Distributed Architecture for Highly Scalable Urban Traffic Simulation

Michał Zych, Mateusz Najdek, Mateusz Paciorek, and Wojciech Turek

AGH University of Science and Technology, Krakow, Poland

Abstract. Parallel computing is currently the only possible method for providing sufficient performance of large scale urban traffic simulations. The need for representing large areas with detailed, continuous space and motion models exceeds the capabilities of a single computer in terms of performance and memory capacity. Efficient distribution of such computation, which is considered in this paper, poses a challenge due to the need of repetitive synchronization of the common simulation model. We propose an architecture for efficient memory and communication management, which allows executing simulations of huge urban areas and efficient utilization of hundreds of computing nodes. In addition to analyzing performance tests, we also provide general guidelines for designing large-scale distributed simulations.

Keywords: Urban traffic simulation · Simulation scalability · HPC

1 Introduction

The demand for reliable simulations of complex social situations draws attention of scientists towards the problems of simulation algorithms efficiency. Providing valuable results requires detailed models of society members and their environment, which, together with the need for simulating large scenarios fast, exceeds the capabilities of sequential algorithms. However, the problem of parallel execution of such simulations is not straightforward. The considered computational task is focused on repetitive modifications of one large data structure, which, when performed in parallel, must be properly synchronized. There are a few successful examples [14,10] of parallel spacial simulation algorithms, however, efficient utilization of HPC-grade hardware for simulating real-life scenarios with continuous space and motion models remains an open problem.

In this paper we present the experiences with scaling the SMARTS simulation system [11], which is presumably the first distributed simulator for continuous urban traffic model. This work is the continuation of the preliminary research presented in [8], where basic scalability issues were identified and corrected, making it possible to execute a complex simulation task on 2400 computing cores of a supercomputer. Here we propose a redesigned architecture of the simulation system, which aims at overcoming limitations of the original approach.

The proposed architecture, together with other improvements introduced to the SMARTS simulation system, made it possible to execute a simulation of

100,000 square kilometers of intensively urbanised area with almost 5 million cars. The system efficiently utilized 9600 computing cores, which is presumably the largest hardware setup, executing a real-life scenario of urban traffic simulation, reported so far.

The experiences with discovering the reasons for scalability limitations of the original implementation may be also valuable in other applications. Therefore, we provide a summary of guidelines for implementing super-scalable spacial simulations. The guidelines concern the architecture of a distributed simulation system, communication protocols and data handling. We believe that the summary may be valuable for researchers and engineers willing to use HPC-grade hardware for large-scale simulations.

2 Scalable Traffic Simulations

The attempts towards implementing improvements of parallel urban traffic simulation have been present in the literature since the end of the 20th century – popular microscopic model of traffic, the Nagel-Schreckenberg model, was the first to be executed in parallel by its creator in 1994 [7]. Further research by the same team [12] suggested that the global synchronization in parallel traffic simulation algorithm might not at all be necessary. Despite that, several centralized approaches to the problem were tried later on. Methods described in [6] and [9] have shown efficiency improvements only up to a few computing nodes. Identified problem of scalability limitations caused by synchronization was addressed in [2] by increased time between global synchronization. It allowed better efficiency, but also influenced the simulation results, which cannot be accepted. The work presented in [13] proposed a method for solving this issue by introducing a corrections protocol. This, however, again resulted in poor scalability.

Achieving significant scalability requires dividing the computational tasks into parts, which are scale-invariant [1]. This refers to the number of computations but also to the volume of communication, which cannot grow with the number of workers. Centralized synchronization of parallel computation guarantees just the opposite.

Presumably, the first architecture of distributed traffic simulation system, which follows these guidelines was described in [16]. The proposed Asynchronous Synchronization Strategy assumes that each computing worker communicates with a limited and constant number of other workers – those responsible for adjacent fragments of the environment. The paper reports linear scalability up to 32 parallel workers.

The proposed distribution architecture was used and extended in [14], where traffic simulation scaled linearly up to 19200 parallel workers executed on 800 computing nodes. The simulation of over 11 million cars run at 160 steps per second, which is presumably the largest scenario tested so far in the domain of traffic simulations. It opened new areas of application for such simulations, like the real-time planning presented in [15]. While proving the possibility of creating efficient HPC-grade traffic simulation, the implementation used discrete model and did not support real-life scenarios. In order to provide these experiences in a useful tool, we decided to investigate the possibility of redesigning the architecture of an existing, highly functional traffic simulation tool. We selected the SMARTS simulator [11], which supports importing of real cities models and simulated continuous traffic models. The details of the original design and the introduced improvements will be presented in the following sections.

3 The SMARTS System

SMARTS *(Scalable Microscopic Adaptive Road Traffic Simulator)* [11] is a traffic simulator designed with an intent to support a continuous spatial model of the environment with a microscopic level of details. The simulation is executed in time steps representing a short duration of real-world time. During the simulation, each driver makes its own decisions based on one of the implemented decision models and the state of the environment. Two decision models are used: the IDM (*Intelligent Driver Model*) [4] predicts appropriate acceleration based on the state of the closest preceding vehicle while the MOBIL (*Minimizing Overall Braking Induced by Lane Changes*) [3] triggers lane changes. SMARTS is implemented in Java. The most important features of SMARTS are:

– Loading data from OSM (OpenStreetMap [1]).
– Static or dynamic traffic lights control.
– Commonly-used right-of-way rules.
– Different driver profiles that can affect the car-following behavior.
– Public transport modelling.
– Graphical user interface for easy configuration and visualization of results.
– Various types of output data, such as the trajectory of vehicles and the route plan of vehicles.

SMARTS provides a distributed computing architecture, which can accelerate simulations using multiple processes running at the same time. The initial architecture of SMARTS (shown in Fig. 1) includes a server process and one or more worker processes, which are responsible for performing the actual computation. Worker processes communicate with each other through TCP connections using sockets. The architecture had a significant drawback, as it forced to run each worker in its own separate virtual machine within shared cluster node, which required environment map to be loaded from the drive and parsed by each worker individually, causing increased heap memory usage.

SMARTS also uses two mechanisms to synchronize the simulation. The first mechanism is a decentralized simulation mode. The very first version of SMARTS ran in a centralized mode, where the server needed to wait for confirmation messages from all the workers and to instruct all the workers to proceed at each time step. Later, the decentralized mode was added. In this mode, the workers do not need to communicate with the server during the simulation stage.

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

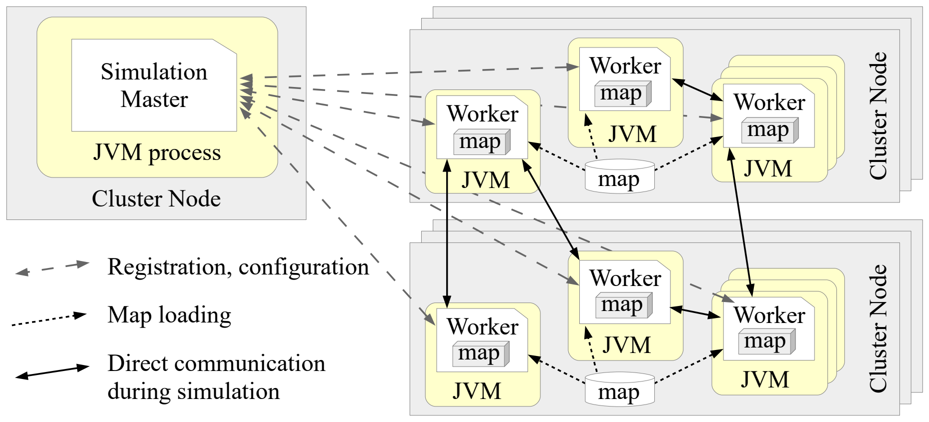


Fig. 1: Initial architecture of the simulation system.

The second mechanism is called Priority Synchronous Parallel (PSP) model, where a worker uses a two-phase approach to reduce the impact of the slowest worker [5]. During the first phase, the worker simulates the vehicles in a high priority zone at the boundary of its simulation area. This zone covers all the vehicles that may cross the boundary of the area at the current time step. During the second phase, the worker sends the information about the border-crossing vehicles to its neighbors while simulating the rest of the vehicles, i.e., the vehicles that are within the boundary but are not in the high priority zone.

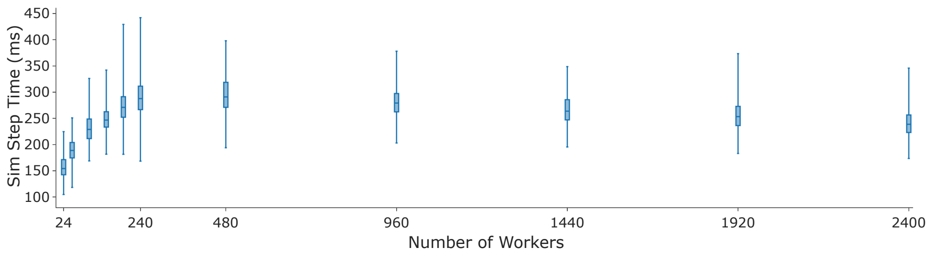


Fig. 2: Simulation step times for problem size increasing in proportion to the number of workers.

The original version of SMARTS [11] was only able to utilize up to 30 workers, depending on simulated environment size, which also determined the limits of the original scalability tests. After our initial improvements, presented in [8], we were able to successfully run simulation that involved the number of vehicles growing in proportion to the increasing computing power (1000 vehicles per core), with execution times shown in Fig. 2. The experiment was based on the road network in $59km \times 55km$ area in Beijing, China. The real-world map model was prepared from OSM map. The road network graph contained approximately 220 thousand nodes and 400 thousand directional edges, which represent roads in this model. The whole simulation process consisted of 1500 steps with a step duration equal to 200 ms. It is equivalent to 5 minutes of real-time traffic. The execution times of the last 1000 steps were measured and the average time of one in every 50 steps was recorded. The software architecture of this solution was not yet optimized for High Performance Computing Systems, although it showed promising results. All average execution times of simulation steps were within similar range and the median stabilized on the level of about 250 ms.

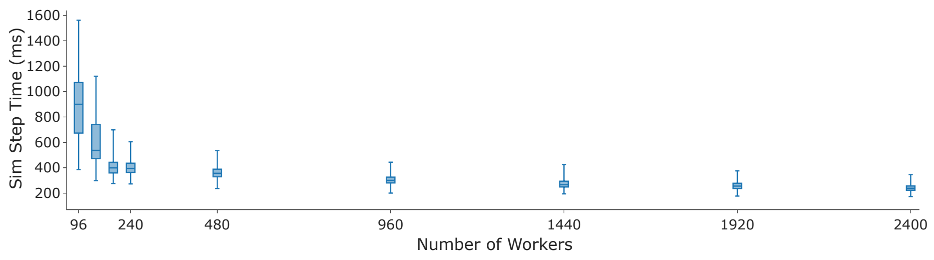


Fig. 3: Simulation step times with fixed problem size for increasing number of workers.

We also prepared an experiment to analyze the strong scalability for the original architecture. It involved a predefined number of 2.4 million vehicles in total, distributed across whole environment. Fig. 3 shows that in each case the increasing number of workers, and hence the computing power, resulted in reduction of average simulation time of a single step, therefore improving overall simulation performance.

The original architecture exposed significant limitations in these experiments. The scalability was not satisfactory and the system was unable to run scenarios larger that the one used in the tests.

4 New Communication Architecture

Despite the improvements proposed in the previous paper [8], SMARTS still have limitations that prevent efficient executions of large scenarios. The main problems are: the inability to run simulations above 2400 cores and memory leaks when simulating huge areas.

These memory problems are mainly caused by the requirements posed by the Worker-based architecture, to store the entire map in each Worker. A computing node with multiple cores to exceed all available memory when simulating a large map. It is important to note that only the area simulated by a Worker is changed, thus the rest of the map is used for read-only operations (e.g. path planning). The other crucial problem is the impossibility to start the simulation correctly if more than 2400 Workers are used. The server trying to handle all server-worker connections, needs to create too many threads, which consequently causes an error at the initialization stage.

This paper proposes a new architecture for the simulator, which is shown in Fig. 4. A proxy instance, the WorkerManager, was created to handle the communication with the server and to manage the lifecycle of the Workers that are under its control. By using this type of architecture, the number of JVMs created on a node was reduced (1 JVM per node instead of 1 per core). By appropriately assigning tasks to the proxy instance, the main problems of the simulator can be eliminated.

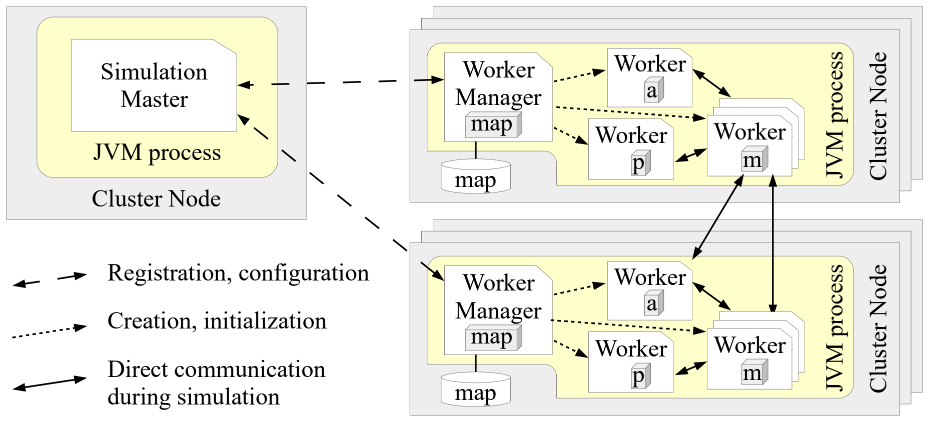


Fig. 4: Modified architecture of the simulation system.

By using a WorkerManager instance, all read-only objects can be stored only once in node memory. While the case is obvious for any kind of constants in the simulation, it is not possible to store the whole map only once, because each Worker uses part of it to execute the simulation. In this case, as shown in figure Fig. 4, the following solution was selected: WorkerManager stores the whole map area, which is used in read-only mode to generate vehicle routes during the simulation, while at the initialization stage individual Workers copy only the part of the map they are responsible for and a small fragment on the boundary of the map division between Workers.

The use of separate simulation zones for individual Workers has also improved simulation performance. Each core now searches for vehicles only in the part of the map for which it is responsible. As a result, the work required to search for vehicles in each step of the simulation has been reduced, which has a positive impact on its efficiency.

Another significant change due to the new architecture is the possibility of aggregating server-worker communication. As shown in Fig. 1, in the previous implementation the communication between the server and the Worker was direct. As can be seen in Fig. 4, the way of communication was changed by creating a dedicated proxy instance. The server currently communicates only with WorkerManagers. This results in a reduction of threads created by the server which are responsible for handling communication. In addition, the server, thanks to the message from the WorkerManager, possesses information which Worker is controlled by which WorkerManager. Thanks to this, it aggregates messages that should be sent to each of the Workers and sends them as a single message to the proxy instance. In addition, for optimization purposes, the common parts of messages were separated so as not to duplicate the information sent to the WorkerManager.

What is also important, the communication between individual Workers during the simulation has not been changed in any way. It still takes place directly in an asynchronous way. The server sends information about all Workers, so they can establish direct communication.

Through the use of proxy instances and communication aggregation, a new algorithm for the initialization and finalization of the simulation was created. The new method of initialization can be described as follows:

- Creating WorkerManagers and registering them to the server.
- The server creates a map using the OSM data and divides it into a desired number of fragments. Then it sends all simulation settings, such as path to the file containing the simulation map, number of Workers and other simulation properties.
- WorkerManagers create a map of the simulation using the file and validate the map (the server sends a hash of the map it created from the file). After this stage is completed they send back a message to the server about the successful creation of the map.
- Server orders the creation of Workers.
- WorkerManagers create a desired number of Workers and send their data to the server.
- Server assigns map fragments and initial number of vehicles for each fragment to individual Workers. Then this information is sent to WorkerManagers.
- WorkerManagers pass the message to individual Workers. Each Worker copies the corresponding fragment of the map and generates vehicle routes. When every Worker finishes this step, the WorkerManager sends a message about being ready to start the simulation.
- After receiving all messages about readiness, server orders to start the simulation stage.

Using the new architecture allowed the simulation to run correctly with up to 9600 computing cores. In addition, the server sends far fewer messages at the initialization stage than before, which has a positive impact on the performance of this stage of the entire process. Moreover, by storing common parts such as a simulation map only once, it is now possible to run much larger test scenarios.

Another improvement is the new distribution of the initial number of cars to individual Workers. Previously, when calculating the number of cars, the value was always rounded down and the last Worker was assigned the rest of the cars. In the case of the largest scenario tested in this study, such a division resulted in the last Worker receiving about 9000 cars, while the rest received only 499. This caused an uneven imbalance from the very beginning of the simulation. The presented example shows how important the seemingly non-obvious details are during the design of systems for such a large scale.

The extended versions of the SMARTS system, used for conducting the presented experiments, is available online[2].

5 Evaluation

In order to measure the scalability of the simulation using the proposed architecture, two experiments were prepared. Both experiments used the same simulation scenario: cars navigating the road network from a real-world area. The map represented the area of size $101,000\ \mathrm{km}^2$ in north-eastern China, containing Beijing and neighboring cities. The map was created with use of OSM data.

In both experiments each worker process was assigned to one computing core, thus the names "worker" and "core" are used interchangeably when the quantities are discussed. The experiments are explained in detail in their respective sections below.

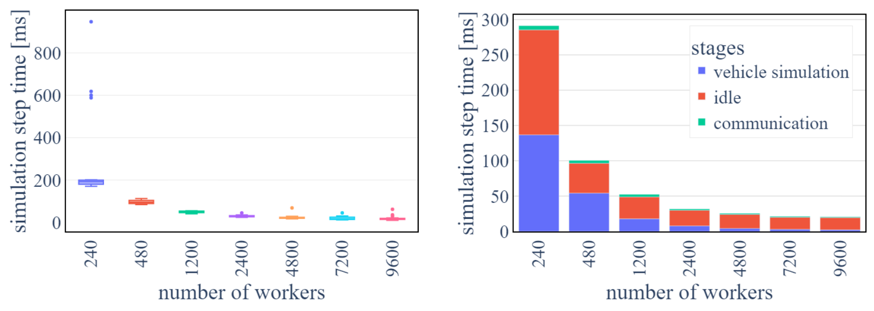
The evaluation was executed with the use of the Prometheus supercomputer located in the AGH Cyfronet computing center in Krakow, Poland. Prometheus is the 440th fastest supercomputer (TOP500[3] list for November 2021). It is a peta-scale (2.35 PFlops) cluster using HP Apollo 8000 nodes connected via InfiniBand FDR network. Each node contains two Xeon E5-2680v3 12-core 2.5GHz CPUs. The total number of available cores is 55,728, accessible as HPC infrastructure with a task scheduling system.

5.1 Constant number of cars

The first experiment followed the usual rules of strong scalability evaluation: the size of the problem was constant and the number of computing power was increased to analyze the scalability of the solution. The number of cars was set to 4.8M. The numbers of cores used for the execution of the experiment were: 240, 480, 1200, 2400, 4800, 7200, and 9600, allocated in groups of 24 per node due to the architecture of used computing environment. The size of the experiment precluded the execution on number of cores lesser than 240, one of the reasons being the portion of a problem assigned to single node exceeding memory limits.

[2] `https://github.com/zychmichal/SMARTS-extension`
[3] `https://www.top500.org/system/178534/#ranking`

Fig. 5 shows the measured execution times, sampled from all workers in various iterations. As can be observed in Fig. 5a, the execution on 240 cores resulted in several values that do not match the general distribution of measured times, achieving more than four times the median value. These values are caused by Java Garbage Collector invocations, which were required due to the size of tasks assigned to single nodes. After increasing the number of nodes such situations did not occur.



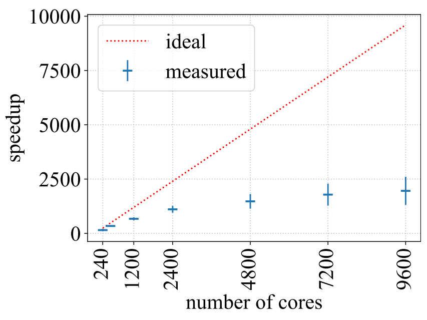
(a) Box-and-whisker plot (b) Average times of time-step stages

Fig. 5: Simulation time-steps with constant number of cars for increasing number of workers.

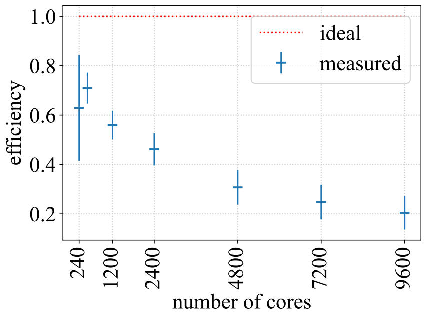
The measured times were analyzed using the *speedup* and *efficiency* metrics. The speedup should be calculated with the reference to single-core execution of the program. As the execution of the experiment on a single worker is not feasible, a different way of determining reference value was used. Assuming constant amount of work to be performed in each experiment, the average time of vehicle simulation (i.e. the time not spent in communication or waiting for other workers) in 240 cores run, multiplied by 240, was chosen as the approximation of single-core execution time.

Fig. 6 shows the described metric values derived from the results presented in Fig. 5. As can be observed in Fig. 6a, the speedup does not follow the ideal line that could be observed for embarrassingly parallel problems. It is expected, as both communication and waiting for other workers contribute to the longer execution times, which impacts the speedup. Nevertheless, by eliminating the bottleneck in form of single point of synchronization, the speedup grows when new resources are added, achieving ca. 2000 for 9600 cores.

The decrease in the benefits from adding new cores is also seen in Fig. 6b, which shows the efficiency based on the measured times. The outlying values for 240 cores impacting the average cause the efficiency for this execution to be significantly worse than expected, which can be additionally observed in the large standard deviation in efficiency for this run.



(a) Speedup



(b) Efficiency

Fig. 6: Speedup and efficiency in experiment with constant problem size. Blue markers show mean value and standard deviation, red dotted lines show ideal values for linear scalability.

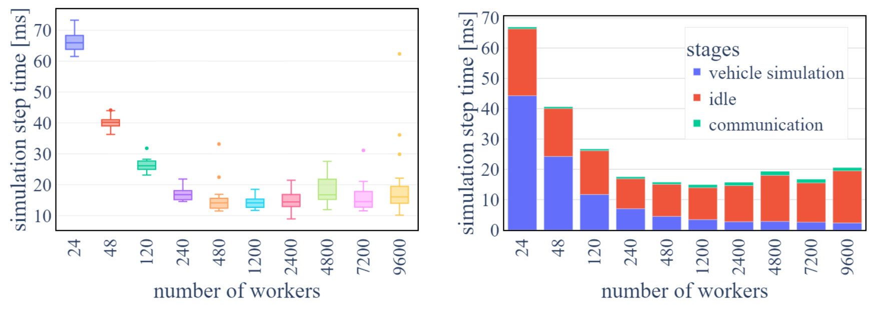
The most significant cause for this loss of efficiency is the uneven distribution of workload. The time of idle waiting for the other workers (Fig. 5b) reduces as the cores are added, but its proportion to the total time of simulation increases. This limits the benefits that can be obtained by adding the resources. The uneven load might result from the fluctuations in the car density — if an area simulated by any given worker contains less cars than the average, then some other workers have to simulate more cars than the average, and all other workers will have to wait until the most loaded one finishes its work.

The results presented in Fig. 3 were obtained in an experiment similar to this one, although with half the number of cars and smaller road network. The median execution time of a single time step for the largest number of cores (2400) was ca. 250 ms. Therefore, a good approximation of expected average execution time of a single step for the same scenario as the experiment presented above would be ca. 500 ms. However, the same number of cores yielded average execution time of ca. 60 ms. The new architecture is clearly outperforming the old one, achieving the execution time 5 times better for large numbers of computing resources.

5.2 Growing number of cars

The second experiment was inspired by the weak scalability evaluation. However, due to the predefined and irregular simulation environment, it was impossible to scale the environment to match the computing power. Therefore, the only parameter that was changed was the number of cars. As the size of the environment does influence the complexity and the time of the simulation, the resulting experiment does not conform strictly to the rules of weak scalability.

The numbers of cores used in consecutive executions were the following: 24, 48, 120, 240, 480, 1200, 2400, 4800, 7200, and 9600. The number of cars was kept at level of 500 per core.



(a) Box-and-whisker plot (b) Cumulative times of step stages

Fig. 7: Simulation time-steps with number of cars growing with the number of workers.

Fig. 7 shows the measured execution times, obtained in the same way as in Fig. 5. As the Fig. 5a shows, for small numbers of workers the size of the environment to process influences the time significantly. However, for 240 workers and above, the execution time levels out, which is a desired behaviour in this test. Adding new tasks and new resources allows preserving the execution time constant.

Once again it is possible to compare to the results from previous architecture, presented in Fig. 2. In this case, the number of cars per core was twice the value in the described experiment. The median execution time of a single time step for the largest number of workers (2400) was ca. 250 ms. The expected average execution time for the number of cars in this experiment calculated from this result would be ca. 125 ms. Once again, the results obtained using the new architecture are far better, around 8 times faster execution is observed.

6 Design Guidelines for Scalable Spacial Simulations

In this section we summarize the conclusions from our previous [8] work and the architecture patterns of the current work, which tends towards more general design guidelines for scalable spacial simulation. As authors of the work [1] observed, to be able to achieve super-scalability in the parallelization of algorithms, they should be scale invariant by design. This applies to all aspects of the task size, including the amount of calculations, but also the volume of possessed and process data, the number and the volume of messages exchanged with other tasks and the number of connections. Besides these requirements concerning the computation itself, serious problems can occur during initialization and results collecting phases.

Although the initial size of tasks can be even in spacial simulations, preserving the evenness during simulation is non-trivial. Simulated entities are transferred between workers, altering the balance and causing the need of waiting for most loaded stragglers. Dynamic load balancing is a complex challenge in this area.

Memory-related issues exhibit their significance only when large scenarios are considered. They are often neglected at the beginning of parallel program development, which magnifies their impact in large scale. Therefore, at the design stage, all data structures should be divided into three categories: task-independent, static task-dependent and dynamically growing task-dependent. All task-dependent structures have to be split between workers in order to support starting large simulations and preserve the scale-invariant assumption. Dynamically growing structures (e.g. simulation results) have to be explicitly managed and serialized when necessary. Failing to address this issue causes run-time problems, which are far more complex to identify.

Probably the most important element of a salable design is the communication architecture and synchronization schema. Centralized communication cannot be used during the simulation process – there are no exceptions here. If it is necessary to distribute messages across all the workers then proper communication topology should be introduced and messages should be aggregated wherever possible. In our case worker managers on each cluster node are responsible for aggregation of incoming and outgoing communication with the server. It is very important to ensure only a limited number of connections per worker, but also per all other elements of the system – the managers tend to become bottlenecks also. When proper simulation occurs, it is a key to ensure direct communication between workers. The algorithm has to ensure limited number of this communication type by proper division of the task.

Another important condition of scalable spacial simulation is the proper synchronization approach. In every case, central synchronization should be removed and implicit synchronization strategy should be used instead. After computing a simulation step, each worker sends messages to its neighbours and expects them to do the same. After that, the computation continues.

An additional issue, which currently prevents us from following all the presented guidelines, is the feature of the simulation algorithm, which requires access to the whole model. In the considered traffic simulation new cars need a path, which is calculated using the whole map. In general, such situations violate the scale-invariant condition and should be disallowed. In our case we could require require all the paths in advance. Otherwise we propose to extract the operations, which require access to the whole model into a single entity located at each computing node to reduce the memory demand.

The described guidelines concern the execution of the simulation task itself, which is not the only source of scalability problems. The phases of initialization and finalization can also burden the efficiency or prevent the simulation from running at all. A few typical challenges to point out here are: model division, distribution and loading and results collecting. Distributing the model data and collecting the results can become bottlenecks, when implemented as one-to-many communication. Our experience show, that far better results are achieved by using a shared file system for both these tasks, while the communication is used only for configuring them.

The model division algorithm has to split the model into fragments for specified number of nodes and cores. Existing algorithms are typically sequential, which is a huge waste of resources, waiting idle for the simulation task. Research on parallel splitting methods continues, however, a simpler approach can be adopted in the meantime. By using a faster and less accurate stochastic algorithm, running in many instances on all available nodes, we can often compute a better solution in shorter time. One should also keep in mind that event a perfect initial division does not solve the imbalance problem.

7 Conclusions and Further Work

The presented distributed architecture, together with proper data handling mechanisms allowed simulating great-scale real-life scenario with continuous space and motion model. The use of HPC-grade hardware provides significant performance improvements, but is also necessary due to memory required by the simulation model. The conclusions from the presented work formulate a set of guidelines for achieving super-scalability in spacial simulations.

The presented results clearly show, that there is still space for further improvements in the area of load balancing. Uneven distribution of work between computing nodes results in significant waste of computational power. This problem poses significant challenge for future research because of high dynamics of load changes in this particular problem.

Acknowledgments

The research presented in this paper was funded by the National Science Centre, Poland, under the grant no. 2019/35/O/ST6/01806. The research was supported by PL-Grid Infrastructure.

References

1. Engelmann, C., Geist, A.: Super-scalable algorithms for computing on 100,000 processors. In: Proceedings of the 5th International Conference on Computational Science - Volume Part I. p. 313–321. ICCS'05, Springer-Verlag, Berlin, Heidelberg (2005). https://doi.org/10.1007/11428831_39
2. Kanezashi, H., Suzumura, T.: Performance optimization for agent-based traffic simulation by dynamic agent assignment. In: Proc. of 2015 Winter Simulation Conference. pp. 757–766. WSC '15, IEEE Press, Piscataway, NJ, USA (2015)
3. Kesting, A., Treiber, M., Helbing, D.: General lane-changing model MOBIL for car-following models. J. of Transportation Research Board **1999**(1), 86–94 (2007)
4. Kesting, A., Treiber, M., Helbing, D.: Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. Trans. of Royal Society of London A **368**(1928), 4585–4605 (2010)
5. Khunayn, E.B., Karunasekera, S., Xie, H., Ramamohanarao, K.: Straggler mitigation for distributed behavioral simulation. In: 2017 IEEE 37th Int. Conf. on Distributed Computing Systems (ICDCS). pp. 2638–2641. IEEE (2017)

6. Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R., Lavanya, R.: A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In: Proc. 2005 IEEE Intelligent Transportation Systems, 2005. pp. 813–818 (2005)
7. Nagel, K., Schleicher, A.: Microscopic traffic modeling on parallel high performance computers. Parallel Computing **20**(1), 125 – 146 (1994)
8. Najdek, M., Xie, H., Turek, W.: Scaling simulation of continuous urban traffic model for high performance computing system. In: International Conference on Computational Science. pp. 256–263. Springer (2021)
9. O'Cearbhaill, E.A., O'Mahony, M.: Parallel implementation of a transportation network model. Journal of Parallel and Distributed Computing **65**(1), 1–14 (2005)
10. Paciorek, M., Turek, W.: Agent-based modeling of social phenomena for high performance distributed simulations. In: International Conference on Computational Science. pp. 412–425. Springer (2021)
11. Ramamohanarao, K., Xie, H., Kulik, L., Karunasekera, S., Tanin, E., Zhang, R., Khunayn, E.B.: SMARTS: Scalable microscopic adaptive road traffic simulator. ACM Trans. on Intelligent Systems and Technology (TIST) **8**(2), 1–22 (2016)
12. Rickert, M., Nagel, K.: Dynamic traffic assignment on parallel computers in transims. Future Generation Computer Systems **17**(5), 637 – 648 (2001)
13. Toscano, L., D'Angelo, G., Marzolla, M.: Parallel discrete event simulation with erlang. In: Proceedings of the 1st ACM SIGPLAN Workshop on Functional High-performance Computing. pp. 83–92. FHPC '12, ACM, New York, NY, USA (2012)
14. Turek, W.: Erlang-based desynchronized urban traffic simulation for high-performance computing systems. Future Generation Computer Systems **79**, 645–652 (2018)
15. Turek, W., Siwik, L., Byrski, A.: Leveraging rapid simulation and analysis of large urban road systems on HPC. Transportation Research Part C: Emerging Technologies **87**, 46–57 (2018)
16. Xu, Y., Cai, W., Aydt, H., Lees, M., Zehe, D.: 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. pp. 259–269. SIGSIM PADS '15, ACM, New York, NY, USA (2015)