HiPUTS: Super-Scalable Simulation of Microscopic Continuous Urban Traffic Model

Mateusz Najdek[1], Natalia Brzozowska[2], and Wojciech Turek[3]

[1]Institute of Computer Science, AGH University of Krakow, e-mail: najdek@agh.edu.pl
[2]Institute of Computer Science, AGH University of Krakow, e-mail: nbrzoza@student.agh.edu.pl
[3]Institute of Computer Science, AGH University of Krakow, e-mail: wojciech.turek@agh.edu.pl

KEYWORDS

HPC, Simulation, Urban traffic, Scalability, Multi-agent systems

ABSTRACT

The presented HiPUTS system is a new urban traffic simulator, which aims at providing (almost) linear scalability of the distributed simulation computation up to several thousand computing cores of a modern supercomputer. The simulation system supports microscopic, continuous traffic models with fully transparent distribution of computations which does not affect simulation results. In this paper we present the architecture and crucial algorithms of the simulation system and evaluate its performance and scalability. The synthetic scalability evaluation includes both strong and weak scaling experiments. The paper is summarized with a set of design guidelines, which are important for obtaining super-scalability of distributed spatial simulations.

1 Introduction

Urban traffic is one of the most problematic aspects of the functioning of modern cities. Despite over a hundred years of motorization development and adaptation of road infrastructure in cities to its needs, the inhabitants of the vast majority of cities and towns experience problems with the smoothness of movement in road networks, even those newly designed. Interestingly, humanity is able to manage traffic in other types of networks, like railway and telecommunications networks. The autonomy and diversity of urban traffic users, the legacy of traffic managing methods and the growing size of road networks still pose a very important and interesting scientific challenge.

One of the key methods of studying phenomena occurring in urban road systems is their modeling and simulation. The first works in this field were published in the 1950s (Gerlough 1955). The first methods were based on modeling flows, without identifying specific vehicles. Over the next decades, research was moving towards more and more detailed microscopic models (van Wageningen-Kessels et al. 2015), which allow to represent the autonomy, diversity and unpredictability of the behavior of individual traffic participants. An obvious consequence of this direction is the growing complexity of models and algorithms for simulating changes in their state over time.

The need for large computing power and memory resulted in the search for methods of parallelizing computations and using multiple computers to simulate complex models. Many attempts of developing parallel versions of urban traffic simulations have been described over the last decades (Nagel and Schleicher 1994; O'Cearbhaill and O'Mahony 2005; Kanezashi and Suzumura 2015; Horni et al. 2016; Arroyo et al. 2018). The results clearly show, that the problem of urban traffic simulation using detailed microscopic models is not straightforward to parallelize due to several reasons. The computational task requires updating one common data structure (the model), using its previous state, which complicates the synchronization between computing workers. The simulation generates large volume of results, which cannot be processed in a centralized manner. Moreover, the size of computational task assigned to each worker changes over time as the simulated cars traverse the road network.

In this work, we present the architecture, the algorithms and their implementation in a new, distributed urban traffic simulation, called HiPUTS. All its elements were designed with distribution and scalability in mind, in order to allow effective utilization of thousands of computing cores of a modern High Performance Computing (HPC) machine. The proposed solution aims at providing two crucial features: scale invariance and distribution transparency. Scale invariance in this context is understood as the invariance of the task size of a single worker when increasing the entire computation with a proportional increase in hardware resources. This feature is the basis for the scalability of a distributed system by making the size of a worker's task independent of the scale of the problem. The distribution transparency is a feature of model processing algorithm in a distributed manner. The simulation model and the algorithm of its state changes over time should not be dependent on the presence or degree of distribution. In other words, the results obtained in the sequential and parallel versions should be identical.

In the next section the analysis of existing approaches towards parallel urban traffic simulation is presented. The details of the proposed architecture and algorithms are

described in Section 3. Systematic tests of the scalability of the developed implementation are described in the Section 4. The paper is summarized with the discussion of elements crucial for obtaining good scalability, presented in Sections 5 and 6.

2 Existing Scalable Traffic Simulations

In the late 20th century, the necessity for parallel execution in traffic simulation became apparent. The Nagel-Schreckenberg model, a widely utilized microscopic traffic model, was initially implemented in parallel by its creator back in 1994 (Nagel and Schleicher 1994). Subsequent research by the same group (Rickert and Nagel 2001) suggested that global synchronization in parallel traffic simulation algorithms might not be necessary, which is a key finding in the context of providing scale invariance. Eliminating all centralized components from a computations is crucial for achieving scalability on HPC hardware. However, transitioning to this approach in traffic simulations posed challenges.

Several endeavors to develop a centrally synchronized parallel traffic simulation have been reported in the recent decades. Research in (O'Cearbhaill and O'Mahony 2005) demonstrated scalability on a limited scale. The idea of extending the intervals between global synchronizations, as discussed in (Kanezashi and Suzumura 2015), resulted in improved scalability but also introduced notable errors in the simulation outcomes. In another approach outlined in (Toscano et al. 2012), a sophisticated protocol was proposed to rectify errors stemming from infrequent synchronizations.

One of the earliest successful attempts for defining a scale-invariant distribution method in a traffic simulation was outlined in (Xu et al. 2015). The authors introduced an Asynchronous Synchronization Strategy, where in each computational process interacts solely with a restricted subset of processes responsible for neighboring segments of the simulated environment. This strategy facilitated linear scalability with the involvement of 32 parallel workers.

Similar distributed architecture without centralized synchronization was used and extended in (Turek 2018), where traffic simulation scaled linearly up to 19200 parallel workers executed on 800 computing nodes of a supercomputer, proving the possibility of running HPC-grade traffic simulations. The simulation was able to simulate over 11 million cars with running speed at 160 steps per second, which is one of the largest scenario run at that time in the context of traffic simulations. The practical application of the solution was very limited - it used the discrete, Nagel-Schreckenberg model with a uniform grid of roads and it was not able to represent any real-life scenarios.

Using a simplified traffic model is one of the possible approaches, when simulation of large scenarios is needed. This approach was applied in one of the most popular traffic simulation tools nowadays, MATSim (Horni et al. 2016) (Multi-Agent Transport Simulation), which uses queues of cars to represent road lanes. Simplified models can provide high efficiency, however, they sacrifice accuracy and correctness in representing traffic behaviors. This trade-off is particularly relevant when simulating complex scenarios where small details can have a substantial impact on traffic flow. In terms of improving the efficiency, up to this point the main focus in MATSim was put on improving the existing engines to work as optimally as possible but still within a single process and a single computing node, utilizing concurrency and parallelization on multiple threads and parallel queues within MATsim internal architecture (Dobler 2010). Unfortunately, such an approach revealed its obvious limitations. MATSim is full of features and additional functions added during its long development, which increase the complexity of computations. Therefore, the simulation of complex scenarios is slow anyway and parallelization seems a desired feature now. Unfortunately, at this point it is hardly possible due to internal architecture and assumptions of the system.

Another very popular traffic simulation tool is SUMO (Acosta et al. 2016) (Simulation of Urban MObility). It uses a continuous space and state models with several variants of decision and motion modelling, and many additional features. Running large scale scenarios with complex traffic models quickly exceeds the abilities of a single computer, therefore several attempts to improve overall performance of SUMO by distributing computations (Acosta et al. 2016; Chen et al. 2020) or using new strategies for synchronization (Arroyo et al. 2018) have been reported. In all cases achieving either very limited scalability or even results worse than the centralized version of the simulation. Another problem of mentioned solutions is also a growing problem of increasing error with increasing scale.

Probably the first advanced traffic simulator which considered distribution at the design stage was SMARTS(Ramamohanarao et al. 2016) (Scalable Microscopic Adaptive Road Traffic Simulator). It is able to simulate continuous traffic models, import real-life models and execute computations on many computing nodes. These promising features encouraged us to investigate it carefully and improve its scalability features in cooperation with SMARTS creators (Najdek et al. 2021). In (Zych et al. 2022) the original architecture of the simulator has been modified and the introduced improvements allowed to scale efficiently to hundreds of cores. Unfortunately, SMARTS is not well-maintained and also suffers from several issues in its assumption. The most important problem is the lack of distribution transparency, which may lead to inconsistent results. When a vehicle requires information of the proceeding vehicle in front, and the proceeding vehicle is not processed by a direct neighbor computing node, then the information cannot be obtained due to the fact that there is no communication channel between the two nodes. This causes that distribution is not transparent, but depends on initial division of the model. This is a major flaw in this approach that should as well be addressed at the design stage.

3 The High Performance Urban Traffic Simulator

The design process of creating a new urban traffic simulation system was guided by four main aims:

- Support for continuous traffic models with the ability of extending the solution with new models in the future,
- Distribution transparency for ensuring equivalency of results in single- and multi-node execution.
- Super-scalability achieved by scale invariance of computations executed by distributed nodes and load-balancing mechanisms.
- Ability to simulate large-scale and real-life scenarios by importing data from open map services.

These aims led us to developing HiPUTS, the High Performance Urban Traffic Simulator. HiPUTS is implemented in Java and it is available for download at `https://github.com/hiputs/hiputs`. It is a fully functional tool, which can be used for simulating large-scale scenarios, efficiently utilizing distributed computational resources. The environment models can be imported from the Open Street Maps[1] format. The traffic can be generated randomly or according to specified rules. The simulation can represent single- and multi-lane roads, vehicles of different types, crossroads with and without traffic lights and many other extensions, including visualization, results collecting and aggregation, modelling of pedestrian crossings or public transport. While these extensions demonstrate the extensibility of the solution, they are not crucial in the context of scalability and distribution transparency, discussed in this paper.

3.1 Simulation models

The environment model consists of two main types of elements: lanes and junctions. The model is represented in the form of a directed graph with edges representing the lanes between the nodes, which represent the junctions as shown at figure 1. Each node can be connected by either one-way (fellow) or two-way (blue) road. Imported data is converted to the graph of lanes and junctions. During the import, several filtering and optimization operations are performed, which are required due to the quality of the source data. There are two types of junctions: bends and crossroads, where bends connect one or two lanes, while crossroads connect more. Each junction has a fixed, known location and each lane represents a straight line between the junctions. As a result, each lane has a fixed, known length and each junction knows the circular order of incoming and outgoing lanes. From the vehicle's perspective it contains information about its current lane and position on it, current speed, acceleration and others, such as its length, max speed of vehicle and route for this car, so it knows where to move.

[1] https://www.openstreetmap.org

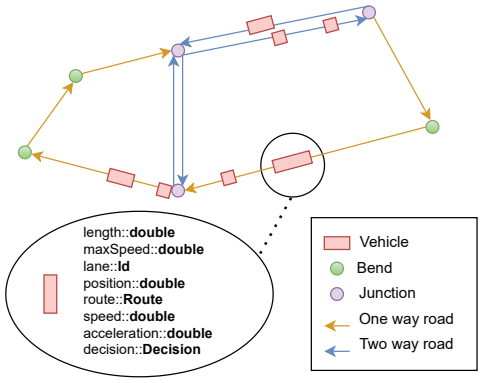


Figure 1: Representation of vehicle model and graph of environment

Each lane contains a list of cars currently present in the lane. Cars state is updated according to decisions generated by a set of so called *deciders*. Each car uses a basic car-following decider (which implements the IDM model (Treiber et al. 2000)) and a junction decider, which implements the right of way on crossroads. There are several other extensions which add the traffic model with multiple lanes, the MOBIL lane change model (Kesting et al. 2007), traffic lights on crossroads and overtaking on single-lane roads.

3.2 Distribution transparency

The environment model needs to be distributed between computing nodes. The road graph is divided into small area parts, which are the basis for ensuring distribution transparency and load balancing. Each computing node needs to contain information about its working area and information about adjacent parts and neighboring areas that are required for the correctness of the simulation. This approach is kind of based on a stencil pattern, but differs in the form of storing data, as the simulation data and state are not globally stored but distributed among computing units instead. Such approach allows also to be scalable in the context of data management. We are not only limited to a single node specification. Instead dividing the environment allows to work only on specific part of the simulation environment, resulting in speeding up overall computation and reducing memory usage overhead, as the result allowing also to simulate very large areas with heavy congestion.

Ensuring correctness in the distribution of the environment model is crucial. Its distribution cannot affect computation itself in any form or way, meaning that it should be transparent from the simulation point of view. Unfortunately, there are examples of existing solutions (Ramamohanarao et al. 2016) that do not address this issue so closely causing simulation to lose required information about e.g. vehicles during the communication process.

The Figure 2 presents the problem in the distributed simulation caused by the exchange of insufficient informa-

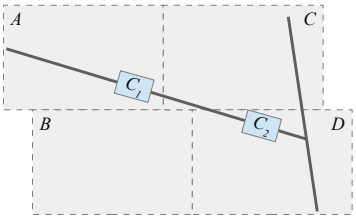


Figure 2: The problem of communication need between non-neighbour fragments

tion between adjacent nodes. Each computing node (A, B, C, D) simulates part of the environment, but A and D in this example are not direct neighbors. The simplistic model division boundary is shown by dashed lines in the mentioned example. Each node processes an assigned fragment of the environment and it needs to exchange information about the state of the fragment's border with the nodes responsible for adjacent fragments. However, determining how the border should be drawn is a challenge – it should be limited in order to reduce the communication overhead, however it has to provide all the necessary information. Even when the size of the border is properly estimated, errors still can occur when the arrangement of fragments causes the need for accessing data from indirect neighbors. This problem has been depicted in Fig. 2, where an improperly thought out division can cause problems. In this example, the car $C1$ is simulated in fragment A by one process and needs information about car $C2$ that is simulated in fragment D, but it is not a direct neighboring process. From the car $C1$ point of view it should contain information about its proceeding vehicle, but due to this division it will not obtain it, as the edge on which the vehicle is moving is cut by the process C (direct neighbour to A) and it does not contain information about $C2$ as it is only in D. For this reason process A will not obtain information about $C2$ causing error in simulation and resulting in different outcome.

We propose to solve these kind of problems by introducing the notion of *patches*. The road network is partitioned into disjoint fragments, called patches, which should be possibly small and guarantee the following feature: every decision algorithm in the simulation is based on the state of car's current patch and its direct neighbours only (allowing to maintain a required amount of neighbors to a minimum).

The proposed algorithm for partitioning the road network into patches is to use the grid composed of regular hexagons. To define the mesh, it is necessary to provide its orientation in a specific coordinate system and the length of its edge. The edge of each hexagon has to be equal to the maximum observation range of all decision algorithms. The indexation in the grid is shown in Fig. 3.

The geometric division of the road network is therefore determined by the coordinates of point A and the length of the side of the hexagon. The next step is to apply this division to the graph structure. Each node in the network model can be uniquely assigned to the interior of one

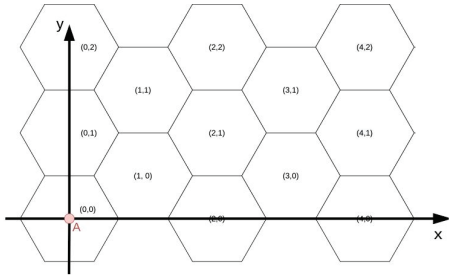


Figure 3: An example mesh of hexagons centered at point A along with their indexation

Figure 4: Exemplary division of a road network into patches

hexagon in which it is located. Such an assignment can be performed in linear time with respect to the number of vertices in the graph. An example of patches in a real-life network is shown in Fig. 4.

The proposed hexagon-based algorithm is not the only solution to the task of creating patches for a road network. However, it is definitely a simple and efficient solution for addressing this problem.

3.3 Parallel execution and scale invariance

The grid of patches can also be considered a graph, where each patch is a node with up to 6 neighbours. This graph is used for dividing the simulation task between the available computing nodes. Each of the patches is assigned an estimated computational cost, calculated as the total length of lanes in the patch. Then the graph of patches is partitioned using the k-means (Ahmed et al. 2020) clustering algorithm.

At the beginning of the simulation each computing node receives a list of patches to load. From the point of view of a one particular node, the patches processed by it are divided into three sets: internal patches, border patches and remote patches. The internal and border patches are updated by the node. The border patches have adjacent patches in the model, which are not processed by the current node. These adjacent patches are called the remote patches. An example of the types of patches is presented in Fig. 5

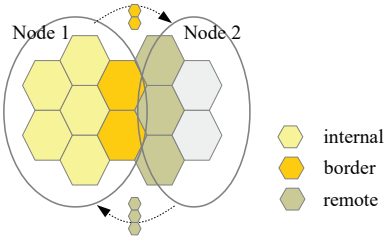
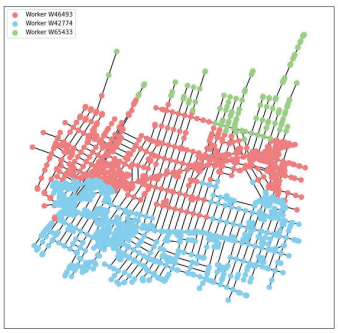


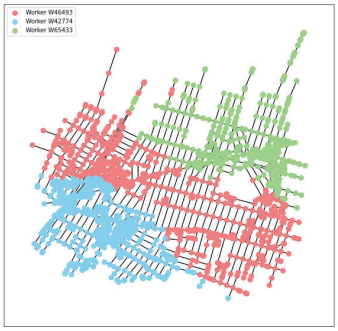
Figure 5: Internal, border and remote patches of exemplary *Node 1* and the exchange of patches' states between nodes

During a simulation step each node sends the state of border patches to the nodes responsible for adjacent remote patches. In return it receives the state of the remote patches (which are actually the border patches of the other nodes). The state of the remote patches contains all the information needed for computing decisions of all cars in the internal and border patches.

The initial partitioning of work between the available nodes cannot guarantee load balance throughout the simulation. The computational cost of updating a patch is not dependent only on the length of its lanes, but rather on the the number of cars currently present in the lanes. This dynamic load imbalance problem requires using a load balancing algorithm, which in our solution is based on exchanging patches between nodes.



(a) Exemplary traffic network during 1st simulation step



(b) Exemplary traffic network during 200th simulation step

Figure 6: Results of the load balancing mechanism on traffic network.

Each node measures the time needed for perform a simulation step computations and exchanges the value with its neighbours. When a significant difference is detected, a border patch is selected and sent to the less-occupied node. The selection of patches to send aims at preserving the integrity of the map fragments processed by the node, which is not always straightforward. The exchange is allowed once in a few steps in order to prevent rapid oscillations. In addition, only one source of transferred patches is allowed in a single step to prevent overloading the less-occupied node by sending it patches from several different nodes at a single step. These three mechanisms turned out to be sufficient to provide satisfactory balancing even in complex real-life scenarios, although there is still field for improvements in this area. A simple example of load balancing results are presented in Fig. 6a and 6b.

3.4 Simulation algorithm

The simulation calculations are carried out by workers who handle a specified number of simulation steps. The simulation can be divided into many workers, each one running on a separate computing node. The main simulation algorithm of a single simulation step can be described in a few stages. The overview is presented in Figure 7.

During each step, workers update vehicle parameters to reflect how they would realistically change over a time span equivalent to that step's duration. Neighboring workers may sometimes be at different stages of calculation or working on different time steps, but these discrepancies are short-lived and are resolved at local synchronization points, typically through message exchanges. This eliminates the need for centralized synchronization, which often creates bottlenecks in this type of computation. A given worker can proceed with simulation locally for the next $n + 1$ step if all of its neighbors have completed the given state of the current n step and sent the state of their changes from n step to the mentioned worker.

The entire algorithm is conceptually divided into two main stages that involve making decisions by vehicles located on lanes and then updating those vehicles' states. This solution was introduced to assess that simulation could be divided into fine-grained level tasks and parallelized. The first stage (decision) allows to perform calculations of traffic models for every vehicle on lane without making any physical changes to vehicles in the environment. Decision allows vehicles to store data about its state in the following $n+1$ step and not modify its current state in n step. This ensures that each vehicle makes decision based on the same state of environment in n step. Once all vehicles finish this stage they can be actually updated by moving them to the appropriate position on the lane and applying other changes to their state in the next iteration step of simulation, this can also be performed similarly to the previous stage in a parallel manner.

The first part of the discussed algorithm is that the vehicle determines its next state based on information from the current step. The decision is expressed by factors like positive or negative acceleration, speed, route changes,

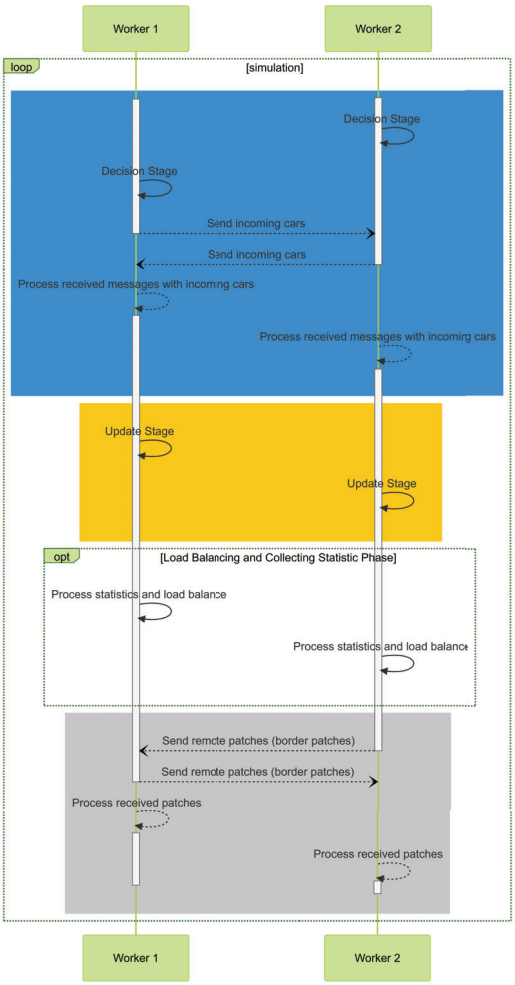


Figure 7: HiPUTS simulation algorithm overview of single step with 2 workers.

the lane the vehicle will occupy, and its position on that lane. The decision within itself involves applying different models (such as IDM, etc.) by taking into account vehicle's current state and the preceding vehicle's state. The decision is essentially a reflection of the vehicle's future state for the next iteration step and it is stored and managed by each vehicle to be used in the following part of the simulation stage (known as the update stage).

The decision part of the algorithm is divided into fine-grained tasks. Each task corresponds to a specific lane separately and is responsible for managing all vehicles on that lane within this task. Those tasks are then executed in parallel on a shared thread pool, with a size equivalent to the available computing cores on the node. This mechanism speeds up calculations by parallelizing execution of those tasks and utilizing all available computing resources – all cores of the computing node. During this stage, vehicles that decide to move to another lane are added to the buffer of their destination lane. This buffer is used to ensure synchronized access to shared lanes among different tasks in the thread pool.

Once the decision-making stage is completed, the worker immediately begins dispatching vehicles located on the boundary lanes that separate his work area from neighboring workers. When all workers finish sending to every neighbor within their work area vehicles to their remote buffers of each border lane then they await to receive all messages for this part in the background and start to process them. The message sending is done asynchronously which is shown with a dotted line in Figure 7. After messages are sent, worker loses control (what is visualized with a block on the timeline) up to the point when all incoming messages from all direct neighbors are received. The whole part including decision stage is highlighted with a blue background. Once the worker sends these messages to each of its neighbors, it awaits status updates from each neighbor. After that, it can freely proceed further to update stage of the simulation. This is due to the fact that from a single worker's perspective, it has all the required knowledge to proceed further in the simulation. The update stage involves applying previously made decision and changing the vehicles's state appropriately for this simulation step. The stored decision of each vehicle is actually applied to it, changing its current state. This is another task which is parallelized in the same manner as the decision stage.

The following part of algorithm contains an optional feature that involves load balancing and uses statistical data of monitored worker performance parameters. It was described in more detail in subsection 3.3 .

The last part involves sending the state of border patches after they are changed due to the update phase. They are shared to ensure correctness and to spread required knowledge of environment after the update stage. For this reason, these updated patches need to be sent to other computing nodes. From the perspective of another worker, it updates the state of its remote patches using the received state of neighboring border patches. Once all these stages are complete the simulation can proceed to the next iteration step of the simulation loop. This part is marked in grey in the algorithm overview shown in Figure 7.

4 Evaluation

In order to investigate benefits offered by the described simulator architecture, experiments based on weak and strong scalability laws were prepared. However, for examination of HiPUTS' super-scalability feature, in particular it's scale invariance requirement, an evaluation conducted in accordance with Gustafson's law (Gustafson 1988) was more essential (weak scalability). This is due to necessity to ensure equal amount of work per each worker while both, number of processors and problem size, increase. Such demand may indeed result in highly satisfying scalability outcome.

Test scenario in both experiments was based on a synthetically generated traffic network, comprising perpendicular, intersecting roads, connected at the ends to form a torus. The roads were two-way and the distance between adjacent intersections was equal to 500 meters. Each patch included 4 connected intersections placed at the vertices

of a square. A single time step represented 1 second of real time. The traffic network had a constant number of cars during simulation (cars were regenerated when they reached their destination). For each scalability experiment two tests with different car density were prepared. In the first case average distance between cars was equal to 30 meters. In the second one, the distance was 60 meters. HiPUTS used load balancing strategy of transferring road network patches, by moving them from the most workloaded neighbour to the least burdened one. Each run computed 1500 simulation time steps. The presented performance results were computed based on the last 1000 steps to exclude strongly variable overhead associated with JVM initialisation. The majority of the tests were repeated at least 10 times. The only exceptions were executions using 1 and 96 nodes during strong scalability tests with vehicles an average of 60 meters apart, which were executed 8 and 4 times respectively due to limitations in hardware availability.

Experiments were executed using Ares supercomputer located in ACK Cyfronet AGH in Krakow, Poland, which is part of the PL-Grid infrastructure. Currently Ares is ranked 442th on the TOP500 list [2]. This supercomputer is composed of Xeon Platinum 8268 24C 2.9GHz processors connected by Infiniband EDR. In total Ares offers 37 824 computing cores. Experiments were performed using up to 96 nodes, each containing 48 cores. This gives a maximum number of 4 608 computing cores used. One simulator's worker was assigned to one node.

4.1 Weak Scalability

One aim of evaluation was to examine simulator's scale invariance, which assumes constant size of overall problem per worker regardless of their total number. To accomplish this, problem size needed to increase proportionally with number of workers. Therefore, as mentioned previously, weak scalability experiments were more relevant in this regard.

In the following tests each worker was initially responsible for a network comprising 65 536 intersections. Number of cars was equal to 2 184 534 or 4 369 067, depending on the experiment. Overall problem size changed as the number of workers increased, thus with 4 nodes traffic network grew to 262 144 junctions and over 17 million cars in the tests with 30 meter average distance between cars.

Results of total simulation time (fig. 8) show interesting trend where the increase in mentioned time stabilises after around 12 used nodes and only slightly grows when more workers are introduced. Reason for shortest computation time, when only 1 node was used is natural and related to no communication, synchronization and load balancing overhead, which also shows fig. 9. As the number of nodes increases, each worker has to interact with a greater number of neighbours. However, this relationship is no longer accurate since the use of 12 workers. The primary reason for this is constant initial number of neighbourhoods (equal to 4) when total number of workers is 12 or greater. Therefore, discussion about scale invariance of HiPUTS should start from this point, since this is when its requirements begin to be met.

[2] https://www.top500.org/lists/top500/list/2024/06

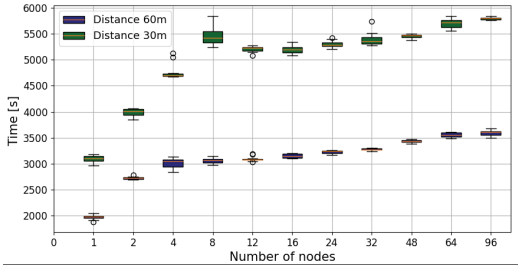


Figure 8: Total simulation time depending on used number of nodes

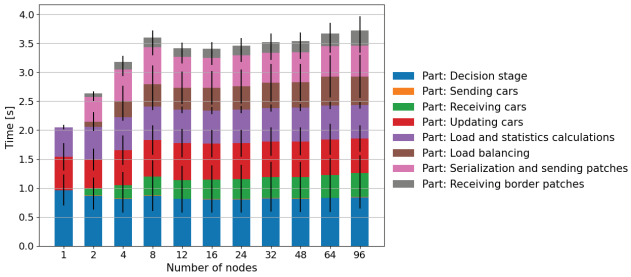
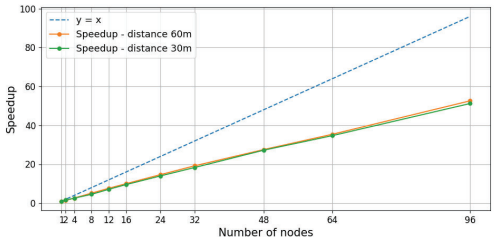


Figure 9: Average duration of one simulation step subdivided into stages (average cars distance equal to 30m)

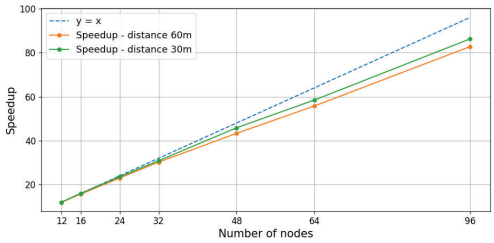
An analysis of average time spent on each part of simulation step (fig. 9) strongly suggests ensuring scale invariance by the simulator. Data show that since the use of 8 nodes, duration of particular calculation stages has remained highly close to constant, while number of nodes has increased. This observation follows that addition of another worker in HiPUTS does not affect performance of the other computing units.

The speedup achieved by HiPUTS simulator is decent, but far from ideal (fig. 10a). Metric calculation was based on the duration of execution with only 1 worker. Achieving similar simulation time while incorporating communication and synchronization processes is a challenging task. For this reason, obtaining of linear speedup while problem size increases and additional overhead occurs is difficult when related to performance on 1 node. However, calculating metrics from the point of ensuring scale invariance yields significantly different results and highlight the impact of this feature on performance (fig. 10b). Achieved speedup, obtained in relation to 12 nodes, is highly close to linear.

Described results clearly show that scale invariance ensures constant calculation time, resulting in almost linear speedup as both problem size and amount of computing power increase.



(a) in relation to single node (increased communication)



(b) in relation to 12 nodes (constant communication)

Figure 10: Speedup of HiPUTS simulator

4.2 Strong Scalability

Evaluation of HiPUTS performance included also experiments based on Amdahl's scalability law. During this ex-amination problem size was constant, thus parts assigned to each worker varied inversely proportional to the number of nodes used. Traffic network distributed among workers had a size of 1 048 576 junctions and contained 69 905 072 or 34 952 536 cars, depending on the task.

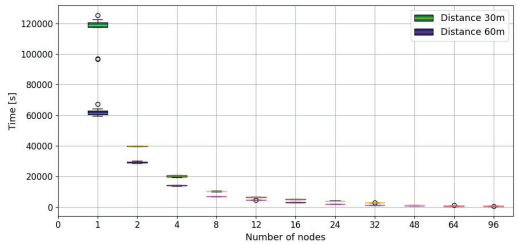


Figure 11: Total simulation time depending on used number of nodes

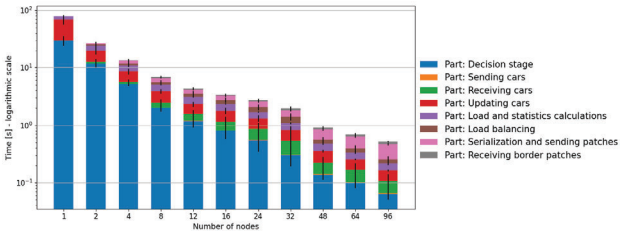


Figure 12: Average duration of one simulation step subdivided into stages (average cars distance equal to 30m)

As expected, when the number of nodes increases, the total simulation time and average duration of a simulation step decrease (fig. 11 and fig. 12).

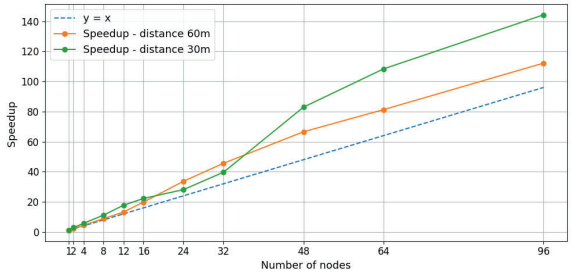


Figure 13: Speedup

However, from a certain point onwards (around 48 used nodes), the addition of more workers results in a diminishing time saving, which is consistent with Ahmdal's observations (Amdahl 1967). The decrease in computation duration between 1 and 2 nodes used is greater than linear in experiments with 30 meter distance between cars and close to linear in the other test. These results are surprising considering the additional overheads associated with communication and load balancing. Although these costs are relatively low for small number of nodes, their proportion increases as the number of nodes grows. The mentioned over-linear reduction of the simulation time results in a super-linear speedup (fig. 13).

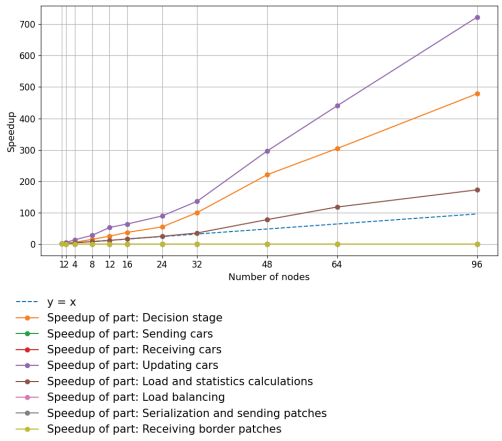


Figure 14: Speedup of each simulation part (average cars distance equal to 30m)

Analysis of the figure 14 reveals that this is due to the excellent scalability of only certain stages of the calculations - the decision stage, updating cars parameters step and calculations of load and statistics phase. All these are entirely computational stages and the two former are embarrassingly parallel tasks, which were executed in this manner. The remaining stages of simulation did not occur in the tests with 1 worker, thus they decelerate execution of the programme and are hidden on figure behind "Receiving border patches" line. The greatest speedup in experiment with 30 meters distance is achieved when the simulation was executed with more than 32 workers. Most probably this is due to the relatively small size of the problem, which resulted in fewer references to RAM and a less frequent need to reload the cache memory, than during the tests with 1 node.

Undoubtedly, HiPUTS simulator is highly scalable, mainly due to the suitable parallel implementation.

5 Towards Super-Scalability

Achieving proper scalability in large-scale distributed systems that iteratively modify a common data structure is a challenge. According to (Engelmann and Geist 2005), a super-scalable system must exhibit scale-invariance, meaning each computing unit maintains a constant workload under weak scaling. This includes not only computational tasks but also the volume of data to be processed, the number of communication partners, message volume, initialization and shutdown tasks, and results collection. Any violation of this principle can lead to bottlenecks and limit scalability.

There are several factors in a distributed urban traffic simulator that may violate scale-invariance. The HiPUTS system was specifically designed to address these challenges, focusing on various aspects of scalability:

Computational task invariance is relatively easy to ensure at the beginning of the simulation. Each patch of the model is assigned a computational cost estimate based on the total lane length and initial vehicle count. Workers are then assigned clusters of patches with similar total costs, ensuring an initially balanced workload.

Load balancing is necessary throughout the simulation to preserve an equal computational effort across nodes. As congestion builds up in specific areas, those areas require more computational resources. To manage this, a mechanism allows patches to be exchanged between neighboring workers. This introduces a minor communication overhead but prevents computational hotspots and ensures load distribution remains even.

Communication invariance is one of the most crucial factors for scalability. Even a single centralized communication point can create a bottleneck, limiting overall performance. To avoid this, HiPUTS is designed around local communication only. Each computing node interacts with a fixed number of neighbors, sending and receiving a consistent volume of messages, regardless of the overall size of the model. The only centralized step is the initial exchange of node addresses, after which synchronization occurs naturally through local interactions.

Static model data consists of the road network, junctions, and patches. In HiPUTS, each worker loads this data independently. While this approach does not fully comply with scale-invariance, as memory usage grows with the size of the simulation, it does not negatively impact efficiency once the data is loaded. On the contrary, it enables various optimizations, such as facilitating load balancing and distributing path planning. If memory constraints become an issue, these optimizations can be omitted.

Path planning can become a major scalability concern when traditional algorithms such as A* are used, as their complexity is dependent on the size of the graph. To mitigate this, alternative solutions can be applied. A simple random-walker approach allows vehicles to select their next junction randomly, eliminating the need for complex computations. When more structured or controlled traffic is required, paths can be precomputed for all vehicles and stored in files or a database. Since path generation can be parallelized, this method allows trajectories to be reused across multiple simulation runs, further improving efficiency.

Parallelization within a computing node is another critical factor for scalability. Even if computational loads are evenly distributed among nodes, poor multi-core utilization can degrade performance. In HiPUTS, all major simulation tasks, including inter-node communication, vehicle decision-making, and state updates, are divided into fine-grained tasks executed via a thread pool. This ensures efficient CPU utilization and prevents local computational bottlenecks.

Graph partitioning is an important preprocessing step required for efficient simulation. This process consists of two stages: generating the patch graph and partitioning it among computing nodes. While uniform grids simplify partitioning, real-world road networks introduce additional complexity. Fortunately, the patch graph is significantly smaller than the full road network, making partitioning a less demanding task. In HiPUTS, a k-means-based partitioning method is used, but more sophisticated techniques can be applied if needed. Since partitioning results are stored, they can be reused in subsequent simulations with the same model and computational resources.

Results collection and processing is another key consideration. Running a simulation without collecting output data is meaningless, but improper data aggregation can create performance bottlenecks. To ensure scalability, HiPUTS employs a decentralized approach in which each node independently records results and writes them to files. While this prevents real-time global statistics computation, it eliminates centralization and maintains performance. The collected data can then be analyzed post-simulation.

The design choices outlined above enabled HiPUTS to achieve a high level of scalability. Many of these solutions are not specific to traffic simulation and can be generalized to other large-scale distributed simulations, serving as valuable guidelines for building super-scalable systems.

6 Conclusions and Further Work

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.

The next steps in the HiPUTS development will be the verification of its features during simulations of large-scale real-life scenarios. Detailed simulation models together with the ability of simulating large cases very fast may open new areas of applications for this type of tools.

Acknowledgements

The research presented in this paper was funded by the National Science Centre, Poland, under the grant no. 2019/35/O/ST6/01806. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2023/016691

References

A. Acosta, J. Espinosa, and J. Espinosa. Distributed simulation in sumo revisited: Strategies for network partitioning and border edges management. In *Proceedings of the 4th SUMO User Conference*, pages 61–71, 2016.

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

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.

N. Arroyo, A. Acosta, J. Espinosa, and J. Espinosa. A new strategy for synchronizing traffic flow on a distributed simulation using sumo. *EPiC series in engineering*, 2:152–161, 2018.

H. Chen, K. Yang, S.G. Rizzo, G. Vantini, P. Taylor, X. Ma, and S. Chawla. Qarsumo: a parallel, congestion-optimized traffic simulator. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, pages 578–588, 2020.

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

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.

D.L. Gerlough. *Simulation of Freeway Traffic on a General-purpose Discrete Variable Computer*. University of California, Los Angeles, 1955.

J. Gustafson. Reevaluating Amdahl's Law. *Commun. ACM*, 31(5):532–533, 1988. ISSN 0001-0782.

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

H. Kanezashi and T. Suzumura. Performance optimization for agent-based traffic simulation by dynamic agent assignment. In *2015 Winter Simulation Conference (WSC)*, pages 757–766. IEEE, 2015.

A. Kesting, M. Treiber, and D. Helbing. General lane-changing model MOBIL for car-following models. *J. of Transportation Research Board*, 1999(1):86–94, 2007.

K Nagel and A Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20(1):125 – 146, 1994. ISSN 0167-8191.

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.

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.

K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E.B. Khunayn. SMARTS: Scalable microscopic adaptive road traffic simulator. *ACM Trans. on Intelligent Systems and Technology (TIST)*, 8(2):1–22, 2016.

M. Rickert and K. Nagel. Dynamic traffic assignment on parallel computers in transims. *Future Generation Computer Systems*, 17(5):637 – 648, 2001. ISSN 0167-739X.

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.

M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

W. Turek. Erlang-based desynchronized urban traffic simulation for high-performance computing systems. *Future Generation Computer Systems*, 79:645–652, 2018.

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.

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.

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.