temporarily in a local cache within each vehicle agent but are not yet applied to the agent's actual state. The decision as the future state (position, speed, acceleration, etc.) of vehicles is visible in the Figure 5.2 in green color $V1$‘ for vehicle 1 and $V2$‘ for vehicle 2. This decision state is among other determined by decider models (IDM and others) and current states of vehicles $V1$ and $V2$ within decision stage.

For every vehicle, the decision models (car-following, lane-changing, intersection control) compute intended next-state parameters, e.g., acceleration, lane-change intent, and route choice. The results are stored in per-vehicle decision buffers; the world state is not modified in this phase.

2. **Update phase** (write): After the whole decision phase has been completed, the simulator launches a second batch known as *update tasks*. These tasks revisit each road and iterate through the same vehicle collection. This time, however, the previously computed decisions are applied to each vehicle, modifying its position, velocity, lane, and possibly triggering inter-patch migration if the vehicle has crossed a boundary. In the update task the actual state of vehicle is assigned based on previously determined decision state in *decision task*. For instance in the Figure 5.2 for the vehicle 1 previously defined in decision stage next state $V1$‘ is applied as its current state after the update and $V1$ state is discarded (visible in grey color).

   All vehicles apply their previously computed decisions, updating position, velocity, lane assignment, and potentially triggering inter-patch migration. Only after all updates are applied can conflicts (e.g., collisions) be consistently detected and resolved.

This two-phase design ensures that results do not depend on the processing order of vehicles within a single step, as all decisions are made based on a consistent and unmodified world state before any updates occur. This enables parallel execution without data races and ensures determinism under identical inputs (described in Section 5.5. In distributed runs (utilizing more than one computing node), the barrier between steps incorporates synchronization for vehicles crossing patch borders (see Section 5.6.1).

Algorithm 1: One simulation step (Δt).

Input: World state at time t: $M(t)$
Output: World state at time $t{+}\Delta t$: $M(t{+}\Delta t)$

// Decision phase (read-only on $M(t)$)
foreach *road r in parallel* **do**
  foreach *vehicle $v \in r$* **do**
    compute_decision$(v, M(t)) \rightarrow D_v(t{+}\Delta t)$
  end
end
// Exchange boundary intents if needed
synchronize_border_state()
// Apply phase (mutating M)
foreach *road r in parallel* **do**
  foreach *vehicle $v \in r$* **do**
    update_state$(v, D_v(t{+}\Delta t))$
  end
end
handle_migrations_and_collisions()
return $M(t{+}\Delta t)$

5.3. HiPUTS: General Overview and Architectural Strategies

The High-Performance Urban Traffic Simulator (HiPUTS) is designed to model and simulate large-scale urban traffic systems with microscopic detail while maintaining high computational efficiency. Its foundation lies in a *microscopic, agent-based paradigm*, in which each vehicle is represented as an autonomous agent with individualized parameters (section 5.2.2), behavioral rules, and a dynamic route through the road network. This level of detail enables to support research on traffic phenomena that emerge from interactions between individual agents, such as congestion formation, lane changes, and intersection dynamics.

The simulation environment is constructed from real or synthetic road networks (section 5.2.1), abstracted as directed graphs where intersections correspond to nodes and roads to directed edges, typically representing individual traffic lanes. The network is partitioned into *patches*—spatially bounded and computationally independent subregions (section 5.4). Patches are distributed across worker processes, which can execute either on multiple cores of a single machine or across nodes in a computing cluster. Each worker is responsible for simulating vehicles and local traffic dynamics within its assigned patches, while maintaining remote representations of adjacent border regions to ensure decision-making consistency near boundaries.

Simulation advances in discrete time steps (section 5.2.5). In each step, vehicles update their state according to behavioral models (section 5.2.3), adjusting acceleration, speed, and position based on both internal objectives (e.g., destination, desired speed) and external stimuli (e.g., leading vehicles, traffic signals). Vehicles moving between patches trigger communication between neighboring workers, which exchange state information to preserve continuity of trajectories and interactions across partitions.

A decentralized execution model ensures scalability. After initialization and data loading — typically coordinated by a temporary server process - the simulation proceeds without central control. Each worker operates autonomously, synchronizing only with immediate neighbors. This decentralized design reduces communication bottlenecks and improves fault tolerance. To adapt to dynamic conditions, HiPUTS incorporates load balancing mechanisms: patches can be reassigned between workers if local traffic densities change significantly, thereby maintaining balanced computational effort without interrupting the simulation timeline.

The simulator has been implemented in Java and is available as open-source software at:

`https://github.com/hiputs/hiputs`

5.3.1. Architectural Strategies

Balancing microscopic fidelity with large-scale performance poses inherent challenges. Microscopic modeling requires detailed state tracking and decision-making for each vehicle, leading to high computational cost. Scalability, in contrast, requires that the system handle increasing environment network sizes and vehicle populations without linear growth in computation time. HiPUTS addresses this tension by adopting a distributed, parallel-first architecture.

The framework was designed to dissect the simulation cycle, isolate computational hotspots, and structure tasks to maximize independence and parallel execution. Road infrastructure and vehicle behaviors are represented in a form that allows each patch to be simulated largely in isolation, with synchronization occurring only at defined boundaries. This strategy enables both intra-node parallelism (via multithreading) within a single computing node and inter-node scalability (via distributed execution) utilizing multiple computing nodes.

The architecture of HiPUTS can be described in four complementary dimensions:

– **Patch-Based Decomposition of Road Networks** – partitioning road networks into manageable subregions (patches), enabling local computation and controlled inter-patch communication.

– **Efficient Parallel Processing** – leveraging multicore architectures at the node level through task scheduling and synchronization mechanisms that allow concurrent execution of vehicle updates and other tasks.

– **Distributed Execution Design** – describes designed solutions such as distributing patches across multiple nodes with their dynamic reallocation for workload balancing, implementing protocols for inter-node synchronization, message exchange, and state sharing based on coordination achieved through neighbor-to-neighbor synchronization rather than centralized control with consistent time progression across the distributed system.

5.4. Spatial Decomposition into Patches

To enable efficient and scalable simulation, the environmental model must be partitioned and allocated across multiple computational nodes. This is achieved by subdividing the global road network graph into smaller, spatially bounded segments. These segments form the foundation for maintaining both distribution transparency and computational load balancing. Each node in the simulation is responsible not only for managing its designated region of the map but also for maintaining awareness of immediately neighboring areas. This local neighborhood information is essential for ensuring correct behavior at region boundaries, particularly for vehicles interacting across those boundaries. The foundation for the patch distribution mechanism used in HIPUTS is the grid partitioning strategy discussed earlier in Section 3.2.5. This method builds upon a domain-specific approach originally introduced in [157].

Although conceptually similar to a stencil-based approach in parallel computing, HiPUTS adopts a distinct data management paradigm. Rather than maintaining a globally shared simulation state, the system employs a fully decentralized model in which each node independently holds and processes only the data relevant to its assigned domain. This strategy significantly enhances scalability by reducing inter-node dependencies and localizing memory and computational requirements.

As a result, the simulation environment can span arbitrarily large areas, including entire cities or regions experiencing intense traffic congestion. By isolating processing to discrete fragments of the network, the simulator improves computational efficiency and minimizes memory overhead. This partitioning is essential for enabling simulations that involve massive datasets which would not fit into the memory of a single machine. The design not only accelerates simulation execution but also supports high-resolution modeling of complex, real-world traffic systems across distributed architectures.

One of the fundamental challenges in distributed traffic simulation is providing distribution transparency (described in section 4.2.3) by ensuring that each vehicle has access to the current information about its surroundings within a predefined visibility radius. The following subsection introduces a novel graph partitioning algorithm specifically designed to address this issue. It outlines the core concept, details key improvements and innovations.

A central element of HiPUTS's distributed architecture is the decomposition of the road network into independent, spatially coherent subregions called *patches*. Each patch represents a contiguous set of road segments and junctions that are processed together on a single compute node. This spatial decomposition strategy facilitates parallel computation by ensuring that different compute nodes can independently simulate traffic activity within distinct, non-overlapping geographic areas. Since patches represent isolated subregions of the road network, each node can perform computations on its assigned patch and requires only synchronization with neighbouring patches. This allows for efficient concurrent execution across multiple processing units. The detailed description of patch and the algorithm for its creation are defined in the following section 5.4.1.

The partitioning process begins with the import of road network data, typically sourced from OpenStreetMap or a custom network description. This data is then transformed into a graph-based representation, where nodes

correspond to intersections and edges denote roads or lanes. Once the network is represented in this form, graph partitioning algorithms are applied to divide it into smaller subgraphs—each representing a patch.

5.4.1. Patch Construction Algorithm Based on Geometric Partitioning

To support efficient and scalable traffic simulation, the road network must be partitioned into computational units—referred to as *patches*—that balance performance and visibility constraints. The method proposed in this work applies a geometric decomposition based on a hexagonal tiling of the simulation space, which offers spatial coherence and minimizes inter-patch dependencies.

Hexagonal Grid and Definitions

Given a directed graph $G = (V, E)$ with a defined edge weight function w that determines the length. Let us consider a vehicle located on an arbitrary edge $(u, v) \in E$ at any position $x \in (0, w((u, v)))$. For a given positive real number $r_0 \in \mathbb{R}$, the visibility radius of the vehicle is defined as the value of the following function:

$$E \times \mathbb{R} \to (V', E'), \text{ where } V' \in 2^V \text{ and } E' \in 2^E \tag{5.7}$$

The sets V' and E' are determined according to the formulas:

$$\begin{aligned} E' &= \{(u_k, v_k) \in E \mid w((u, v)) - x + \text{shortest\_path}(v, u_k) < r_0\} \\ V' &= \{v_k \in V \mid \exists (u_l, v_l) \in E' : u_l = v_k \vee v_l = v_k\} \end{aligned} \tag{5.8}$$

Considering all possible positions a vehicle can occupy along a graph edge, the edge assignment to a specific partition depends on the maximum achievable visibility radius. As a result, the definition of the set E' can be simplified as follows:

$$E' = \{(u_k, v_k) \in E \mid \text{shortest\_path}(v, u_k) < r_0\} \tag{5.9}$$

This simplification implies that it is possible to define a visibility radius not only for specific positions but for the entire edge (u, v). In general, edges that converge at the same endpoint will share the same visibility radius, as they lead to a common vertex. Given that road network graphs typically contain far more edges than vertices, it is more efficient to define visibility in terms of vertices and then apply this measure uniformly to all incident edges. For this reason, this is the following definition:

Visibility radius at distance $r_0 \in \mathbb{R}_+$ in a graph $G = (V, E)$, is defined as the output of the function r:

$$r : V \to 2^V \times 2^E \tag{5.10}$$

Specifically:

$$\begin{aligned} r(v) &= (V', E'), \quad \text{where} \\ E' &= \{(u_k, v_k) \in E \mid \text{shortest\_path}(v, u_k) < r_0\} \\ V' &= \{v_k \in V \mid \exists (u_l, v_l) \in E' : u_l = v_k \vee v_l = v_k\} \end{aligned} \tag{5.11}$$

A **hexagonal grid** is defined as a partition of the two-dimensional simulation space into regular hexagons of side length r_0. Each point in the space lies uniquely within one hexagon (excluding shared edges), and each hexagon is indexed using a consistent coordinate system (e.g., axial or cube coordinates). The hexagon side length

r_0 is selected to correspond to the *vehicle visibility radius*, ensuring that all necessary environmental data for an agent's decisions are contained locally or within adjacent patches.

Although many alternative spatial decomposition schemes exist (e.g., square tilings [94], Voronoi diagrams [27], or graph-based clustering [66]), selecting one that aligns with the simulation's performance and communication constraints is non-trivial. A poor choice can result in high inter-node communication or unbalanced workloads or boundary synchronization errors. Therefore, this thesis introduces an original, hexagon-based decomposition strategy tailored to preserve locality, support balanced computation, and minimize inter-patch communication within distributed simulation environments, ensuring correctness.

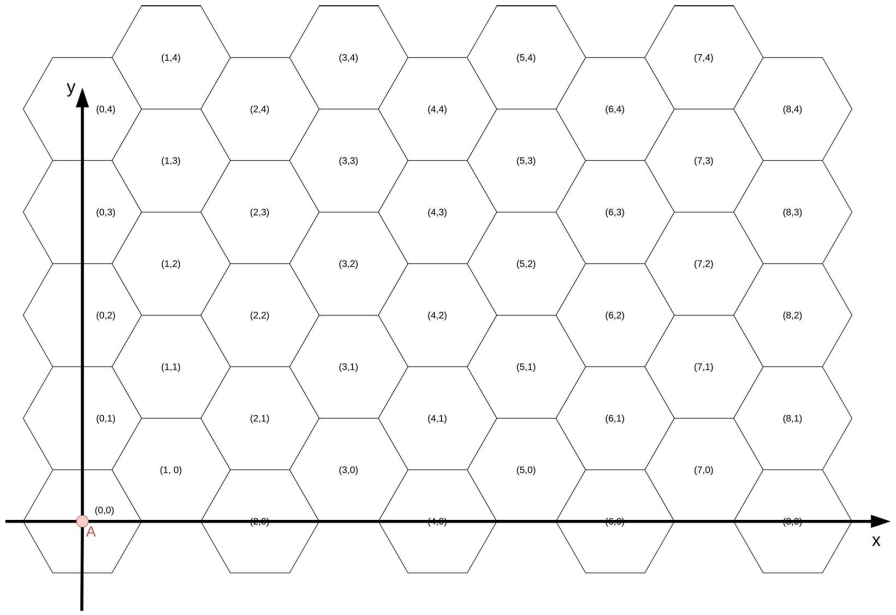


Figure 5.3: An example mesh of hexagons centered at point A along with their indexation

The grid is defined by two parameters:

- r_0 – the side length of each hexagon (visibility-based radius),
- $A = (x_0, y_0)$ – the center of the reference hexagon at index $(0, 0)$.

Each spatial coordinate (x, y) is mapped to a hexagon by determining its index analytically based on its offset from point A. Additionally, there is introduced indexing in such a grid in the manner as presented in the Figure 5.3.

Two-Phase Mapping: Vertex and Edge Assignment

The graph is projected onto the hexagonal grid through a two-stage process:

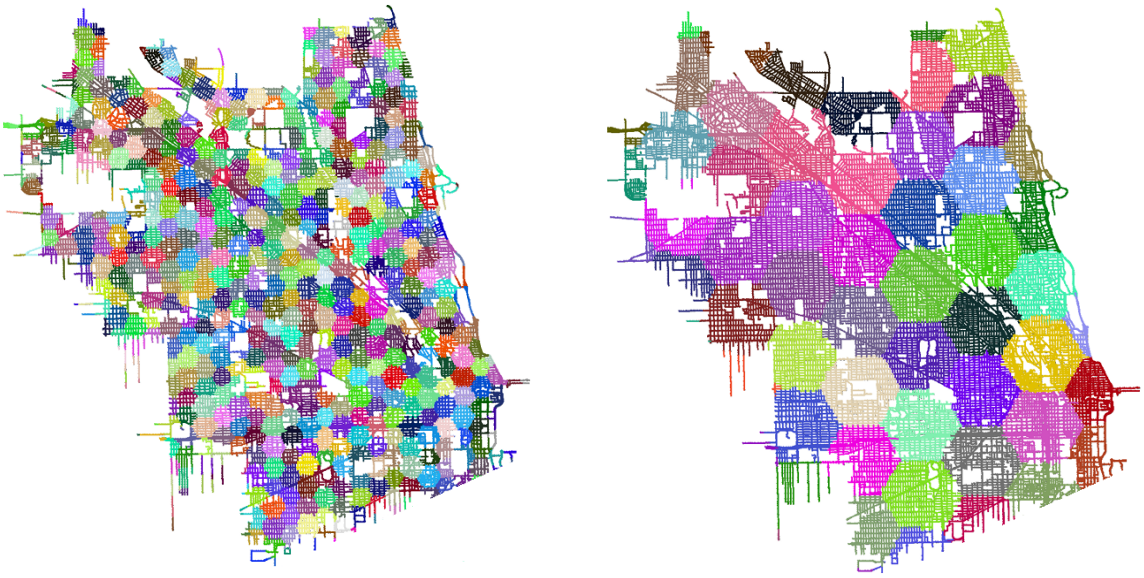
1. **Vertex assignment:** Each vertex in the road network is mapped to a hexagon patch based on its coordinates. This can be performed efficiently using geometric inequalities and is linear in the number of vertices. Special handling is applied for vertices lying on hexagon boundaries—such ambiguities are resolved randomly or through heuristic tie-breaking.
2. **Edge assignment:** For each edge $e \in E$, two cases can be identified:
   - If both endpoints belong to the same hexagon, the edge is assigned to that patch.
   - If the endpoints lie in different hexagons, the edge is treated as a *border edge* and handled using dedicated rules to preserve visibility constraints.

Handling Border Edges

To preserve correctness, border edges—edges connecting vertices in different patches—require careful treatment. Several strategies are possible:

- **Direction-based rule:** Assign the edge to the patch of its destination vertex. While simple, this can lead to violations when the edge spans non-neighboring patches.
- **Edge splitting:** Divide the edge at its intersection with the hexagon boundary, creating a new vertex at the splitting point. Each segment is then assigned to the corresponding patch. This guarantees that all road segments reside within a patch and its neighbors.
- **Hybrid strategy:** Preprocess and split long edges to reduce the maximum edge length $l_{\max}$, then increase the visibility radius r_0 by a buffer equal to $l_{\max}$. This ensures correctness while avoiding excessive patch enlargement.

Bidirectional roads, modeled by two directed edges, must be split symmetrically to preserve consistency. This requires pairing such edges and handling them together during the splitting phase.



(a) Division into smaller patches

(b) Division into bigger patches

Figure 5.4: Division of the same northern part of Chicago into patches using different handling of border edges, resulting in varying patch sizes.

The division of the northern part of Chicago into patches, as illustrated in Figure 5.4, demonstrates significant variation in the size and shape of the resulting areas. These differences arise primarily from the structural characteristics of used road network. In densely urbanized zones with high intersection density and shorter road segments, patches tend to be smaller and more uniform. In contrast, areas with sparse connectivity, fewer intersections, or long uninterrupted road segments produce larger patches.

This variation is a direct consequence of using a fixed visibility radius r_0 to define patch boundaries. In dense regions, the visibility radius quickly encompasses nearby nodes and edges, leading to compact patches. However, in areas with long roads or limited connectivity, the same radius must stretch further to collect sufficient graph elements, resulting in larger patches.

This adaptive behavior is beneficial in several ways: it aligns patch size with local network complexity, reduces unnecessary fragmentation in low-density areas, and enables more balanced computational workloads when simulating traffic or optimizing routing decisions. Additionally, the localized nature of patches supports scalability in distributed systems by limiting the scope of required data exchange between adjacent regions.

Correctness Condition

A patch is considered *correct* if the visibility radius r_0 of every vehicle on any road segment within the patch is entirely contained within that patch or its directly adjacent patches. Formally:

For all $v \in V$, the visibility region $r(v) = (V_r, E_r)$ derived from vertex v must satisfy:

$$\forall w \in V_r, \quad (f_v(v), f_v(w)) \in S(P) \quad \text{and} \quad \forall e \in E_r, \quad (f_v(v), f_e(e)) \in S(P) \tag{5.12}$$

where $S(P)$ denotes the adjacency relation among partitions in the grid, and f_v, f_e are mapping functions assigning vertices and edges to patches.

Due to the geometric properties of regular hexagons, if all patches are constrained within hexagons of radius r_0, and border edges are appropriately assigned, the visibility condition is inherently satisfied.

Optimization and Grid Parameter Search

To enhance partition quality, especially with respect to load balance and cut minimization, the origin A and strategy parameters can be optimized. The space of potential grid origins (i.e., within a single hexagon) is discretized into a triangular mesh, allowing exhaustive search over candidate configurations.

A quality function Q is defined to guide this optimization:

– **Primary criterion:** Maximize the number of small patches (enabling fine-grained parallelism).

– **Secondary criterion:** Minimize the weighted length of inter-patch edges (cut cost), where edge weights are proportional to the speed limit or expected flow and edge length.

Tie-breaking is resolved using the secondary criterion. Since the process is inherently parallelizable, multiple configurations can be evaluated concurrently using available computational resources.

5.4.2. Characteristics of Patches and benefits

Each patch is assigned the following characteristics:

– **Geographical Cohesion:** Road segments and junctions grouped into a patch are selected to be spatially adjacent or closely connected. This geographic locality reduces the complexity of inter-patch dependencies and communication overhead, enabling more efficient simulation and load balancing.

– **Visibility Sufficiency:** Each patch is extended enough to provide every vehicle with the necessary visibility into its surroundings, including those just beyond the patch boundaries. This guarantees that any vehicle interacting with edge conditions—such as following or merging near the border—can acquire all relevant state information from the neighboring patch directly. Consequently, no vehicle ever requires awareness of entities two or more patches away, simplifying the simulation's data access and synchronization model.

– **Traversal Boundaries:** The physical size and connectivity of each patch are carefully designed to prevent any vehicle from fully traversing a patch within a single simulation step. This restriction is crucial for maintaining synchronization integrity, ensuring that vehicle transitions across patches are captured and processed explicitly in subsequent steps.

– **Encapsulation:** A patch is constructed with sufficient internal state and context to simulate local traffic dynamics independently, requiring only minimal data exchange with adjacent patches. This includes the layout of roads, signals, junction logic, and resident vehicles. As a result, most computational decisions can be resolved locally, reducing the need for frequent data exchange between nodes.

The proposed geometric partitioning method ensures that:

– Vehicle agents access all necessary data within their local patch and neighbors,

– Patches are spatially compact and consistent with traffic dynamics,

– Partitioning can be computed efficiently and optimized for parallel execution,

– The resulting simulation framework scales well for large, realistic traffic networks.

Patch boundaries are chosen to preserve spatial locality (short perimeter-to-area ratio) and to bound the communication surface. This yields: (i) high cache locality and contention-free per-road processing; (ii) bounded and stable neighbor degree, which is a prerequisite for scale invariance; and (iii) a natural unit for **dynamic load balancing**(Sec. 5.6.5), since patches can be migrated across workers without global repartitioning.

5.5. Efficient Parallel Processing Within Single Node

In high-fidelity urban traffic simulations like those handled by HiPUTS, computational performance must scale with both the complexity of the modeled environment and the number of autonomous agents (vehicles). Efficient parallel processing within a single computing node—especially those leveraging multi-core processors—represents a fundamental design pillar of the HiPUTS simulation engine. Without this optimization, even a well-distributed architecture would struggle to deliver real-time or near-real-time results in complex traffic conditions.

To meet these demands, HiPUTS incorporates a sophisticated task scheduling and multi-threading engine within each computational node. This local parallelism is essential for maximizing CPU utilization, reducing simulation step latency, and balancing workload when handling hundreds of thousands of vehicles in densely populated areas.

Parallelization at the intra-node level involves carefully designing simulation tasks that can be split among threads without introducing synchronization overheads or race conditions. In HiPUTS, several such phases of the simulation loop—including vehicle behavior updates, integration of incoming vehicle agents, and updates to adjacent patches—are designed as independent units of work. These units can be processed concurrently, allowing near-linear speedups relative to the number of cores available.

5.5.1. Task Definition

In the context of intra-node parallelism within the HiPUTS simulation architecture, computational tasks are the smallest schedulable units of execution. To fully exploit the concurrency offered by modern multicore processors, HiPUTS decomposes simulation logic into lightweight and independent tasks, each operating on a localized subset of the simulation environment. This subsection elaborates on the structure, role, and lifecycle of these tasks.

Granularity of Tasks. Previously described decision-update phases of 2-phase algorithm (see Section 5.2.5) can be divided into small execution tasks. Each task in HiPUTS is tightly bound to a single ***road*** *segment*—the fundamental unit of the simulation space—which acts as both a computational domain and a data container. Each task within each phase operate on a single road, iterating through all vehicles located on it. This road segment

in HiPUTS connects two junctions and holds a collection of vehicles currently present on that segment. Tasks do not span multiple roads, ensuring strong data locality and enabling isolated, conflict-free parallel execution across threads.

Two-Stage Task Model. In the 2-phase algorithm first stage (decision phase) of each simulation iteration, the system creates a batch of *decision tasks*, one for each road. These tasks iterate over all vehicles located on assigned road segment (assigned to this specific task) and invoke their behavior models (such as IDM and MOBIL) to determine next-step decisions.

After the whole decision phase has been completed, the simulator launches a second batch known as *update tasks*. These tasks similary to previous ones operate within single road as the smallest undividedd unit, but this time they revisit each vehicle within this road segment and update its state by modifying their current states.

This two-phase design ensures that all decisions are made based on a consistent and unmodified world state before any updates occur. As a result, the execution order of individual tasks within each phase becomes irrelevant, which enables safe and efficient parallel execution. Conceptually, this structure resembles a Map-Reduce-style synchronization barrier: the decision phase corresponds to the map step, where independent decisions are computed in parallel, and the update phase acts as the reduce step, where state changes are applied concurrently but consistently. This approach guarantees deterministic outcomes and avoids unintended interference between concurrently running tasks.

Benefits of Fine-Grained Task Design. This task definition and decomposition strategy provides multiple benefits:

- **Scalability:** By aligning tasks with road-level granularity, the system can generate a large number of lightweight units that scale linearly with network size. The task pool naturally adjusts to varying computational loads on different road segments.
- **Determinism and consistency:** Decoupling decision and update phases preserves consistency even under high parallelism.
- **Flexibility:** New models and behaviors can be added by modifying the task logic without altering the overall parallel execution framework.

In sum, task definition in HiPUTS not only facilitates efficient multi-threaded execution but also embodies the simulator's broader design philosophy: clean decomposition of computation, encapsulation of behavior, and scalability by construction.

5.5.2. Intra-Node Multithreading and Task Scheduling

Most modern computing environments offer multi-core processors capable of executing tasks concurrently. HiPUTS capitalizes on this by employing multi-threading for core simulation stages, enabling better utilization of hardware resources and further reducing execution time.

The simulation loop within each worker is designed to include several stages that are naturally parallelizable. Some of these operations include:

- Computing and updating vehicle parameters (e.g., position, velocity, acceleration),
- Integrating vehicles received from neighboring workers into the local domain,
- Updating border patch (described in section 5.6.1) states to reflect nearby activity in adjacent worker areas.

These operations fall under the category of *embarrassingly parallel tasks*—processes that can be executed independently without requiring constant synchronization. Each such task is encapsulated as a job and dispatched to a thread pool managed by an internal scheduler (or planner).

This thread pool is instantiated at system startup and persists throughout the simulation, avoiding the performance penalty of repeatedly creating and destroying threads. The number of threads typically corresponds to the number of physical cores on the CPU, ensuring maximal parallel coverage while avoiding unnecessary context switching. Each worker node in the HiPUTS framework is powered by such task scheduler that manages a pool of threads dedicated to processing simulation jobs. These threads execute a variety of discrete and concurrent tasks, such as computing acceleration values using the IDM (Intelligent Driver Model), determining potential lane changes using the MOBIL model, and preparing vehicles for inter-node migration. All of these calculations are required for every agent in the system at every timestep, and thus represent a substantial computational burden.

Both types of tasks are submitted to a centralized task queue managed by a thread pool scheduler. This thread pool is pre-initialized and persists throughout the lifetime of the simulation. This avoids the high overhead of thread creation and destruction, which would otherwise degrade performance significantly. The simulation kernel breaks these tasks down into smaller workloads at the granularity of road segments (lanes) as described in previous section 5.5.1.

The scheduler assigns tasks to threads in a work-stealing pattern, maximizing core utilization and balancing the computational load across the available threads. Each thread maintains a local queue of simulation tasks. If a thread finishes its queue early, it randomly selects another thread and attempts to *steal* a task from its queue. An intelligent job-stealing mechanism is also implemented within the thread pool to balance load among threads. This self-balancing mechanism ensures a uniform load across threads, even when the computational demands of different simulation jobs vary significantly. This dynamic redistribution of work prevents performance bottlenecks and allows for efficient CPU usage even in heterogeneous or imbalanced workloads. To ensure minimal overhead from task division and rebalancing, the job sizes are tuned to be neither too small (which incurs high dispatch cost) nor too large (which hinders parallelism).

Since each task only interacts with its own road segment and vehicles, the approach minimizes data contention and synchronization overhead. By deferring state mutations until the update phase, the system ensures that decision tasks are free from read-write conflicts. This fine-tuned multithreading infrastructure enables HiPUTS to maintain high simulation fidelity and performance even in resource-constrained environments.

5.6. Distributed Execution Design

Simulating a dense and expansive urban traffic network on a single computational node, even with modern multicore processors, quickly becomes impractical as the number of vehicles and intersections increases. Not only does it strain the processor, but it also risks turning each simulation run into a computational bottleneck where simulation time exceeds real-time. For real-world or near-real-time simulations—such as traffic control systems or emergency evacuation planning—this performance limitation is unacceptable.

To address this, HiPUTS employs a distributed simulation strategy wherein the road network is partitioned and each segment is handled by a different computational node (called *worker*). The number of nodes is user-configurable and may vary depending on the size and complexity of the simulation scenario.

As distributed simulations scale across multiple computing nodes, coordinating activity among them becomes increasingly complex. Unlike traditional, centralized approaches, HiPUTS adopts a decentralized strategy for run-time coordination to preserve scalability and eliminate single points of failure. This design enables each worker to operate independently while maintaining consistent and synchronized simulation behavior across the system.

Dynamic runtime coordination in HiPUTS encompasses communication protocols, peer-based time synchronization, and decentralized load redistribution. Together, these mechanisms ensure that the simulation proceeds coherently despite differences in local workloads, asynchronous message arrivals, and uneven agent distribution. The system achieves this through lightweight message passing, autonomous decision-making, and asynchronous processing that decouples computation from coordination.

This section details the such components of HiPUTS' as dynamic coordination system: the communication model that enables inter-node messaging, the decentralized synchronization approach that ensures temporal alignment without a global clock, and the adaptive load redistribution algorithm that maintains balanced computation across the simulation space.

5.6.1. Patch Management and Inter-Node Coordination

Scalability in HiPUTS is achieved through an intelligent spatial decomposition of the road network into independently computable units known as *patches* defined in section 5.4.1. Each patch comprises a contiguous group of roads and intersections, carefully selected to maintain spatial cohesion and limit inter-node communication. A core design principle of this architecture is that no vehicle should be able to traverse an entire patch in a single simulation time step. This ensures consistency across the distributed system by avoiding complex, multi-hop transitions that would complicate state synchronization and prediction.

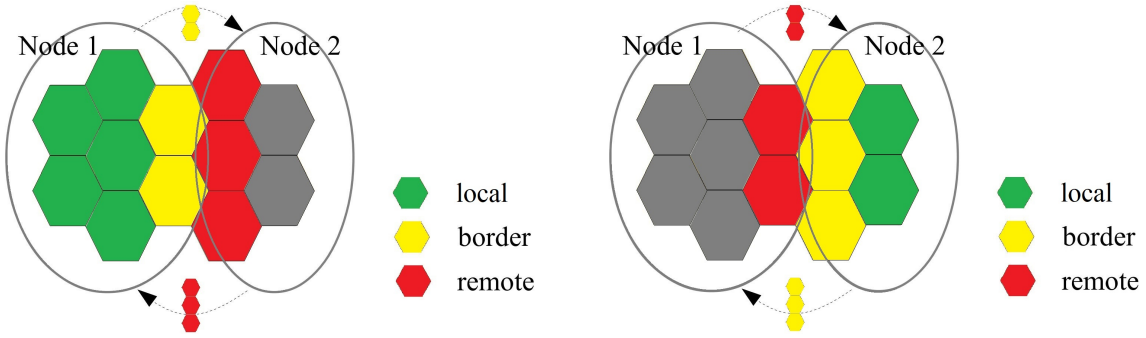
The network is initialized either from OpenStreetMap data or user-defined synthetic maps and then transformed into a directed graph where vertices represent intersections and edges denote roads. This graph is further decomposed into patches, which are then mapped to worker nodes for simulation. Notably, the patches themselves form a higher-level graph, in which each patch is treated as a node with up to six neighboring patches—reflecting spatial adjacency due to its hexagon nature.

Each patch is assigned a computational cost, calculated based on its total lane length, which serves as a proxy for expected simulation workload. These cost-weighted patches are then clustered using the **k-means** algorithm [1], producing balanced partitions that evenly distribute the workload among the available computing nodes. While alternative partitioning methods could be considered in the future, this clustering strategy provides a pragmatic balance between simplicity and effectiveness.

At the start of the simulation, each node receives a set of patches to manage. From the perspective of any given node (worker), patches are divided into three functional categories:

- **Local (Internal) Patches**: Patches that are fully contained within the worker's domain and are exclusively updated and simulated by that worker. Only the owning worker has any access to them.
- **Border (Internal) Patches**: These patches lie at the boundary of a worker's region and directly neighbor patches owned by other nodes. While simulated locally, they interact with remote data and require frequent synchronization. In other words they are the patches in the boundary between worker's local patches and its remote patches. Though still simulated locally (as the local patches these are also changed by the owning worker), but they require awareness of vehicles and infrastructure from neighboring patches of other workers (remote patches).
- **Remote (External) Patches**: These are the patches which state is not changed by the owning worker but rather changed by neighboring worker. These represent the border patches of neighboring workers. Although not simulated directly, their state is mirrored on the local node to enable accurate decision-making in vehicle behavior in regions of border patches.

An illustration of this classification is shown in the Figure 5.5. This shows an exemplary division of some environment on patch abstraction level between 2 nodes. In the left Figure 5.5a the classification of patches can



(a) Perspective of Node 1

(b) Perspective of Node 2

Figure 5.5: Internal, border and remote patches of exemplary division and the exchange of patches' states between nodes (workers)

be observer from the perspective of node 1 (worker 1). The local patches (colored with green) and border patches (colored with yellow) together form an internal patches of the worker. The worker needs to update and manipulate the vehicles within its internal patches. For this purpose the workers need to know the state of the vehicles within the adjacent patches. In case of the local patches, the adjacent patches are either the local or border patches, which are both manipulated by the same worker, so there is no problem here. But in context of border patches, the adjacent patches can be either internal patches of this worker or external patches (remote) of node 1, that are out of responsibility zone of this worker for simulation, nevertheless this worker has to have the knowledge about the state of those remote patches (this information is exchanged as ilustrated in the figures with arrows).

On the other hand in the right Figure 5.5b the very same environment divided into patches can be observed, but the classification of patches is from the perspective of node 2 (worker 2). The previously shown as remote patches from node 1 perspective in this case become border patches and the border patches of node 1 become remote patches for node 2. The patches colored in grey on the very right of the Figure 5.5a are not visible to Node 1 and analogously in second case for Node 2 in the Figure 5.5b the patches colored with grey color on the left are not accessible or visible for Node 2.

During each simulation step, nodes exchange the current state of their border patches with the nodes that manage adjacent remote patches. This communication is bi-directional: each worker receives the up-to-date state of remote patches (corresponding to the border patches of neighboring nodes), which includes all necessary vehicle and infrastructure information required for consistent behavioral modeling and for computing decisions of all cars in the internal and border patches.

This design ensures that vehicles in local or border patches can make decisions based on a complete and up-to-date view of their surroundings, even when those surroundings span multiple workers. Because of the patch structure and strict movement constraints, each vehicle's position, speed, route progression, and interaction with nearby traffic lights or junctions can be updated in a behaviorally correct and decentralized fashion.

The patch-based framework thus serves a dual purpose: enabling highly parallelized simulation by distributing workload across nodes and preserving the integrity of traffic dynamics near patch boundaries. By combining partitioning with structured communication, HiPUTS achieves both scalability and fidelity in simulating complex, large-scale urban traffic systems.

5.6.2. Decentralized Initialization and Execution Model

The design of HiPUTS (High Performance Urban Traffic Simulator) addresses the challenge of achieving distributed scalability. To accommodate complex, high-density urban environments while preserving computational

efficiency, the simulator employs a modular and distributed execution model. This model ensures seamless parallelism and coordinated simulation across multiple computational nodes. To maintain both scalability and resilience, HiPUTS adopts a fully decentralized approach especially to execution stage (proper simulation part). Unlike traditional master-worker architectures, where a central coordinator manages all activities, HiPUTS decentralizes responsibilities to eliminate bottlenecks.

Figure 5.6: An example of splitting the road network graph of central Kraków across two simulation nodes.

HiPUTS distributes the computational simulation problem across a user-defined number of nodes. Each node runs a single worker process responsible for handling a portion of the road network used in the simulation. While each worker manages the vehicles within its assigned area, it also keeps track of the state of its immediate neighboring regions. An example of how the road network graph for central Kraków can be divided between two nodes is shown in the Figure 5.6.

The system is able to achieve super-scalability when its architecture avoids relying on a central component responsible for managing communication and synchronization. In centralized systems, routing these operations through a single control unit creates a communication bottleneck, increasing latency and limiting the system's ability to scale with additional processing units. Moreover, centralization negatively affects not only the simulation runtime but also other critical stages of execution such as initialization, distributed data collection, final aggregation and collection of results. These aspects often impose additional overheads that scale poorly with system size. This centralization contradicts the principle of scale invariance, which requires that the workload and communication overhead per node remain constant as the system grows. As highlighted by Engelmann and Geist [28], achieving super-scalability necessitates decentralized designs where each component operates autonomously and coordination occurs locally.

In the case of the developed simulator, completely eliminating the central component during the initialization phase would be highly challenging. The simulation nodes, which need to communicate with each other during

runtime, must be aware of one another and at least know each other's identifiers. Additionally, the road network has to be properly divided among the worker processes, and each worker must be informed about the specific area it is responsible for. That is why at startup, one compute node assumes a temporary role as the *initialization server*. This node is responsible for:

- Importing the road network and configuration data.
- Executing the graph partitioning algorithm to divide the network into patches.
- Establishing connections with the other worker nodes
- Assigning patches to available worker nodes.
- Broadcasting neighbor relationships and initial simulation parameters.
- Sending a signal to each worker to initiate the simulation

Once this setup phase is complete, the initialization server transitions into a standard worker node and from this point forward the whole simulation is distributed and it is not centrally synchronized during runtime. All nodes then operate in a fully peer-to-peer configuration. This decentralization has several key advantages:

- **No Single Point of Synchronization:** The simulation can proceed even if some desynchronization occurs within non-adjoining nodes. Additionally lack of bottleneck as the nodes do not need globally synchronize their state.
- **Bounded communication:** The number of connections and communication cost per node remains bounded and scale-invariant.
- **Reduced Latency:** Peer-to-peer messaging enables faster communication between nodes responsible for adjacent patches.
- **Improved Scalability:** As more nodes are added, there is no growth in coordination costs since there is no central bottleneck.

To facilitate this, HiPUTS implements a low-latency, event-driven messaging system in which each worker exchanges state information with its neighboring nodes every simulation step. These messages include vehicle parameters for agents approaching or leaving patch boundaries, as well as synchronization signals that ensure consistency of simulation time across the distributed environment.

Each worker communicates with a fixed number of neighboring nodes. However, this configuration can change due to the use of load balancing mechanisms, which are discussed later in this work. Once the simulation steps begin, the system no longer relies on any central component to manage or coordinate node operations. Throughout the simulation, each worker exchanges messages only with its immediate neighbors, and both communication and synchronization are handled in a fully decentralized manner. Such architecture is designed to support scale invariance and enable super-scalability.

5.6.3. Communication and Message Passing

Efficient communication between distributed nodes is essential in any high-performance simulation system, especially one that emphasizes decentralization. HiPUTS employs TCP-based Java socket communication for message exchange between worker nodes, ensuring reliable and ordered data transmission.

The communication protocol in HiPUTS is designed with minimalism and predictability in mind. During simulation initialization, the following message exchanges occur between the server and each worker:

1. Workers connect to the server using its known address and transmit identification data for establishing future connections.
2. The server confirms network loading and patch partitioning, signaling workers to begin data import.
3. The server assigns each worker its patch set and informs them of their neighboring workers.
4. Workers acknowledge successful initialization and readiness.
5. The server broadcasts a final signal to start the simulation.

After the simulation starts, the server becomes passive, and all subsequent communication occurs directly between workers. In each simulation step, a worker exchanges two categories of messages with each of its neighboring nodes:

– **Vehicle Migration Updates**: Information about agents entering or leaving the local domain (internal patches).

– **Remote Patch State Reports**: Summaries of vehicle states in neighboring border zones (located in remote patches) for interaction modeling.

When load balancing is activated (described in section 5.6.5), additional messages are introduced to facilitate patch transfer and update neighbor state consistency.

To handle incoming messages without blocking simulation progress, dedicated receiver threads are used. Each communication link with a neighbor has its own thread, which listens for and processes messages asynchronously. This architectural choice decouples computation from communication and helps maintain real-time or near-real-time performance even under heavy network activity.

5.6.4. Peer-Based Time and State Synchronization

Unlike centralized systems where a master clock governs simulation timing, HiPUTS relies on decentralized synchronization. This is both a design constraint and an opportunity—by avoiding a central synchronization authority, HiPUTS eliminates potential bottlenecks but must solve time coordination in a peer-to-peer manner.

Each worker maintains its own local simulation clock, progressing to the next time step only after it has completed the current step and received all required messages from its neighbors. These messages serve as implicit synchronization barriers. For example, a worker can only finalize its vehicle update phase after processing incoming ghost patch data and vehicle migration information.

This strategy ensures causal consistency and allows each worker to simulate in lockstep with its neighbors. In scenarios involving dynamic patch reallocation or high agent movement across boundaries, the message-based synchronization ensures that all interactions remain consistent without the need for a centralized scheduler.

The decentralized synchronization strategy mirrors physical systems where locality governs causality. It allows HiPUTS to scale horizontally with minimal coordination overhead, supporting massive simulations across many nodes without a master clock or centralized state manager.

5.6.5. Runtime Load Redistribution

A critical challenge in large-scale simulations is uneven computational load. Because vehicles are not evenly distributed across the map—and tend to cluster in high-traffic zones—some workers may become overloaded while others remain underutilized. HiPUTS addresses this challenge using a decentralized load balancing mechanism based on patch transfer.

Each worker periodically evaluates its own workload (primarily based on vehicle count) and compares it with that of its neighbors. If a significant imbalance is detected, the overloaded worker may choose to transfer one or more border patches to a less loaded neighbor. The decision is made locally and does not require global coordination.

Two decision algorithms are implemented for triggering patch transfers:

1. **Tri-State Rule**: Based on relative workload difference—no transfer if below 5%, one patch transfer if above 5%, and multiple patches if above 25%.

2. **PID-Controlled Strategy**: Predicts future load based on past data using a Proportional-Integral-Derivative (PID) controller and determines how many vehicles (and corresponding patches) should be moved.

To prevent situations where all neighboring nodes simultaneously offload patches to a single worker—potentially overloading it—the system introduces a *ticket-based negotiation mechanism*. Each worker is issued a finite number of communication tickets, limiting how often it can receive new patches. The tickets are evaluated based on the current simulation time and help avoid cyclical or conflicting transfers.

All workers have access to the entire road network topology, so patch transfers only involve sending agent states and metadata, not map definitions. Once a transfer is initiated, all affected workers are informed, and the recipient updates its internal state accordingly.

The dynamic balancing system is crucial for preserving the *scale invariance* of the simulator. Without it, nodes would diverge significantly in computation time, destroying the synchronicity and undermining the simulation's scalability.

5.7. Detailed overview of simulation algorithm

The simulation process is executed in parallel by multiple worker units, each responsible for handling a designated portion of the total simulation steps. These workers operate on separate computational nodes, allowing the simulation workload to be distributed across a high-performance computing environment. The overall process of simulating a single step follows a structured algorithm that consists of several phases, which are visually summarized in Figure 5.7.

At each simulation step, individual workers compute the changes in vehicle parameters, representing how vehicles evolve over a simulated time slice. Although neighboring workers may occasionally operate on slightly different simulation steps due to computational variance, such inconsistencies are resolved through localized synchronization. This is achieved via direct inter-worker communication, typically through asynchronous message exchanges. This localized approach avoids global synchronization, a method known to introduce significant performance bottlenecks in large-scale parallel systems. A worker can advance to step $n + 1$ once it confirms that all adjacent workers have completed their computations for step n and have transmitted the necessary state information.

The simulation algorithm is logically divided into two major components: decision-making and state updating. This separation enables the algorithm to exploit fine-grained parallelism by decomposing the problem into smaller tasks. During the decision phase, vehicles determine their next actions based on current traffic conditions, without modifying the actual simulation state. This ensures that every vehicle operates based on a consistent snapshot of the environment at step n, and allows the planned changes to be stored for use in the subsequent update phase.

In the decision stage, each vehicle computes its intended future state—such as acceleration, deceleration, speed adjustments, lane changes, and target positions—based on internal models like the Intelligent Driver Model (IDM)

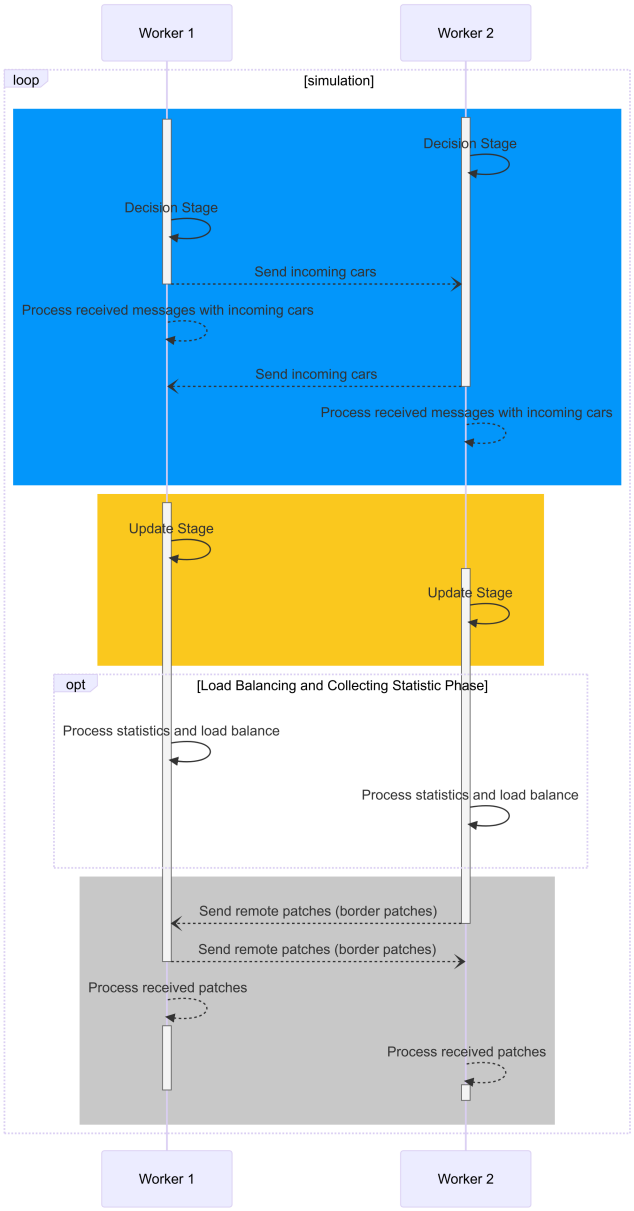


Figure 5.7: Overview of single-step simulation flow in HiPUTS using two distributed workers.

and the relative state of nearby vehicles. These decisions are stored in vehicle-specific buffers and are not applied immediately, which prevents data inconsistencies during concurrent execution.

To maximize performance, decision-making tasks are assigned at the level of individual lanes. Each lane is managed independently by a task that handles all vehicles on that lane. These tasks are distributed across a thread pool sized to the number of CPU cores available on the node, ensuring efficient parallel execution. Vehicles that opt to change lanes are temporarily stored in lane-specific buffers to maintain safe and synchronized access across threads.

Once all decisions are made, workers initiate the exchange of vehicle data relevant to boundary lanes—those that interact with adjacent workers. These exchanges are carried out asynchronously, as depicted with dashed lines in Figure 5.7. Upon sending out these updates, a worker enters a waiting phase (illustrated by a block in the timeline) until it receives all necessary data from its neighbors. This synchronization is still decentralized and localized, preventing delays associated with centralized coordination. After successful data reception, the worker

transitions to the update phase.

In the update stage, vehicles apply their previously computed decisions, updating their physical state in the simulation environment. This stage, like the decision-making phase, is parallelized using the same thread-pool strategy, allowing efficient use of available computational resources.

An additional, optional component of the simulation algorithm is the load-balancing mechanism, which dynamically adjusts computational responsibilities among workers. This is guided by performance metrics collected during execution and is mentioned in Section 5.6.5.

The final stage involves communicating updated border patch states to neighboring workers. This ensures that the environmental data remains consistent across simulation boundaries and supports accurate calculations in subsequent steps. Once all these tasks are completed, the system is ready to proceed to the next simulation iteration, continuing the loop. This final synchronization phase is marked in grey in the sequence diagram in Figure 5.7.

5.7.1. Start-to-End Simulation

The operation of the HiPUTS simulator can be broadly divided into three main stages: initialization, simulation execution, and termination. Among these, the simulation phase is the most resource-intensive and critical, forming the core of the system. Consequently, particular attention is given to its super-scalability, as this phase dictates the simulator's overall performance and efficiency.

This section outlines the simulation process in a structured and concise manner.

Initialization Phase

During initialization, the server process plays a central coordinating role. It handles key responsibilities such as dividing the map into patches, assigning these patches to different workers, and broadcasting the command to initiate the simulation.

The key steps of the initialization process are as follows, with many of them executed in parallel across nodes:

1. Server and worker processes are instantiated and initialized.
2. Workers establish connections with the server.
3. The server loads the road network data, partitions the map into patches, writes road-related information to external files, and notifies workers that the network is ready for loading (this applies, for instance, when the map originates from OSM data).
4. Workers read the full road network from the prepared files.
5. The server assigns patches to workers and sends them relevant configuration messages.
6. Workers prepare their assigned segments of the road network, including generating initial vehicle data for their specific areas.
7. After completing initialization and vehicle generation, each worker sends a confirmation message back to the server.
8. The server then sends a final signal to start the simulation phase.

Once initialization is complete, the server process transitions into a worker role to contribute to the simulation. This design choice ensures optimal utilization of all computing resources, avoiding idle nodes.

Simulation Execution

The core simulation is carried out by the workers in a series of discrete time steps. During each step, workers update the states of vehicles to reflect their expected real-world behavior over the duration of the step.

It is possible for neighboring workers to be at slightly different stages of the simulation. These temporary desynchronizations are resolved at predefined synchronization points using inter-process communication.

A typical simulation step involves the following operations:

1. Vehicles independently determine their next state, including new positions and parameter values (this step is parallelized).
2. Vehicles that plan to leave the current worker's region are sent to neighboring workers (one message per neighbor).
3. Each worker receives incoming vehicles from adjacent workers (parallelized synchronization).
4. Workers update their internal state by removing outgoing vehicles and updating the positions and parameters of remaining and newly arrived vehicles (executed in parallel).
5. A decision is made whether to perform dynamic load balancing. If required, the worker executes the balancing procedure described below.
6. Workers send updated information about border regions to their neighbors (parallelized).
7. Workers receive data related to ghost patches and update their local map state accordingly (parallelized).
8. If defined by the simulation parameters, new vehicles are generated or existing vehicle routes are extended (parallelized).
9. Statistical metrics are collected for performance monitoring.

The majority of the computations in a simulation step are parallelized to fully utilize the available cores. Tasks that are not parallelized are typically minor and would not benefit significantly from multithreading due to their small size.

Dynamic Load Balancing The simulator includes a dynamic load balancing mechanism designed to redistribute workload between workers when imbalances arise. The process proceeds as follows:

1. A worker selects a neighboring worker to offload part of its workload.
2. The most overloaded patches are identified and serialized (in parallel) before being sent to the selected neighbor.
3. Notifications about ownership changes are sent to other neighboring workers, especially if the transferred patches were previously ghost patches. If there are no such patches, an empty message is sent for synchronization.
4. A consistency update is performed to ensure all workers have an accurate view of patch ownership.
5. The worker receiving the patches integrates them and places the transferred vehicles in its region (parallelized).
6. All workers update their ghost patch ownership information based on received notifications.

Although computationally demanding, load balancing is essential for maintaining simulation efficiency, especially in large-scale or long-duration simulations. It plays a vital role in supporting the simulator's super-scalable architecture by helping ensure uniform workload distribution.

Simulation Termination

Upon completion of the designated simulation steps, all workers initiate the shutdown process. Before exiting, they send a summary of the simulation data to the server, which stores this information for post-analysis.

The shutdown sequence includes the following steps:

1. Workers notify the server that the simulation has concluded.
2. Performance and simulation data are sent to the server for evaluation.
3. The server broadcasts a message to all workers signaling the end of the simulation.
4. Workers receive this signal and terminate their processes.
5. The server logs the received data to external files.
6. Finally, the server process terminates.

This final data collection phase is not parallelized or distributed. However, the collected data focuses on the performance of the simulator itself, such as message sizes and execution times, rather than the actual simulation output. Therefore, this stage was excluded from the scalability benchmarks discussed in following Chapter 6.

The simulator architecture and the mechanisms outlined in this chapter are designed to maintain consistent workload distribution across nodes, regardless of the total system size. This property—known as scale invariance—is a key feature of systems designed for super-scalability.

To verify that HiPUTS meets the criteria for super-scalability, its performance and the efficiency of its core algorithms must be rigorously evaluated. The results of such evaluations are presented in detail in Section 6.3.1 and 6.3.2.

6 Evaluation

To verify the performance and advantages of the HiPUTS algorithms, a comprehensive scalability analysis was conducted. This chapter presents results from both strong and weak scalability experiments, executed on a single node and across multiple nodes, designed to evaluate the simulator under varying workloads and system configurations. Particular emphasis is placed on examining the system's ability to achieve *super-scalability*, with a focus on *scale invariance*—the property that ensures a consistent workload per processing unit even as the number of computational nodes and the overall problem size grow proportionally.

Among the two scalability models, weak scalability, inspired by Gustafson's Law (described in section 1.1.3), plays a more critical role in this context. It reflects realistic use cases where expanding problem size accompanies growing computational capacity. This approach not only provides a more meaningful measure of distributed efficiency, but also captures the overhead introduced by communication and coordination in multi-node environments. The experiments presented in this chapter aim to demonstrate how the HiPUTS architecture maintains performance under such scaling conditions and where potential bottlenecks may arise.

The scalability experiments carried out in this chapter can be classified into two main categories: experiments executed on a single computing node (described in Section 6.2) and those conducted across multiple nodes (described in Section 6.3). Running the simulation on multiple nodes not only highlights the potential for super-scalability but also incorporates the influence of communication costs and the overhead inevitably associated with distributed execution.

6.1. Testing environment

All experiments were performed on the Ares[1] supercomputer, located at ACK Cyfronet AGH in Kraków, Poland, as part of the PL-Grid[2] infrastructure. At the time of testing, Ares was ranked 442nd on the TOP500 list[3]. The system is equipped with Intel Xeon Platinum 8268 24-core processors running at 2.9 GHz, interconnected via Infiniband EDR. Ares provides a total of 37,824 processing cores. The experiments utilized up to 96 nodes, each containing exaclty 48 cores, yielding a maximum of 4,608 active computing cores for maximal testing setup. Each HiPUTS worker process was mapped to a separate node.

For both scalability studies, a synthetic traffic network designed for a controlled benchmarking environment was used, which allows proportional growth of the problem size. The network consists of perpendicular roads connected at boundaries to form a toroidal layout (Figure 6.1). Each intersection links to four neighbors via bidirectional roads (eight lanes total, two per direction). The distance between adjacent intersections was fixed at 500 meters, and each simulation patch comprised a 2×2 grid of connected intersections.

A typical configuration for weak scalability experiment assigned a 256x256 intersection section of the network to each worker node. Given the spacing between intersections, this setup resulted in more than 131 million

[1] https://www.cyfronet.pl/komputery/18486,artykul,superkomputer_ares.html
[2] http://www.plgrid.pl/
[3] https://www.top500.org/lists/top500/list/2024/06

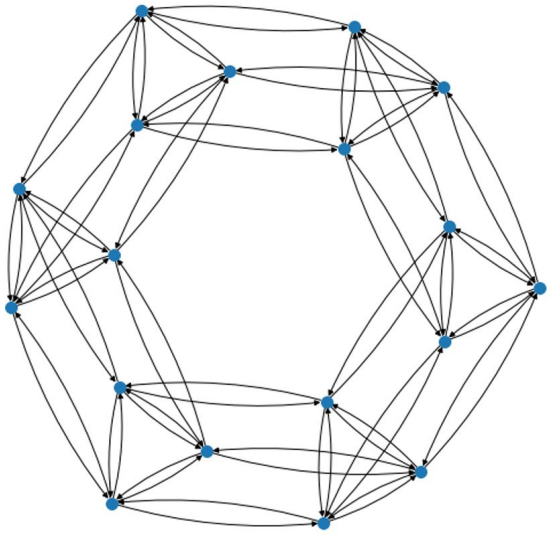


Figure 6.1: An example of a road network modeled as a torus.

kilometers of road length per worker when counting both directions. If measured as single-lane, two-way roads, this still represented over 65 million kilometers—substantially more than the length of Poland's entire public road network, which stood at roughly 420,000 km in 2022 according to Polish General Directorate for National Roads and Motorways [36].

The simulation progressed in discrete steps, with each step representing one second of real time. Vehicle density was varied across two test scenarios: in the first, vehicles were spaced on average every 30 meters, while in the second, the average spacing was increased to 60 meters. Each worker was initialized with either approximately 4 million or 2 million vehicles depending on the density setting. The total vehicle count across the simulation remained constant, as vehicles reaching their destinations were instantly replaced with new ones.

HiPUTS employed a dynamic load balancing method that redistributed road network patches between workers. Patches were transferred from the most heavily loaded node to the one with the lowest computational burden. It utilized a Tri-State Rule load balancing strategy (described in Section 5.6.5), enhanced with a token-based mechanism to manage patch transfers between workers. This decentralized approach enabled dynamic redistribution of workload by transferring patches from overloaded nodes to those with lower processing demands. Tri-State Rule delivered results comparable to a PID-based method, while avoiding the complexity and parameter tuning required by the latter.

Each experiment ran for 1,500 simulation steps, and each step was set up to simulate 1 second of real time. Performance metrics were calculated only from the final 1,000 steps. This approach minimized distortion from JVM warm-up and other transient overheads at startup. Most configurations were tested at least 10 times to ensure statistical relevance. Exceptions included simulations using 1 and 96 nodes under the low-density (60 meters between vehicles) configuration, which were executed 8 and 4 times respectively due to hardware constraints. In weak scalability tests involving multiple nodes, the size of the road network and the number of vehicles were scaled proportionally with the number of workers. For instance, using 4 nodes, the simulator modeled a 512x512 grid of intersections and over 17 million vehicles. The largest experiment, run on 96 nodes, simulated a 2048x2048 intersection network containing approximately 280 million vehicles.

6.2. Single-Node Scalability Evaluation

Evaluating performance within a single compute node helps separate the effects of parallelized and sequential computations. These results indicate the effectiveness of the implemented solutions and identify areas requiring further improvement. Although single-node (intra-node) performance does not directly determine super-scalability (scaling across nodes), it significantly affects throughput and achievable speedup. All experiments described in this section were conducted using a single worker per node, which initially also performed server tasks. Most computations within the worker were executed in parallel using software threads. These intra-node experiments are particularly useful for evaluating the performance of implemented algorithms without the overhead introduced by multi-node (inter-node) communication, and when running with varying numbers of threads.

6.2.1. Weak Scalability

In weak scalability tests (per Gustafson's law), the problem size was increased proportionally to the number of utilized CPU cores. Both the road network and number of vehicles in the system were expanded.

Experiment Description

Experiments were performed on a single node of the Ares[1] supercomputer, which features nodes equipped with 48 cores. A synthetic torus-shaped road network was used, consistent with previous multi-node tests. Each road segment was 500 meters long.

The simulation started with 1 thread and a map size of 37x37 intersections with 91,022 vehicles. As the number of cores increased, so did the network size, reaching 256x256 intersections and 4,369,068 vehicles at 48 cores. The vehicle density remained consistent across all tests (approx. one car per 30 meters).

Each test ran for 1500 simulation steps (each step simulating 1 second of real time), with the last 100 steps used for performance analysis. Each configuration was repeated 10 times to ensure reliable results. Other parameters were consistent with the multi-node experiments.

Results

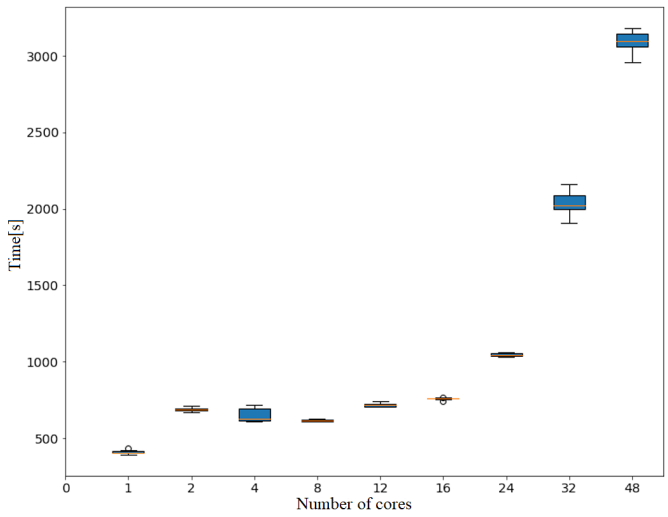


Figure 6.2: Weak scalability on single node - Total simulation duration time depending on the number of cores.

[1] https://www.cyfronet.pl/komputery/18486,artykul,superkomputer_ares.html

Ideally, weak scalability tests should maintain a constant runtime as the number of cores increases. This was nearly achieved for 2 to 16 cores (see Fig. 6.2). However, runtime and average iteration time increased slightly when using 2 threads due to the JVM's common thread pool mechanism, which creates one fewer thread than the number of cores. As a result, both decision and update stages were executed sequentially in single-threaded configurations (Fig. 6.3). Both during computations on 1 and 2 cores, the thread pool contained only a single thread, which caused the decision-making and vehicle state update computations to be executed sequentially (Fig. 6.3). In subsequent experiments using a larger number of cores, the number of software threads increased with the problem size, resulting in good scalability up to 16 utilized cores.

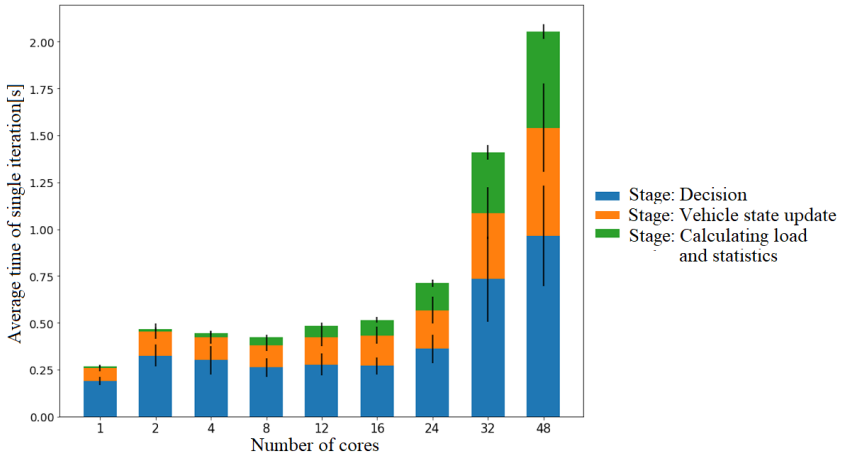
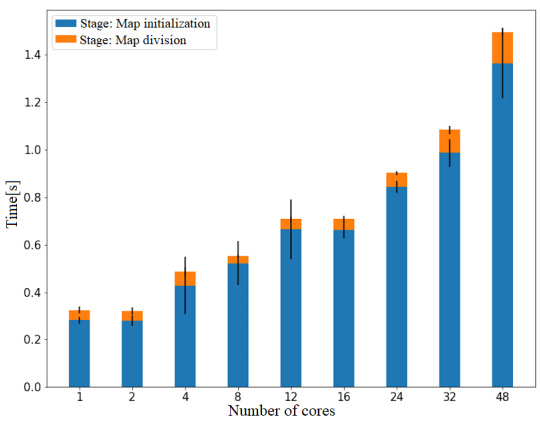
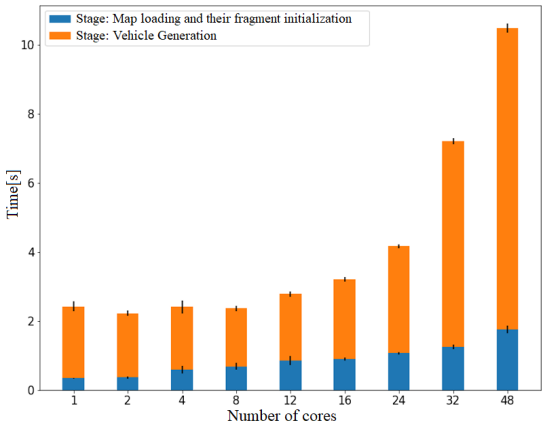


Figure 6.3: Weak scalability on single node - Average duration of a simulation step broken down by individual stages.

Good scalability was observed up to 16 threads. Beyond that, performance degraded, particularly at 24, 32, and 48 cores. This was due in part to the increased cost of computing statistics and vehicle decision-making logic. These stages are fully parallelized, assigning one task per road and distributing them across threads.



(a) Server initialization time



(b) Worker initialization time

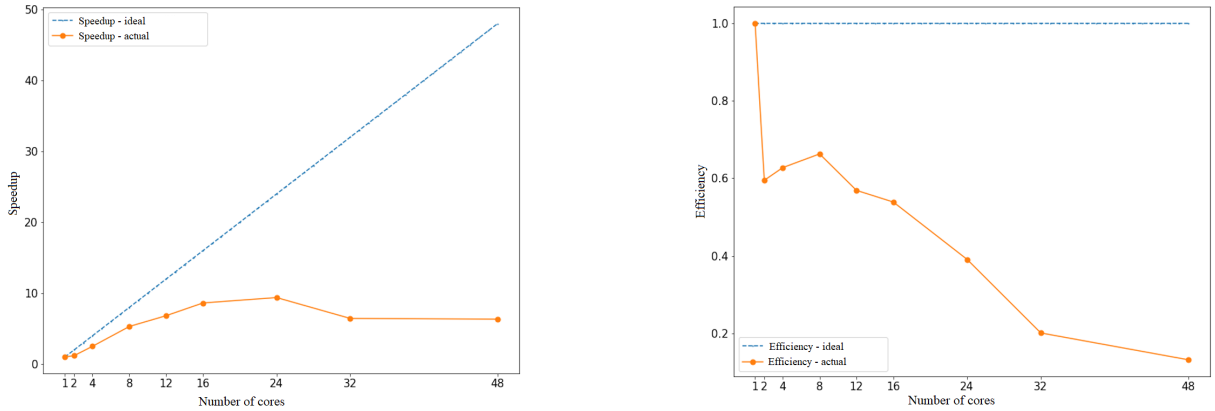
Figure 6.4: Weak scalability on single node - Initialization duration

More problematic is the increased duration of the stage in which vehicles make decisions about their behavior in a given simulation step, as well as the stage responsible for updating vehicle parameters. These stages are fully parallelized—each road segment is assigned a task that processes the vehicles located on it. The resulting group of tasks is then distributed across the available threads.

A similar situation occurs during the initialization of the worker process (Fig. 6.4b). The duration of the vehicle generation phase remains relatively constant in tests using up to 16 cores, but begins to increase in subsequent experiments. This phase has been parallelized in a manner analogous to the simulation step components described above.

It can thus be assumed that the increase in computation time for 24, 32, and 48 cores has a common cause across all three stages. One possible explanation for this behavior is a sequential task creation process. As the road network grows, the number of roads requiring processing increases, which demands more time. While this is a plausible justification, it does not explain why the duration of these stages remains constant for a smaller number of cores. Therefore, this explanation alone is insufficient.

The increase in computation time when using more than half of the available cores suggests the possible impact of Hyper-threading. With this mechanism enabled, two logical cores would share a single physical core, potentially leading to contention for resources and longer computation times. However, the Ares supercomputer node architecture consists of two interconnected processors, each with 24 physical cores. Hyper-threading is not enabled, thus this scenario does not hold.



(a) Speedup

(b) Efficiency

Figure 6.5: Weak scalability on single node - Performance metric results (according to Gustafson's law)

Another hypothesis relates to the processor's cache size. As the problem size increases, the limited cache memory could cause more frequent accesses to RAM, resulting in additional time overhead. Application profiling was conducted to examine the number of cache misses, but the results did not confirm this hypothesis either.

In collaboration with staff at AGH's Cyfronet, the most likely cause for the increased computation time beyond 24 utilized cores was identified. The Ares supercomputer nodes employ a Non-Uniform Memory Access (NUMA) architecture. This architecture divides available RAM into memory banks, each optimized for fast access by an assigned group of cores. Each processor within a compute node has two NUMA banks, with 12 cores assigned per bank. While all cores can access the entire node's memory, access to a memory bank associated with a different processor takes approximately twice as long compared to accessing its own local memory. The difference in access time between two banks on the same processor is negligible. Once the system uses more than 24 cores, memory from the second processor is engaged, leading to slower performance and explaining the observed results.

Figure 6.4 illustrates the increasing duration of phases associated with loading and initializing the road network. This behavior clearly correlates with the increased problem scale and the sequential processing algorithm for this stage, which, as mentioned, is difficult to optimize.

Due to the significant increase in average simulation step duration, the system does not achieve high speedup in tests using more than 16 cores (Fig. 6.5). Experiments using fewer cores achieved satisfactory speedup and efficiency values.

Summary

The computations performed by the worker, excluding tasks related to communication and load balancing, scale well only up to a certain point. Utilizing more than 24 cores causes the total simulation time to increase compared to running a correspondingly reduced problem on fewer cores. This behavior is due, among other things, to the architecture of the supercomputer used in the experiments. Therefore, it may be worth considering repeating the experiments with two worker processes running on a single node. In such a setup, each worker should be restricted to using only the cores and NUMA memory banks assigned to the processor it runs on.

Unfortunately, it cannot be said that the simulator always scales well within a single node. This property depends on the supercomputer's architecture; however, it does not prevent achieving super-scalability at the node level, which is one of the primary objectives of the simulator.

6.2.2. Strong Scalability

During the scalability study based on Amdahl's Law, as additional compute units are added, each core should perform progressively fewer computations. The expected outcome is a continuously decreasing program execution time.

Strong scalability analysis helps assess the impact of sequential computations on the algorithm. Throughout the experiments, the overall problem size remained constant.

Experiment Description

The maximum problem size was constrained by the memory available to a single core, resulting in a test scenario with 1,456,354 vehicles on a 148x148 torus-shaped network. Vehicle density was again approx. one car per 30 meters.

As in weak scalability tests, each run consisted of 1500 time steps (each representing 1 second of real time) and was repeated 10 times in total. The same configuration of the Ares node with 48 cores was used.

Results

As expected, the simulation runtime decreases with the increasing number of cores (Figure 6.6). The same trend can be observed for the individual stages that comprise the computation process (initialization—Figure 6.8, and simulation step—Figure 6.7).

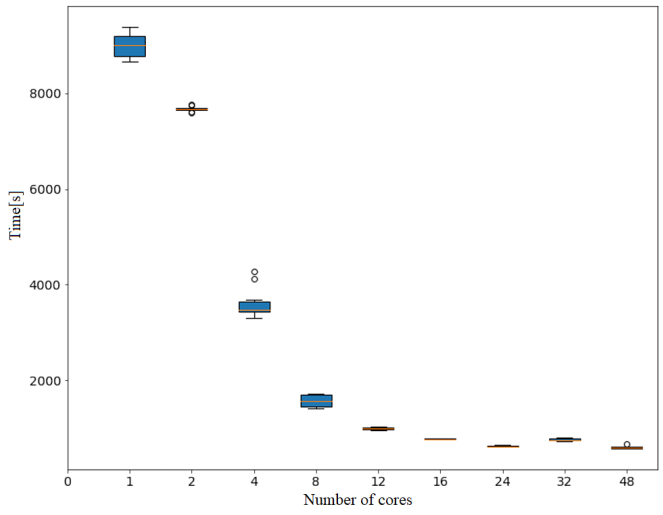


Figure 6.6: Strong scalability on single node - Total simulation duration time depending on the number of cores.

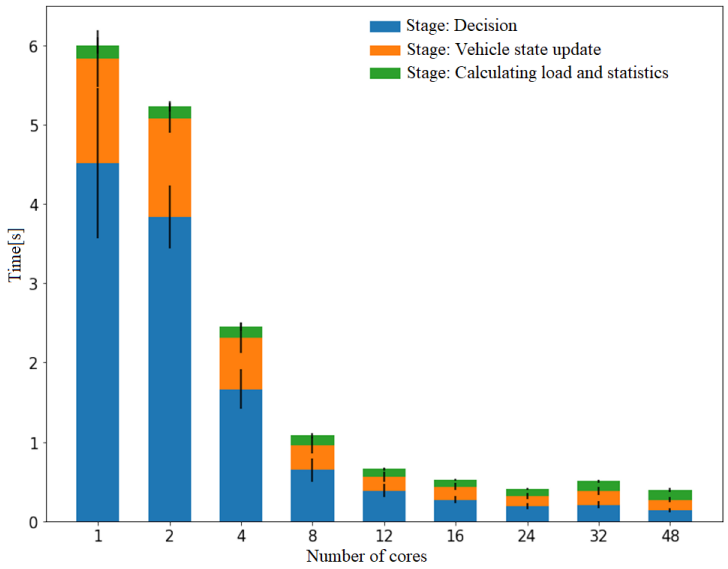
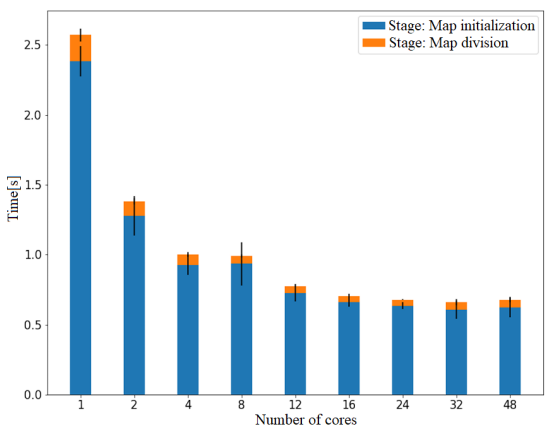
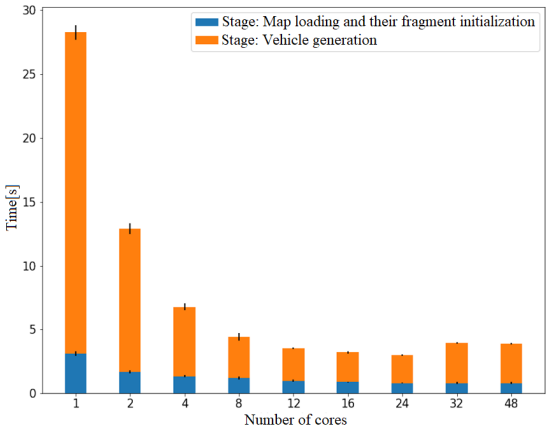


Figure 6.7: Strong scalability on single node - Average duration of a simulation step broken down by individual stages.

In the chart showing the average duration of a single simulation step (Figure 6.7), a disproportionate drop in computation time is noticeable when running on two cores compared to one. This is due to the size of the thread pool in the scheduler mechanism, which is equal to the number of cores minus one. As a result, in both experiments, the parallel tasks assigned to the scheduler were executed by a single thread.



(a) Server initialization time



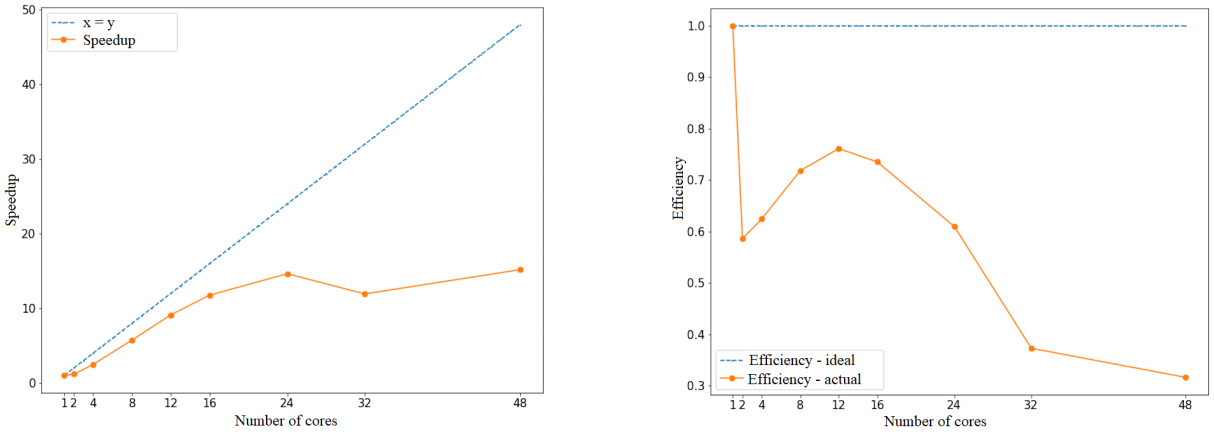
(b) Worker initialization time

Figure 6.8: Strong scalability on single node - Initialization duration

This effect can be seen across all charts presenting the simulation and its stages' runtime. Based on this, one can infer, among other things, that the shortest achievable duration of a single simulation step is approximately 0.5 seconds, and the duration of the vehicle generation stage should be around 4 seconds. The results also indicate that using more than 24 cores does not yield significant performance gains. Combined with the conclusions from the intra-node weak scalability study (Section 6.3.1), this forms a strong argument for conducting experiments where two worker processes are run on a single node—each assigned to one processor with 24 cores and its corresponding, fastest-access memory.

The map initialization stage, both in the worker (Figure 6.8b) and in the server (Figure 6.8a), is sequential. Therefore, increasing the number of cores should not affect the computation time. However, after increasing the

core count from 1 to 2, the duration of this stage decreased significantly. This may be related to the fact that, in each experiment, both server and worker computations were executed on the same node. When running the simulation on a single core, the server and worker threads could not process the map in parallel. This situation changed with the addition of a second core.



(a) Speedup

(b) Efficiency

Figure 6.9: Strong scalability on single node - Performance metric results (according to Amdahl's law)

The achieved speedup is close to linear for 16 or fewer cores. The deviation from the ideal result is mainly due to the size of the thread pool used for executing most parallel tasks. In tests with more than one core, the number of threads in the pool was smaller than the number of available cores, resulting in a pool size of 1 for both the one-core and two-core experiments. For the remaining experiments, the pool size remained smaller than the number of cores. Therefore, the speedup values for these tests could not be ideal since they were calculated relative to the single-core result with the same number of threads.

The impact of the thread pool size can also be seen in the simulator efficiency chart (Figure 6.9b), where the metric value drops significantly after adding the second core. In experiments using more than 16 cores, the speedup curve begins to flatten, suggesting that adding more cores for this problem size will no longer yield performance improvements. However, if the map size and the number of vehicles simulated within a node were increased, a higher number of cores might enable efficient computations.

Summary

Strong scalability tests demonstrated efficient scaling of the simulator for up to 16 cores. Beyond that point, speedup gains diminished, and iteration time stabilized. These results suggest that for the given problem size, using more than 16 threads yields limited performance improvements. Nonetheless, the experiments provided valuable insight into the lower bounds of execution time for each simulation phase and highlighted architectural factors (e.g., NUMA) affecting scalability within a compute node.

6.3. Multi-Node Scalability Evaluation

The multi-node experiments provide a fuller picture of simulator capabilities, incorporating communication costs and the inevitable overheads of distributed execution. These experiments are particularly valuable because they shed light on how the system behaves under conditions of horizontal scaling. Unlike vertical scaling, which is limited by the capacity of a single machine, horizontal scaling enables the problem size to grow to much larger dimensions by distributing the workload across several nodes. This approach not only allows for tackling com-

putational problems of significantly greater complexity but is also generally easier and more cost-effective to implement, making it a practical solution for large-scale simulations.

6.3.1. Weak scalability

One of the primary goals of the scalability evaluation was to assess whether the HiPUTS simulator adheres to the principle of scale invariance—a defining characteristic of super-scalable systems. Scale invariance implies that, as the number of computational units increases, each unit continues to process an approximately constant share of the overall workload. To achieve this, the algorithm must be designed such that as the problem size increases—e.g., due to higher traffic density or larger urban areas—additional computational resources can be added while maintaining a constant workload per node. This scale-invariant behavior ensures that each worker remains fully utilized regardless of problem size.

To examine this behavior, the test design followed the principles of weak scalability, where both the simulated road network and the number of vehicles were scaled in line with the number of processing nodes. This approach not only mirrors realistic system expansion but also avoids the artificial performance degradation often seen when fixed-size problems are distributed across an increasing number of workers.

Experiment Description

In each weak scalability test, every worker node was initially assigned a traffic network segment comprising 65,536 intersections. Depending on the scenario, either approximately 2.18 million or 4.37 million vehicles were simulated within that section, resulting in average inter-vehicle distances of around 60 meters or 30 meters, respectively. As the system scaled, both the network size and total vehicle count grew proportionally. For instance, simulations run with 4 worker nodes expanded the traffic network to 262,144 intersections and over 17 million vehicles in the high-density case. This ensured that each node retained a consistent workload even as overall system complexity increased.

Results

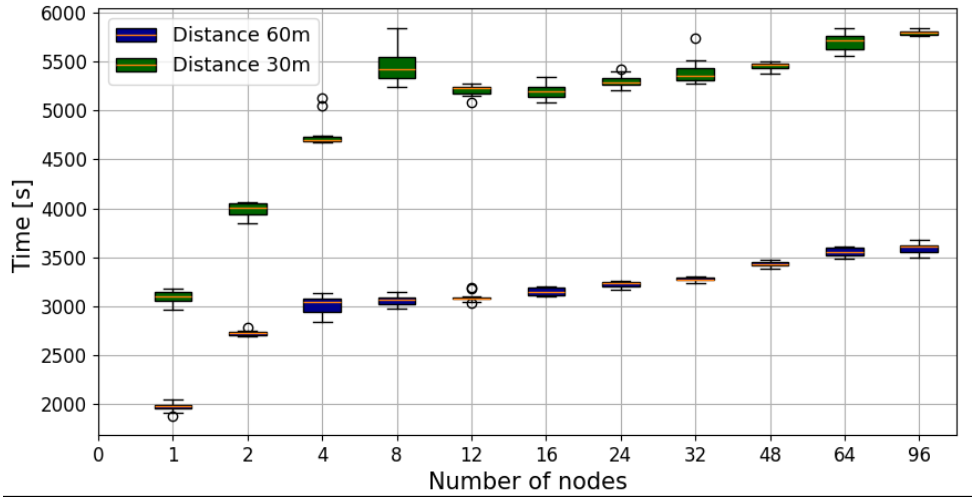


Figure 6.10: Total simulation time depending on used number of nodes

As illustrated in Figure 6.10, total simulation time initially decreases with the addition of nodes but then begins to stabilize, particularly after the use of 12 nodes. Beyond this point, further additions of compute nodes result in only marginal increases in total runtime. The shortest simulation time, observed when using a single node, is unsurprising—it benefits from the absence of inter-node communication, synchronization barriers, and load

balancing overhead. These effects are also visible in Figure 6.11, which shows the breakdown of average time spent per simulation step.

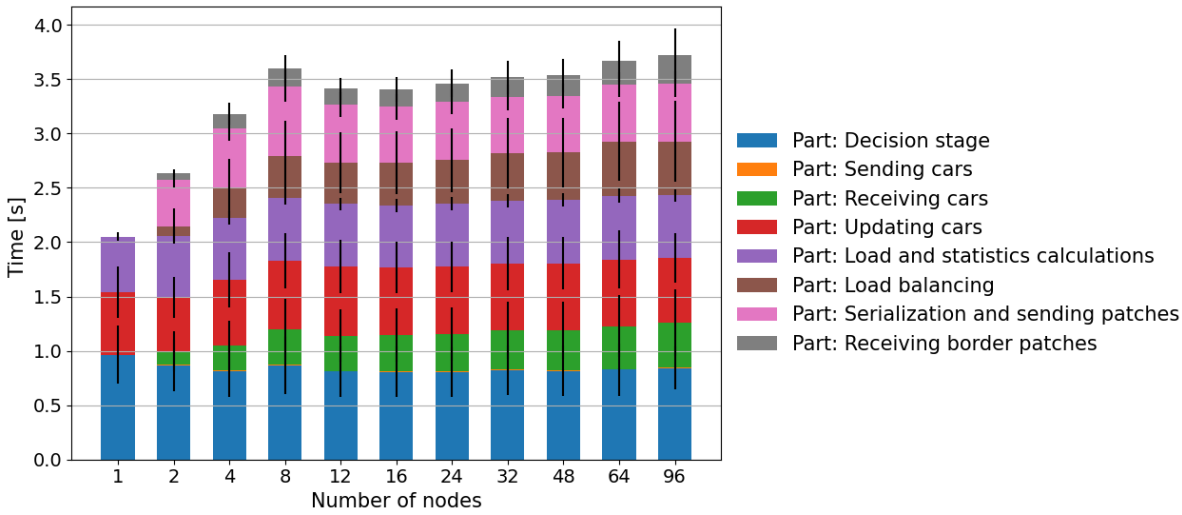


Figure 6.11: Average duration of one simulation step subdivided into stages (average cars distance equal to 30m)

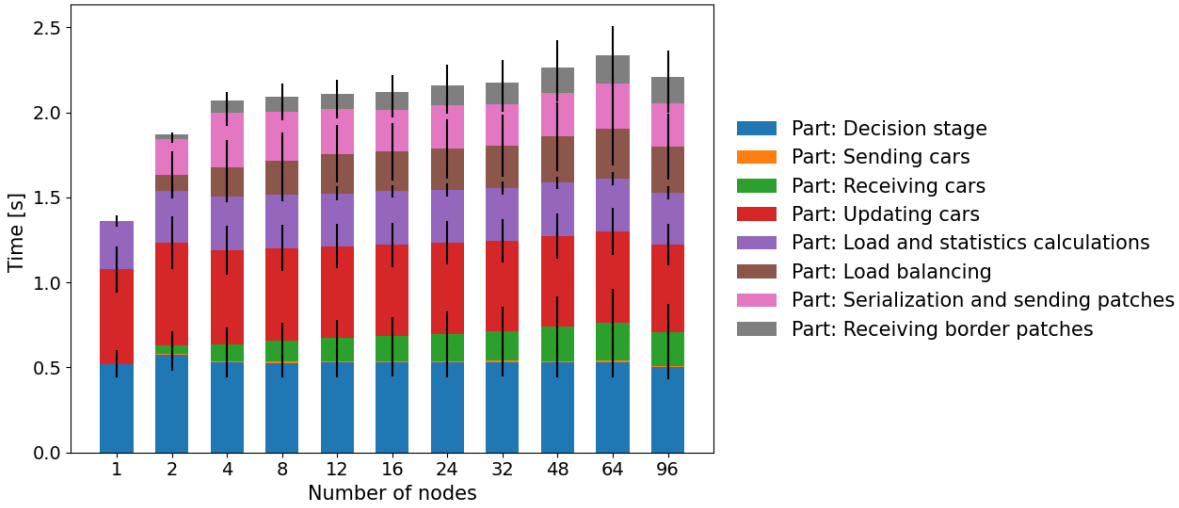
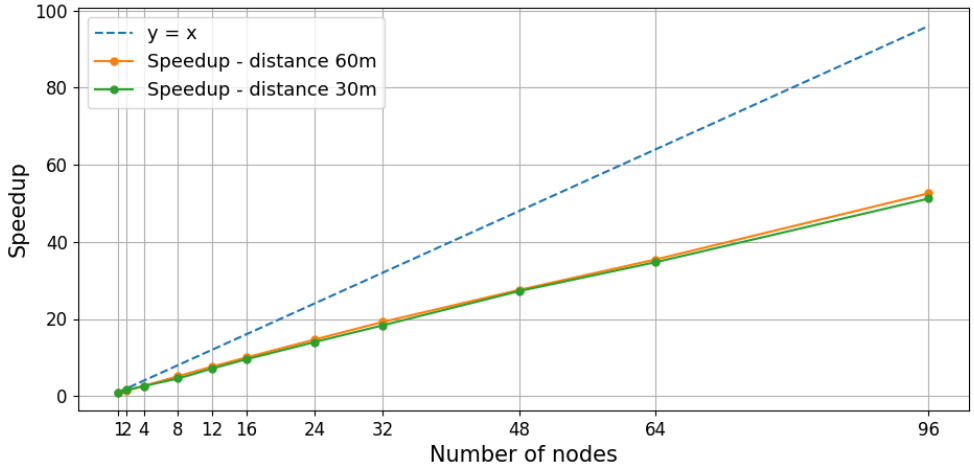


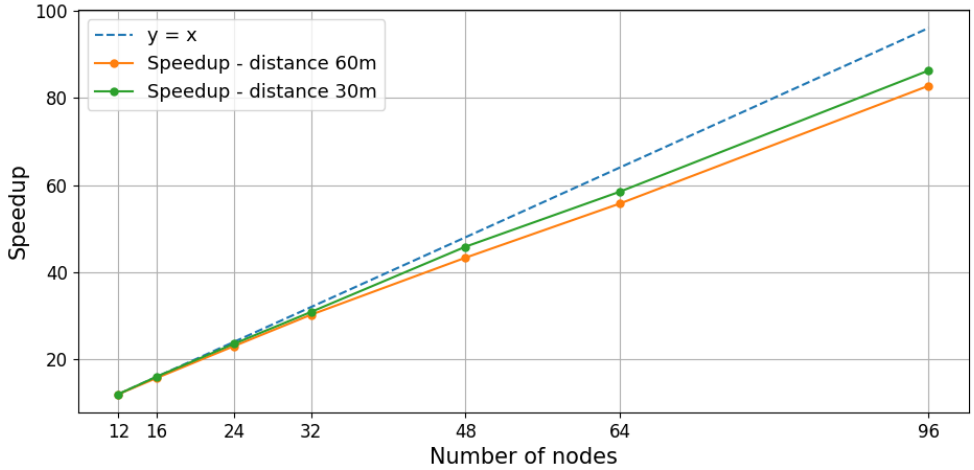
Figure 6.12: Average duration of one simulation step subdivided into stages (average cars distance equal to 60m)

Initially, increasing the number of nodes leads to a rise in communication-related tasks, as each worker must exchange data with more neighbors. However, this overhead levels off once the system reaches a certain scale. Specifically, when 12 or more workers are used, the average number of neighboring workers per node stabilizes at four—thanks to the grid-based partitioning of the toroidal network. As a result, the communication load per worker ceases to grow, and the system starts to operate under the assumptions of scale invariance. From this threshold onward, the simulator's scalability characteristics can be meaningfully evaluated.

Figures 6.11 and 6.12 provides further insight by showing the average time consumed by different stages of a simulation step. From about 8 nodes onward, the duration of each stage remains nearly constant across the system, indicating that HiPUTS maintains consistent per-node performance regardless of total system size. This consistency suggests that additional workers do not negatively affect the computational efficiency of others—precisely the behavior expected of a super-scalable architecture.



(a) Relative to 1 node (increasing communication)



(b) Relative to 12 nodes (fixed communication pattern)

Figure 6.13: HiPUTS simulator weak speedup results

Speedup measurements provide additional evidence of this scalability behavior. Figure 6.13a shows the speedup relative to a single-node execution. While the trend is generally upward, it deviates from ideal linear scaling. This is expected, as the inclusion of communication, synchronization, and load balancing overhead limits the degree of speedup that can be achieved compared to a completely isolated single-node run.

More telling results appear in Figure 6.13b, which recalculates speedup relative to a 12-node baseline—the point at which scale invariance is first maintained. From this perspective, the speedup becomes nearly linear, reflecting the simulator's ability to scale efficiently as both problem size and compute capacity increase in tandem.

Summary

In conclusion, the results strongly indicate that HiPUTS meets the conditions necessary for scale invariance under weak scaling scenarios. The system demonstrates consistent processing time per node and achieves near-linear speedup once a sufficient number of workers is reached. This validates the effectiveness of the simulator's distributed architecture and supports its classification as a super-scalable solution for large-scale traffic simulations.

6.3.2. Strong Scalability

Strong scalability, as articulated by Amdahl's Law, refers to a system's ability to achieve faster execution times as more processing units are added, while the total problem size remains fixed. In this model, the workload per processing unit decreases as the number of units increases. Accordingly, in the context of HiPUTS, both the size of the simulated road network and the number of vehicles handled by each worker are reduced proportionally when more computational nodes are employed.

While strong scalability is a widely used metric in parallel computing, it holds less significance in the context of super-scalability. Super-scalable systems are designed to maintain a constant workload per node even as the system expands, which aligns more closely with weak scalability principles. In strong scaling scenarios, adding more processors without increasing the overall problem size can lead to idle resources, reducing efficiency and ultimately limiting performance improvements. Nonetheless, strong scalability testing remains a critical diagnostic tool, as it helps identify limits imposed by sequential components, communication bottlenecks, and overhead introduced by coordination mechanisms. Understanding these constraints is key to refining distributed algorithms and system design.

Experiment Description

To evaluate HiPUTS under this paradigm on a synthetically generated torus environment, experiments were conducted using a fixed-size traffic simulation consisting of 1,048,576 intersections and either 69,905,072 or 34,952,536 vehicles, depending on the traffic density configuration. These settings provided a stable test environment to observe how performance changed as the number of processing nodes increased. During the experiments on 16 nodes, each worker simulated 4,369,067 vehicles, i.e. the same number that was initially present on each node during the weak scalability tests. For the strong scalability tests, the tri-state load balancing method was used. The experiments simulated 1,500 time steps, of which the last 1,000 steps were used for processing the results.

Results

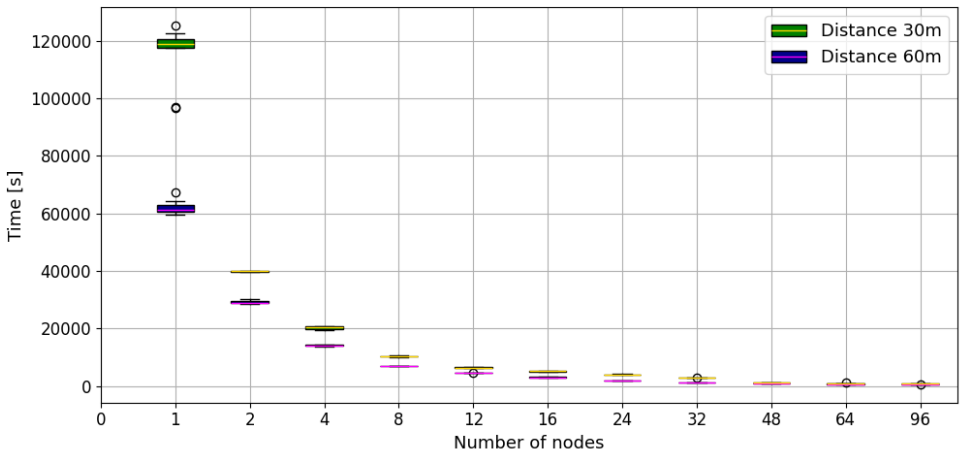


Figure 6.14: Total simulation time depending on the number of nodes used

As demonstrated in Figures 6.14 increasing the number of worker nodes results in a consistent reduction in total simulation time and in the average duration of a simulation step—hallmarks of good strong scalability. Especially the results for both experiments with more vehicles 6.15 and with less vehicles 6.16 also illustrate it. However, the rate of performance gain diminishes beyond approximately 48 nodes, reflecting the typical behavior predicted by

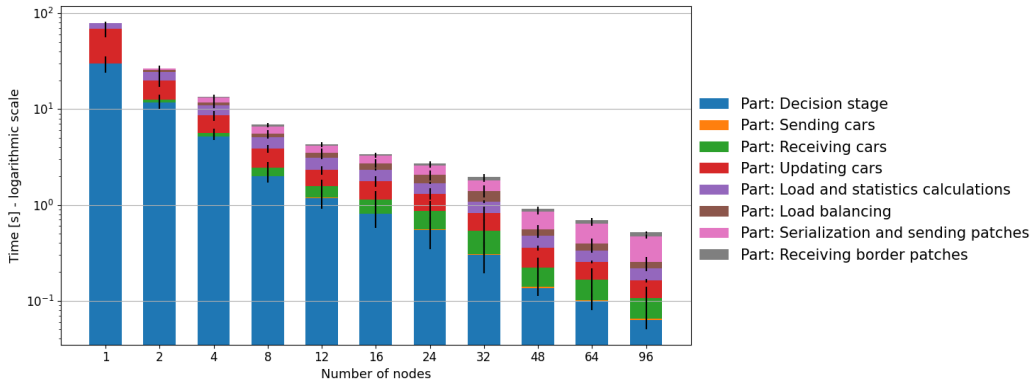


Figure 6.15: Average time per simulation step, broken down into individual stages (30m vehicle spacing)

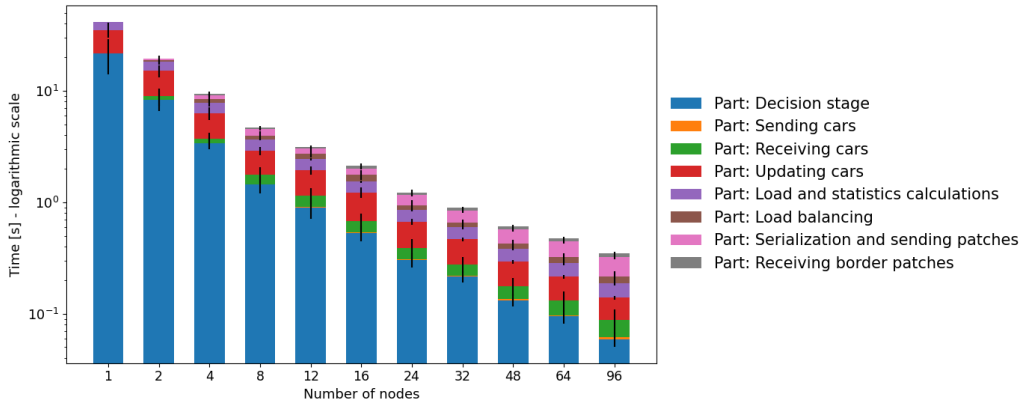


Figure 6.16: Average time per simulation step, broken down into individual stages (60m vehicle spacing)

Amdahl's Law [5]. At larger scales, the impact of non-parallelizable tasks and inter-node communication overheads becomes increasingly pronounced, which naturally limits further gains.

Notably, the most significant performance improvement was observed during the transition from 1 to 2 nodes, especially in the high-density scenario (30-meter spacing between vehicles). In this configuration, the simulator achieved a super-linear speedup, where performance improved more than proportionally to the number of added processors. Even in the lower-density case, the speedup remained close to linear. These outcomes are particularly interesting considering that communication, synchronization, and load-balancing overheads usually increase with the number of participating nodes. However, at low node counts, these overheads are negligible compared to the computational benefits of distributing the workload.

The observed super-linear behavior (Figure 6.17) is likely influenced by architectural efficiencies. When the data processed by each worker fits more comfortably within the processor's cache, the number of cache misses and memory access delays is significantly reduced. This leads to better use of the available computational resources than in single-node executions, where the larger local dataset may exceed cache capacity and force more frequent memory operations.

Further insights are revealed in Figures 6.18 and 6.19, which illustrates the relative speedup of different simulation stages. The most scalable components of the simulation include the decision-making phase, the vehicle parameter update step, and the load/statistics computation. These tasks are purely computational and benefit significantly from parallelism due to their embarrassingly parallel nature—each can be executed independently without requiring synchronization or shared state.

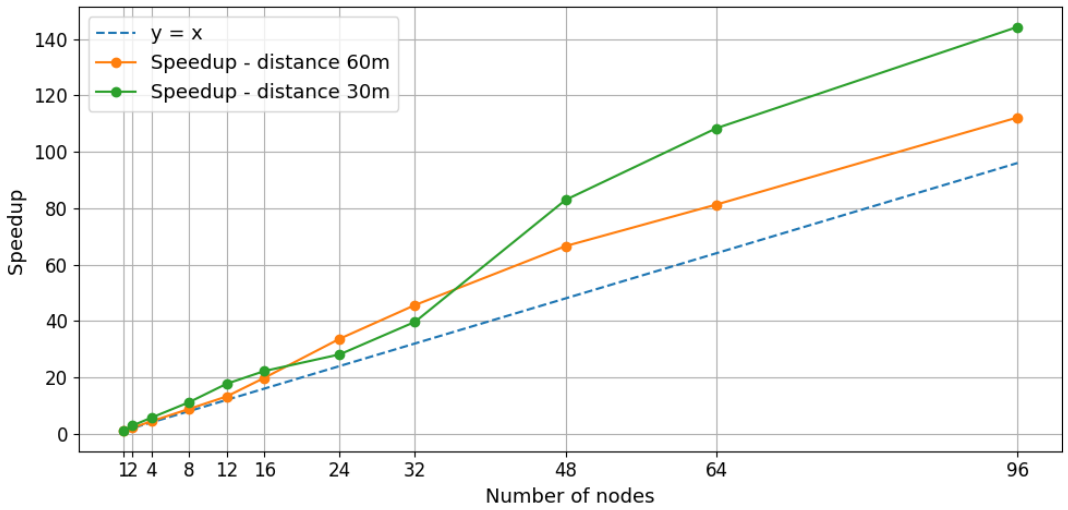


Figure 6.17: Overall speedup under strong scalability tests

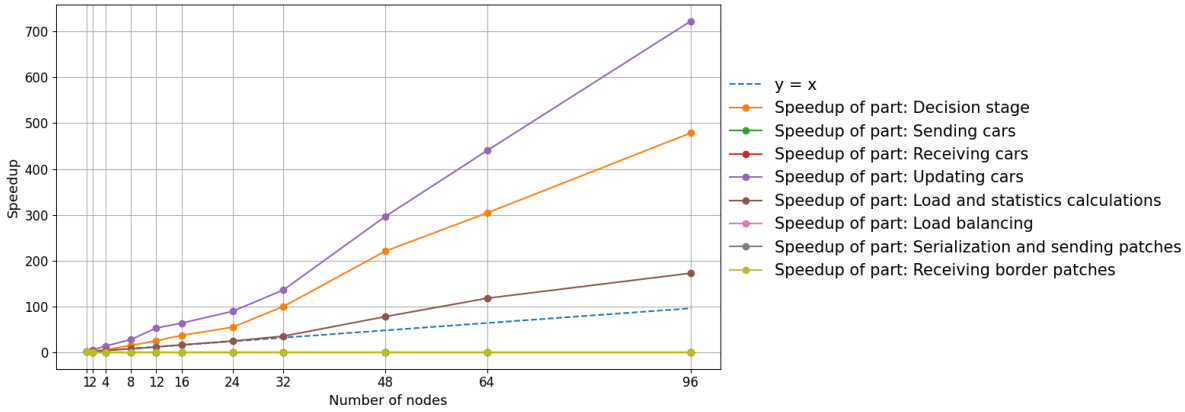


Figure 6.18: Speedup of individual simulation phases (30m average vehicle spacing)

In contrast, stages involving inter-node communication, such as border patch synchronization, show less favorable scaling. These components were absent in the single-node baseline but become more prominent as more workers are introduced. While their runtime contributions are relatively minor at small scales, they grow as the simulation expands and begin to offset some of the gains from parallel execution. In the figures, these communication-intensive stages are mostly hidden behind the "Receiving border patches" line.

A particularly noteworthy observation is the sharp performance gain seen when the simulator was run with more than 32 worker nodes in the 30-meter vehicle spacing test. This improvement likely results from reduced per-node memory pressure, allowing faster data access and lower cache contention. By spreading the workload across more nodes, the simulator achieves better hardware utilization and avoids bottlenecks that would otherwise occur in memory-bound operations on a single node.

Summary

In summary, the HiPUTS simulator demonstrates excellent strong scalability characteristics. While constrained by the fundamental limits described by Amdahl's Law, the simulator's architecture is highly parallel and well-optimized for distributed execution. The results show clear and consistent performance improvements as node count increases—up to the point where communication overheads and sequential components begin to dominate. These findings confirm that HiPUTS is not only capable of scaling effectively in weak scalability scenarios but

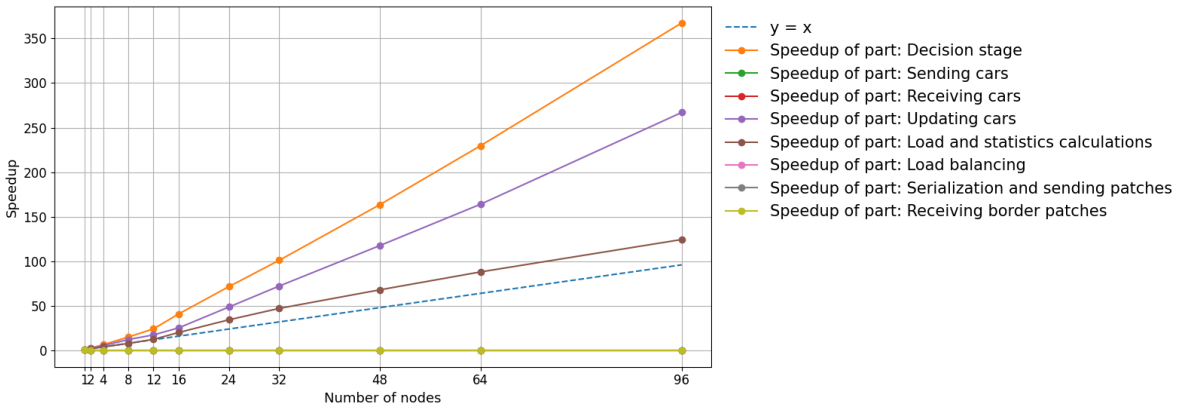


Figure 6.19: Speedup of individual simulation phases (60m average vehicle spacing)

also performs robustly under strong scalability conditions, making it suitable for a wide range of high-performance traffic simulation tasks.

However, despite the obvious negative impact of initialization on performance, access to the entire map in the system is crucial for load balancing and vehicle route generation. In the long run, the cost of initializing the road network should pay off precisely through the load balancing mechanism.

6.4. General Summary

The evaluation presented in this chapter provides a comprehensive validation of the HiPUTS simulator, with particular emphasis on its ability to achieve super-scalability in realistic distributed environments. The results obtained on a single compute node revealed that scalability is strong and consistent up to 24 cores, after which the architecture of the Ares supercomputer (specifically its NUMA memory model) introduced hardware-induced performance limitations. Although this observation underscores the importance of hardware-aware execution, it does not reflect shortcomings of the simulation algorithms themselves. As noted, deploying two worker processes per node—each bound to its own processor and memory banks—would likely alleviate this bottleneck. However, such an approach was outside the scope of the current consideration and remains a direction for future work.

The most important aspects and conclusive findings arise from the multi-node scalability experiments. Here, HiPUTS demonstrated the ability to scale to very large configurations while preserving the principle of scale invariance across many nodes. Under weak scalability conditions, the simulator maintained a nearly constant workload per node, even as the total problem size grew to hundreds of millions of vehicles and thousands of square kilometers of road network. The results show that once the system reached a threshold of approximately twelve nodes, the communication overhead stabilized due to the toroidal grid structure, enabling efficient parallelism without further increase in synchronization costs. From this point onward, the simulator exhibited near-linear speedup relative to the number of nodes, fully validating the architectural design of HiPUTS as a super-scalable system.

Strong scalability tests in multi-node environments further highlighted the robustness of the simulator. Execution time decreased consistently as nodes were added, and in several cases super-linear speedup was observed, particularly when the distribution of workload allowed better utilization of cache hierarchies. These results emphasize that HiPUTS is not only able to exploit parallelism effectively, but also to adapt to the specific characteristics of the underlying hardware to maximize performance. Equally significant is the observation that the most computationally demanding phases of the simulation—such as vehicle decision-making and state updates—scaled almost perfectly across nodes. This validates the design choice of decomposing the simulation into fine-grained, embarrassingly parallel tasks executed within a distributed framework.

7 Conclusions and further work

The research presented in this dissertation has addressed the problem of enabling super-scalable simulations of microscopic urban traffic using continuous models—a domain historically constrained by computational limitations and synchronization bottlenecks. Existing approaches in the field often rely on simplifying assumptions or struggle to maintain consistency and performance as simulation scales increase. The presented research has introduced a new method along with system architecture and algorithmic designs that overcomes these limitations by ensuring distribution transparency, maintaining scale invariance, and enabling robust decentralized control. Through the development of HiPUTS, this work demonstrates that high-fidelity, agent-based simulations can be executed efficiently on thousands of computing cores, without compromising correctness or realism. The results confirm that it is possible to build a system that combines detailed modeling with extreme scalability, thereby contributing both a novel simulation framework and a set of transferable architectural and algorithmic principles.

7.1. Summary of contributions

This dissertation addresses a central and long-standing challenge in computational urban traffic modeling: how to simulate complex, dynamic, real-world road networks using detailed microscopic models at scales that fully exploit modern high-performance computing (HPC) infrastructure. This thesis has explored and validated the feasibility of performing super-scalable, high-fidelity urban traffic simulations using a continuous microscopic model. The central outcome of this research is the description of the method and design, together with the implementation of those concepts and simulation methods resulting in the HiPUTS, a high-performance urban traffic simulator. It involves scale invariance, distribution transparency, and computational efficiency on a supercomputing platform.

HiPUTS introduces a new architectural approach to traffic simulation, based on spatial decomposition into hexagonal patches with local communication boundaries. This structure allows the system to maintain correctness independently of the number of computing nodes involved. Unlike many existing simulators, HiPUTS guarantees consistency between sequential and distributed executions, addressing one of the most critical limitations in the domain.

The simulator employs a decentralized model of execution that eliminates global synchronization barriers, relying instead on peer-based messaging and localized state management. This not only enhances performance but also contributes to the robustness of the system under real-world computing conditions, where heterogeneous node loads and latencies are common.

A further advancement is the inclusion of a dynamic load balancing mechanism that adapts at runtime to shifts in traffic density. By migrating simulation patches between nodes based on observed computational load, the system ensures sustained performance even in highly non-uniform or evolving scenarios.

The scalability and correctness of HiPUTS have been empirically verified through extensive simulations conducted on the Ares supercomputer. These simulations involved synthetic road networks and demonstrated that the system can efficiently handle millions of simulated agents across thousands of computing cores.

Importantly, the architectural concepts proposed here are general enough to extend beyond the traffic simulation domain. The principles of local communication, modular patch-based design, and scale-invariant workload distribution may be applicable in other fields where continuous spatial simulation is required at extreme scale.

The created HiPUTS system demonstrates the possibility of building super-scalable simulations of microscopic continuous urban traffic models. The design of the system allowed achieving scale invariance and distribution transparency in this complex computational problem. The proposed architectural concepts and algorithms can be used in other simulation tools, also outside the field of traffic simulation. These results indicate that the general methodology may have broader applicability in distributed spatial simulations that require high fidelity and performance.

HiPUTS was developed with a modular architecture, enabling seamless addition of new vehicle behaviors, decision models, traffic control algorithms, or visualization tools. The framework supports various extensions, including traffic lights, multi-lane behaviors (IDM+MOBIL), public transport elements, and real-world map data import. This makes it not only a research prototype but also a practical tool ready for adoption in larger simulation-based studies, integration with AI-driven traffic systems, and educational use.

7.2. Further research directions

While the current work lays a strong foundation, several promising research directions remain. While preliminary validation has already been carried out using real-world map data from selected cities, a natural next step is to perform systematic validation against high-resolution empirical traffic datasets (e.g., loop detectors, floating car data, or city sensor networks). Such studies would not only confirm the accuracy of simulated dynamics but also support calibration of behavioral models for specific urban contexts. Integrating detailed vehicle behavior models, public transport systems, and pedestrian flows could significantly broaden the scope of its application.

Another opportunity lies in adapting HiPUTS for real-time simulation. This would enable its use in live traffic monitoring and decision-support systems for urban traffic control. Real-time responsiveness would also support integration with autonomous vehicle systems and digital twin platforms for cities.

There is also scope for enriching the simulator with adaptive or intelligent agents, potentially powered by machine learning models. These agents could simulate realistic driver behavior or optimize traffic control strategies through reinforcement learning, offering a testbed for AI-driven transportation solutions.

Finally, extending HiPUTS to leverage heterogeneous computing architectures such as GPUs or emerging exascale systems could further improve its speed and scale. Coupled with fault-tolerant mechanisms, this would enable long-running, resilient simulations suitable for use in critical infrastructure planning and risk analysis.

In conclusion, this thesis has demonstrated that it is possible to construct a scalable, high-fidelity simulator for urban traffic that is both technically robust and practically useful. HiPUTS not only advances the capabilities of traffic simulation but also opens new paths for research in high-performance spatial simulation more broadly.

List of Figures

List of Tables

List of Equations

Bibliography

[1] M. Ahmed, R. Seraj, and S. Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8):1295, 2020.

[2] M. S. Ahmed and M. A. Hoque. Partitioning of urban transportation networks utilizing real-world traffic parameters for distributed simulation in sumo. In *2016 IEEE Vehicular Networking Conference (VNC)*, pages 1–4. IEEE, 2016.

[3] S. Ahn and M. J. Cassidy. Freeway traffic oscillations and vehicle lane-change maneuvers. *Transportation and traffic theory*, 1:691–710, 2007.

[4] S. Albeaik, A. Bayen, M. T. Chiri, X. Gong, A. Hayat, N. Kardous, A. Keimer, S. T. McQuade, B. Piccoli, and Y. You. Limitations and improvements of the intelligent driver model (idm). *SIAM Journal on Applied Dynamical Systems*, 21(3):1862–1892, 2022.

[5] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[6] K. J. Astrom, T. Hagglund, and P. Controllers. Theory, design and tuning. *Research Triangle Park: 2nd Ed. Instrumentation, Systems and Automatic Society*, 1995.

[7] M. Bando, K. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama. Dynamical model of traffic congestion and numerical simulation. *Physical review E*, 51(2):1035, 1995.

[8] J. Barceló and J. Casas. Dynamic network simulation with aimsun. In *Simulation approaches in transportation analysis: Recent advances and challenges*, pages 57–98. Springer, 2005.

[9] J. Barceló et al. *Fundamentals of traffic simulation*, volume 145. Springer, 2010.

[10] F. F. Bastarianto, T. O. Hancock, C. F. Choudhury, and E. Manley. Agent-based models in urban transportation: review, challenges, and opportunities. *European Transport Research Review*, 15(1):19, 2023.

[11] E. R. Boer. Car following from the driver's perspective. *Transportation Research Part F: Traffic Psychology and Behaviour*, 2(4):201–206, 1999.

[12] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203, 2000.

[13] M. Boulle. Compact mathematical formulation for graph partitioning. *Optimization and Engineering*, 5:315–333, 2004.

[14] M. Brackstone and M. McDonald. Car-following: a historical review. *Transportation Research Part F: Traffic Psychology and Behaviour*, 2(4):181–196, 1999.

[15] Q. Bragard, A. Ventresque, L. Murphy, et al. dsumo: towards a distributed sumo. 2013.

[16] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg. Optimizing traffic lights in a cellular automaton model for city traffic. *Physical review E*, 64(5):056132, 2001.

[17] W. Burghout, H. N. Koutsopoulos, and I. Andreasson. Hybrid mesoscopic–microscopic traffic simulation. *Transportation Research Record*, 1934(1):218–225, 2005.

[18] G. O. Burnham and G. A. Bekey. A heuristic finite-state model of the human driver in a car-following situation. *IEEE Transactions on Systems, Man, and Cybernetics*, (8):554–562, 1976.

[19] A. Buss and A. Al Rowaei. A comparison of the accuracy of discrete event and discrete time. In *Proceedings of the 2010 Winter Simulation Conference*, pages 1468–1477. IEEE, 2010.

[20] G. D. Cameron and G. I. Duncan. Paramics—parallel microscopic simulation of road traffic. *The Journal of Supercomputing*, 10:25–53, 1996.

[21] A. R. Campos, J. L. Davis, W. L. S. Costa, and B. S. Soares Filho. Spatially explicit agent based model of rabbit population.

[22] A. Ceder and A. D. May. Further evaluation of single-and two-regime traffic flow models. *Transportation Research Record*, (567), 1976.

[23] I. Cenamor, L. Chrpa, F. Jimoh, T. L. McCluskey, and M. Vallati. Planning & scheduling applications in urban traffic management. 2014.

[24] S. De Marchi and S. E. Page. Agent-based models. *Annual Review of political science*, 17:1–20, 2014.

[25] T. Dijker, P. H. Bovy, and R. G. Vermijs. Car-following under congested conditions: empirical findings. *Transportation Research Record*, 1644(1):20–28, 1998.

[26] C. Dobler. Implementation of a time step based parallel queue simulation in matsim. In *10th Swiss Transport Research Conference. Monte Verita, Ascona*, 2010.

[27] Q. Du, V. Faber, and M. Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM review*, 41(4):637–676, 1999.

[28] C. Engelmann and A. Geist. Super-scalable algorithms for computing on 100,000 processors. In *International Conference on Computational Science*, pages 313–321. Springer, 2005.

[29] L. Evans. Perceptual thresholds in car-following. *Transportation Science*, 11(1):60–72, 1977.

[30] L. Evans and R. Rothery. Experimental measurements of perceptual thresholds in car-following, highway res. *Record*, 464:13, 1973.

[31] M. Fellendorf and P. Vortisch. Microscopic traffic flow simulator vissim. *Fundamentals of traffic simulation*, pages 63–93, 2010.

[32] R. Fernández. Modelling public transport stops by microscopic simulation. *Transportation Research Part C: Emerging Technologies*, 18(6):856–868, 2010.

[33] J. L. Ferrer and J. Barceló. Aimsun2: advanced interactive microscopic simulator for urban and non-urban networks. *Research Report*, 1993.

[34] U. Fhwa. Traffic network analysis with netsim: a user guide. Technical report, Report FHWA-IP-80-3. Department of Transportation, 1980.

[35] F. Fleissner and P. Eberhard. Load balanced parallel simulation of particle-fluid dem-sph systems with moving boundaries. *Advances in Parallel Computing*, 15:37–44, 2008.

[36] G. D. for National Polish Roads and Motorways. Current road statistics. https://www.gov.pl/web/gddkia/aktualna-statystyka-drogowa2. Published: 2022-05-16, Accessed: 2025-05-01.

[37] H.-T. Fritzsche and D.-b. Ag. A model for traffic simulation. *Traffic Engineering+ Control*, 35(5):317–21, 1994.

[38] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[39] D. C. Gazis, R. Herman, and R. B. Potts. Car-following theory of steady-state traffic flow. *Operations research*, 7(4):499–505, 1959.

[40] D. C. Gazis, R. Herman, and R. W. Rothery. Nonlinear follow-the-leader models of traffic flow. *Operations research*, 9(4):545–567, 1961.

[41] K. Germaschewski, W. Fox, S. Abbott, N. Ahmadi, K. Maynard, L. Wang, H. Ruhl, and A. Bhattacharjee. The plasma simulation code: A modern particle-in-cell code with patch-based load-balancing. *Journal of Computational Physics*, 318:305–326, 2016.

[42] A. Giordano, A. De Rango, R. Rongo, D. D'Ambrosio, and W. Spataro. Dynamic load balancing in parallel execution of cellular automata. *IEEE Transactions on Parallel and Distributed Systems*, 32(2):470–484, 2020.

[43] P. G. Gipps. A behavioural car-following model for computer simulation. *Transportation research part B: methodological*, 15(2):105–111, 1981.

[44] H. Gong, H. Liu, and B.-H. Wang. An asymmetric full velocity difference car-following model. *Physica A: Statistical Mechanics and its Applications*, 387(11):2595–2602, 2008.

[45] J. L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[46] T. Hägglund and J. L. Guzmán. Give us pid controllers and we can control the world. *IFAC-PapersOnLine*, 58(7):103–108, 2024.

[47] A. Halati, H. Lieu, and S. Walker. Corsim-corridor traffic simulation model. In *Traffic Congestion and Traffic Safety in the 21st Century: Challenges, Innovations, and OpportunitiesUrban Transportation Division, ASCE; Highway Division, ASCE; Federal Highway Administration, USDOT; and National Highway Traffic Safety Administration, USDOT.*, 1997.

[48] S. H. Hamdar and H. S. Mahmassani. From existing accident-free car-following models to colliding vehicles: exploration and assessment. *Transportation research record*, 2088(1):45–56, 2008.

[49] A. Hanken and T. Rockwell. A model of car following derived empirically by piece-wise regression analysis. in vehicular traffic science. In *Proceedings of the Third International Symposium on the Theory of Traffic FlowOperations Research Society of America*, 1967.

[50] D. Helbing and M. Schreckenberg. Cellular automata simulating experimental properties of traffic flow. *Physical review E*, 59(3):R2505, 1999.

[51] W. Helly. Simulation of bottlenecks in single-lane traffic flow. 1959.

[52] R. Herman and I. Prigogine. A two-fluid approach to town traffic. *Science*, 204(4389):148–151, 1979.

[53] B. Higgs, M. Abbas, and A. Medina. Analysis of the wiedemann car following model over different speeds using naturalistic data. In *Procedia of RSS Conference*, pages 1–22, 2011.

[54] J. H. Holland and J. H. Miller. Artificial adaptive agents in economic theory. *The American economic review*, 81(2):365–370, 1991.

[55] S. P. Hoogendoorn and P. H. Bovy. State-of-the-art of vehicular traffic flow modelling. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 215(4):283–303, 2001.

[56] A. Horni, K. Nagel, and K. W. Axhausen. Introducing matsim. In *The multi-agent transport simulation MATSim*, pages 3–7. Ubiquity Press, 2016.

[57] M.-B. Hu, R. Jiang, Y.-H. Wu, W.-X. Wang, and Q.-S. Wu. Urban traffic from the perspective of dual graph. *The European Physical Journal B*, 63(1):127–133, 2008.

[58] Y. Imai, H. Fujii, K. Okano, M. Matsudaira, and T. Suzuki. Development of dynamic micro-and macroscopic hybrid model for efficient highway traffic simulation: Model extension to merging sections and validation with probe data. *International Journal of Intelligent Transportation Systems Research*, 22(1):159–170, 2024.

[59] P. Jagnere and A. Bansal. Road traffic simulation-a discrete event driven model. In *International Conference on Reliability, Infocom Technologies and Optimization (ICRITO-2013)*, 2013.

[60] M. Janczykowski, W. Turek, M. Malawski, and A. Byrski. Large-scale urban traffic simulation with scala and high-performance computing system. *Journal of computational science*, 35:91–101, 2019.

[61] R. Jiang, Q. Wu, and Z. Zhu. Full velocity difference model for a car-following theory. *Physical Review E*, 64(1):017101, 2001.

[62] S. Jin, D.-H. Wang, Z.-Y. Huang, and P.-F. Tao. Visual angle model for car-following theory. *Physica A: Statistical Mechanics and its Applications*, 390(11):1931–1940, 2011.

[63] H. Kanezashi and T. Suzumura. Performance optimization for agent-based traffic simulation by dynamic agent assignment. In *Proc. of 2015 Winter Simulation Conference*, WSC '15, pages 757–766, Piscataway, NJ, USA, 2015. IEEE Press.

[64] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *ICPP (3)*, pages 113–122, 1995.

[65] G. Karypis and V. Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1997.

[66] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[67] A. Kesting, M. Treiber, and D. Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[68] E. B. Khunayn, S. Karunasekera, H. Xie, and K. Ramamohanarao. Straggler mitigation for distributed behavioral simulation. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2638–2641, 2017.

[69] S. Kikuchi and P. Chakroborty. Car-following model based on fuzzy inference system. *Transportation Research Record*, pages 82–82, 1992.

[70] R. Klefstad, Y. Zhang, M. Lai, R. Jayakrishnan, and R. Lavanya. A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, pages 813–818. IEEE, 2005.

[71] W. Knospe, L. Santen, A. Schadschneider, and M. Schreckenberg. Arealistic two-lane traffic model for highway traffic. *Journal of Physics A: Mathematical and General*, 35(15):3369, 2002.

[72] K. Kollman, J. H. Miller, and S. E. Page. Adaptive parties in spatial elections. *American Political Science Review*, 86(4):929–937, 1992.

[73] E. Kometani. Dynamic behavior of traffic with a nonlinear spacing-speed relationship. *Theory of Traffic Flow (Proc. of Sym. on TTF (GM))*, pages 105–119, 1959.

[74] H. N. Koutsopoulos and H. Farah. Latent class model for car following behavior. *Transportation research part B: methodological*, 46(5):563–578, 2012.

[75] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.

[76] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[77] H. Lenz, C. Wagner, and R. Sollacher. Multi-anticipative car-following model. *The European Physical Journal B-Condensed Matter and Complex Systems*, 7:331–335, 1999.

[78] W. Leutzbach. *Introduction to the theory of traffic flow*, volume 47. Springer, 1988.

[79] W. Leutzbach and R. Wiedemann. Development and applications of traffic simulation models at the karlsruhe institut für verkehrswesen. *Traffic engineering & control*, 27(5):270–278, 1986.

[80] H. Li, C. Roncoli, and Y. Ju. A helly model-based mpc control system for jam-absorption driving strategy against traffic waves in mixed traffic. *Applied Sciences*, 14(4):1424, 2024.

[81] M. J. Lighthill and G. B. Whitham. On kinematic waves ii. a theory of traffic flow on long crowded roads. *Proceedings of the royal society of london. series a. mathematical and physical sciences*, 229(1178):317–345, 1955.

[82] J. Long, Z. Gao, X. Zhao, A. Lian, and P. Orenstein. Urban traffic jam simulation based on the cell transmission model. *Networks and Spatial Economics*, 11:43–64, 2011.

[83] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[84] H. S. Mahmassani and K. F. Abdelghany. Dynasmart-ip: Dynamic traffic assignment meso-simulator for intermodal networks. In *Advanced modeling for transit operations and service planning*, pages 200–229. Emerald Group Publishing Limited, 2002.

[85] H. S. Mahmassani, R. Jayakrishnan, and R. Herman. Network traffic flow theory: Microscopic simulation experiments on supercomputers. *Transportation Research Part A: General*, 24(2):149–162, 1990.

[86] S. Mammar, S. Mammar, and J.-P. Lebacque. Highway traffic hybrid macro-micro simulation model. *IFAC Proceedings Volumes*, 39(12):627–632, 2006.

[87] P. Manser, H. Becker, S. Hörl, and K. W. Axhausen. Designing a large-scale public transport network using agent-based microsimulation. *Transportation Research Part A: Policy and Practice*, 137:1–15, 2020.

[88] R. Mardiati, N. Ismail, and A. Faroqi. Review of microscopic model for traffic flow. *ARPN Journal of Engineering and Applied sciences*, 9(10):1794–1800, 2014.

[89] A. Meshkat and J. Vrancken. Multi-objective road network partitioning. *Procedia-Social and Behavioral Science*, 1, 2014.

[90] B. Monien and S. Schamberger. Graph partitioning with the party library: Helpful-sets in practice. In *16th Symposium on Computer Architecture and High Performance Computing*, pages 198–205. IEEE, 2004.

[91] S. Moridpour, M. Sarvi, and G. Rose. Lane changing models: a critical review. *Transportation letters*, 2(3):157–173, 2010.

[92] K. Nagel and M. Rickert. Parallel implementation of the transims micro-simulation. *Parallel Computing*, 27(12):1611–1639, 2001.

[93] K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20(1):125–146, 1994.

[94] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de physique I*, 2(12):2221–2229, 1992.

[95] M. Najdek, N. Brzozowska, and W. Turek. HiPUTS: Super-Scalable Simulation of Microscopic Continuous Urban Traffic Model. In M. Scarpa, S. Cavalieri, S. Serrano, and F. D. Vita, editors, *ECMS 2025, Proceedings*. European Council for Modelling and Simulation, 2025.

[96] M. Najdek, M. Paciorek, W. Turek, and A. Byrski. Three new design patterns for scalable agent-based computing and simulation. *Informatica*, 35(2):379–400, 2024.

[97] M. Najdek, H. Xie, and W. Turek. Scaling simulation of continuous urban traffic model for high performance computing system. In *International Conference on Computational Science*, pages 256–263. Springer, 2021.

[98] G. F. Newell. Nonlinear effects in the dynamics of car following. *Operations research*, 9(2):209–229, 1961.

[99] H. S. Nwana. Software agents: An overview. *The knowledge engineering review*, 11(3):205–244, 1996.

[100] S. E. Olson, A. J. Christlieb, and F. K. Fatemi. Pid feedback for load-balanced parallel gridless dsmc. *Computer Physics Communications*, 181(12):2063–2071, 2010.

[101] E. A. O'Cearbhaill and M. O'Mahony. Parallel implementation of a transportation network model. *Journal of Parallel and Distributed Computing*, 65(1):1–14, 2005.

[102] M. Paciorek, A. Bogacz, and W. Turek. Scalable signal-based simulation of autonomous beings in complex environments. In *International Conference on Computational Science*, pages 144–157. Springer, 2020.

[103] M. Paciorek, J. Bujas, D. Dworak, W. Turek, and A. Byrski. Validation of signal propagation modeling for highly scalable simulations. *Concurrency and Computation: Practice and Experience*, 33(14):e5718, 2021.

[104] M. Paciorek, D. Poklewski-Koziełł, K. Racoń-Leja, A. Byrski, M. Gyurkovich, and W. Turek. Microscopic simulation of pedestrian traffic in urban environment under epidemic conditions. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 69(4):e137725, 2021.

[105] G. Peng and D. Sun. A dynamical model of car-following with the consideration of the multiple information of preceding cars. *Physics Letters A*, 374(15-16):1694–1698, 2010.

[106] A. Prabu, N. Ravi, and L. Li. A novel method for lane-change maneuver in urban driving using predictive markov decision process. *arXiv preprint arXiv:2212.12008*, 2022.

[107] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 259–269. Springer, 1999.

[108] Y. Qin, H. Wang, and D. Ni. Lighthill-whitham-richards model for traffic flow mixed with cooperative adaptive cruise control vehicles. *Transportation Science*, 55(4):883–907, 2021.

[109] S. F. Railsback and V. Grimm. *Agent-based and individual-based modeling: a practical introduction*. Princeton university press, 2019.

[110] J. Raimbault, A. Banos, and R. Doursat. A hybrid network/grid model of urban morphogenesis and optimization. *arXiv preprint arXiv:1612.08552*, 2016.

[111] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn. Smarts: Scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–22, 2016.

[112] P. Renc, M. Bielech, T. Pęcak, P. Morawiecki, M. Paciorek, W. Turek, A. Byrski, and J. Wąs. HPC large-scale pedestrian simulation based on proxemics rules. In *International Conference on Parallel Processing and Applied Mathematics*, pages 489–499. Springer, 2019.

[113] P. I. Richards. Shock waves on the highway. *Operations research*, 4(1):42–51, 1956.

[114] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. *Physica A: Statistical Mechanics and its Applications*, 231(4):534–550, 1996.

[115] S. Ristov, R. Prodan, M. Gusev, and K. Skala. Superlinear speedup in hpc systems: why and when? In *2016 federated conference on computer science and information systems (fedcsis)*, pages 889–898. IEEE, 2016.

[116] J. Ruiz-Rosero, G. Ramirez-Gonzalez, and R. Khanna. Masivo: Parallel simulation model based on opencl for massive public transportation systems' routes. *Electronics*, 8(12):1501, 2019.

[117] A. Schadschneider. Cellular automaton approach to pedestrian dynamics-theory. *arXiv preprint cond-mat/0112117*, 2001.

[118] S. Schamberger. Improvements to the helpful-set algorithm and a new evaluation scheme for graph-partitioners. In *International Conference on Computational Science and its Applications*, pages 49–53. Springer, 2003.

[119] J. L. Schiff. *Cellular automata: a discrete view of the world*. John Wiley & Sons, 2011.

[120] M. Schmidt, C. Wissing, T. Nattermann, and T. Bertram. A probabilistic model for discretionary lane change proposals in highway driving situations. *Forschung Im Ingenieurwesen*, 85(2):485–500, 2021.

[121] R. Sear. Plot of average velocity v as a function of density of cars ρ in the nagel–schreckenberg model. Wikimedia Commons, July 11 2011.

[122] Y. Shoham. Agent oriented programming. Technical report STAN-CS-90-1335. *Computer Science Department, Stanford University*, 1990.

[123] A. Skarbek-Żabkin and M. Szczepanek. Autonomous vehicles and their impact on road infrastructure and user safety. In *2018 XI International Science-Technical Conference Automotive Safety*, pages 1–4. IEEE, 2018.

[124] L. Soule and A. Gupta. An evaluation of the chandy-misra-bryant algorithm for digital logic simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 1(4):308–347, 1991.

[125] R. Tahmasbi and S. M. Hashemi. Modeling and forecasting the urban volume using stochastic differential equations. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):250–259, 2013.

[126] A. Talebpour, H. S. Mahmassani, and S. H. Hamdar. Multiregime sequential risk-taking model of car-following behavior: specification, calibration, and sensitivity analysis. *Transportation research record*, 2260(1):60–66, 2011.

[127] E. Thonhofer, T. Palau, A. Kuhn, S. Jakubek, and M. Kozek. Macroscopic traffic model for large scale urban traffic network design. *Simulation Modelling Practice and Theory*, 80:32–49, 2018.

[128] T. Toledo. Driving behaviors: models and research directions. *Transport Reviews*, 27(1):65–84, 2007.

[129] L. Toscano, G. D'Angelo, and M. Marzolla. Parallel discrete event simulation with erlang. In *Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing*, pages 83–92, 2012.

[130] P. Tranouez, P. Langlois, and É. Daudé. A multiagent urban traffic simulation part i: dealing with the ordinary. *arXiv preprint arXiv:0909.1021*, 2009.

[131] M. Treiber and D. Helbing. Memory effects in microscopic traffic models and wide scattering in flow-density data. *Physical Review E*, 68(4):046119, 2003.

[132] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805, 2000.

[133] M. Treiber, A. Hennecke, and D. Helbing. Microscopic simulation of congested traffic. In *Traffic and Granular Flow'99: Social, Traffic, and Granular Dynamics*, pages 365–376. Springer, 2000.

[134] M. Treiber and A. Kesting. Traffic flow dynamics. *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, 227:228, 2013.

[135] W. Turek. Erlang-based desynchronized urban traffic simulation for high-performance computing systems. *Future Generation Computer Systems*, 79:645–652, 2018.

[136] W. Turek, L. Siwik, and A. Byrski. Leveraging rapid simulation and analysis of large urban road systems on hpc. *Transportation Research Part C: Emerging Technologies*, 87:46–57, 2018.

[137] N. Uthpala, N. Hansika, S. Dissanayaka, K. Tennakoon, S. Dharmarathne, R. Vidanarachchi, J. Alawatugoda, and D. Herath. Analyzing transportation mode interactions using agent-based models. *SN Applied Sciences*, 5(12):357, 2023.

[138] F. van Wageningen-Kessels, H. Van Lint, K. Vuik, and S. Hoogendoorn. Genealogy of traffic flow models. *EURO Journal on Transportation and Logistics*, 4(4):445–473, 2015.

[139] A. Ventresque, Q. Bragard, E. S. Liu, D. Nowak, L. Murphy, G. Theodoropoulos, and Q. Liu. 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*, pages 202–209. IEEE, 2012.

[140] P. Vissim. Ptv vissim. *Karlsruhe, Germany*, 2022.

[141] J. Von Neumann. The general and logical theory of automata. In *Systems research for behavioral science*, pages 97–107. Routledge, 2017.

[142] T. P. E. Vranken and M. Schreckenberg. Cellular automata intersection model. *Collective Dynamics*, 5:1–25, 2020.

[143] C. Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In *Hybrid Metaheuristics: An Emerging Approach to Optimization*, pages 261–289. Springer, 2008.

[144] D.-h. Wang, L.-j. Zhou, and W.-q. Li. Modeling lane changing behavior using fuzzy logic. In *International Conference on Transportation Engineering 2007*, pages 1897–1902, 2007.

[145] D. Wei, F. Chen, and X. Sun. An improved road network partition algorithm for parallel microscopic traffic simulation. In *2010 international conference on mechanic automation and control engineering*, pages 2777–2782. IEEE, 2010.

[146] R. Wiedemann. Simulation des straßenverkehrsflusses. 1974.

[147] M. Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[148] J. Wu, M. Brackstone, and M. McDonald. Fuzzy sets and systems for a motorway microscopic simulation model. *Fuzzy sets and systems*, 116(1):65–76, 2000.

[149] Y. Xu, W. Cai, H. Aydt, M. Lees, and D. Zehe. An asynchronous synchronization strategy for parallel large-scale agent-based traffic simulations. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 259–269, 2015.

[150] Y. Xu, W. Cai, D. Eckhoff, S. Nair, and A. Knoll. A graph partitioning algorithm for parallel agent-based road traffic simulation. In *Proceedings of the 2017 ACM SIGSIM conference on principles of advanced discrete simulation*, pages 209–219, 2017.

[151] D. Yang, L.-l. Zhu, D. Yu, F. Yang, and Y. Pu. An enhanced safe distance car-following model. *Journal of Shanghai Jiaotong University (Science)*, 19:115–122, 2014.

[152] J. Zhang, X. Wang, J. Wang, and J. Wang. Decision-making model of lane-change behavior based on integrated cognitive vehicle cluster situations. In *International conference on green intelligent transportation system and safety*, pages 77–94. Springer, 2016.

[153] Y. Zhang, Q. Xu, J. Wang, K. Wu, Z. Zheng, and K. Lu. A learning-based discretionary lane-change decision-making model with driving style awareness. *IEEE transactions on intelligent transportation systems*, 24(1):68–78, 2022.

[154] Y. Zhang, Y. Zou, Y. Xie, and L. Chen. Modeling mandatory and discretionary lane changes using dynamic interaction networks. *arXiv preprint arXiv:2207.12793*, 2022.

[155] H.-T. Zhao, S. Yang, and X.-X. Chen. Cellular automata model for urban road traffic flow considering pedestrian crossing street. *Physica A: Statistical Mechanics and its Applications*, 462:1301–1313, 2016.

[156] Z. Zheng. Recent developments and research needs in modeling lane changing. *Transportation research part B: methodological*, 60:16–32, 2014.

[157] J. Ziarko, M. Najdek, and W. Turek. Partitioning of complex discrete models for highly scalable simulations. In *Journal of Automation, Mobile Robotics and Intelligent Systems*, IN PRESS.

[158] M. Zych, M. Najdek, M. Paciorek, and W. Turek. Distributed architecture for highly scalable urban traffic simulation. In *International Conference on Computational Science*, pages 517–530. Springer, 2022.