


Performance improvements for large scale traffic simulation in MATSim

Conference Paper**Author(s):**

Waraich, Rashid A.; Charypar, David; Balmer, Michael; Axhausen, Kay W. 

Publication date:

2009-09

Permanent link:

<https://doi.org/10.3929/ethz-a-005864320>

Rights / license:

In Copyright - Non-Commercial Use Permitted

Originally published in:

Arbeitsberichte Verkehrs- und Raumplanung 565

Performance Improvements for Large Scale Traffic Simulation in MATSim

Date of submission: 2009-aug-1

Rashid A. Waraich

Institute for Transport Planning and Systems (IVT), ETH Zurich, CH-8093 Zurich

phone: +41-44-633 32 79

fax: +41-44-633 10 57

waraich@ivt.baug.ethz.ch

David Charypar

Institute for Transport Planning and Systems (IVT), ETH Zurich, CH-8093 Zurich

phone: +41-44-633 35 62

fax: +41-44-633 10 57

charypar@ivt.baug.ethz.ch

Michael Balmer

Institute for Transport Planning and Systems (IVT), ETH Zurich, CH-8093 Zurich

phone: +41-44-633 27 80

fax: +41-44-633 10 57

balmer@ivt.baug.ethz.ch

Kay W. Axhausen

Institute for Transport Planning and Systems (IVT), ETH Zurich, CH-8093 Zurich

phone: +41-44-633 39 43

fax: +41-44-633 10 57

axhausen@ivt.baug.ethz.ch

Words: 5752 + (6 Figures) = 7252

ABSTRACT

Multi-Agent transport simulation models, e.g. MATSim have proven to be suitable for modeling microscopic demand for large scale scenarios based on planning networks. In the recent years survey methods are using technologies which provides mobility information with a much higher spatial resolution (e.g. GPS tracking). Therefore, the need to model travel demand on detailed navigation networks rises, which slows down simulation speed significantly. This paper presents methods to increase the performance of the micro simulation model of MATSim using event driven concepts as well as a parallel implementation. The performance experiments with navigation networks of Switzerland containing up to one million roads and 7.3 million agents clearly show that large-scale, multi-agent micro-simulation can also be applied on high resolution networks.

INTRODUCTION

Traffic simulations can be performed at different levels of detail. One common technique is to model traffic as flows consisting of accumulated number of cars. A different approach is to model each individual vehicle. Agent-based modeling (representing each person as a simulation agent) combined with traffic flow micro-simulation allows for tracking persons dynamically in time. Of course agent-based modeling is more expensive in terms of computation power than aggregated models.

The open source Multi-Agent Transport Simulation Toolkit (MATSim, 1) was designed from the beginning to meet the challenges of simulating large scenarios to optimize travel demand. As shown in Figure 1, the demand optimization is an evolutionary process (2). This means the iterative loop consisting of *execution*, *scoring* and *replanning* has to be executed many times before the optimized demand can be analyzed. This paper focuses on the performance related to the execution part. The traffic simulation (execution) generates *events* for each action in the simulation, e.g. enter road, leave road or arrival at destination. These events are then processed as the simulation is running, e.g. for generating statistical data on the simulation and for the scoring module.

MATSim is currently facing a major challenge, as more and more applications require simulation on high resolution networks. A few example applications are listed below:

- **Global Positioning System (GPS) Surveys:** Travel studies based on GPS data require high resolution networks to map the GPS tracking data properly to the network (3, 4, 5).
- **Intelligent Transportation Systems (ITS):** Measurements to improve traffic flow, such as traffic lights at intersections have often impact on a larger area (e.g. see 6). In order to model the impact of traffic lights and related ITS phenomena, simulation on high resolution networks is required.
- **Commercial Applications:** Companies owning street bill-boards need to know the traffic which passes a specific one in order to determine the appropriate price tag for it. Furthermore the properties of the people driving along these bill-boards are important (e.g. where they live).

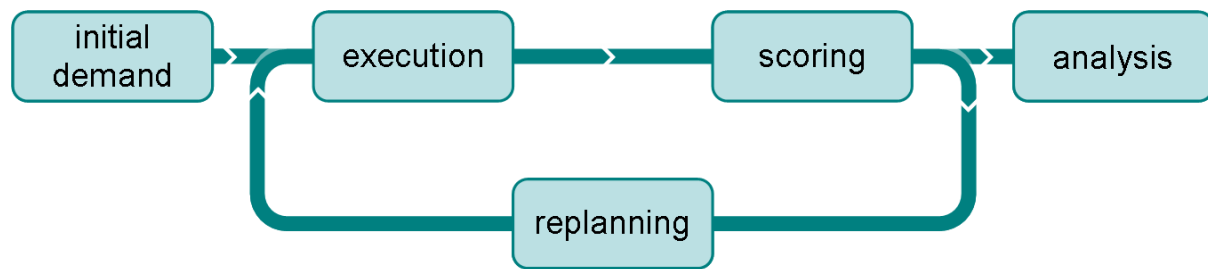
The difference in size of the networks is substantial. Planning networks often used with MATSim before contained less than 100 thousand links. The new navigation networks are of much higher resolution and contains more than a million links. This increase puts high pressure on both the micro-simulation and event processing in MATSim. This paper introduces several ways to improve the performance of MATSim and any other event driven micro-simulation in this regard together with experiments and results. In the next section related work is presented.

RELATED WORK

MATSim has always focused on large-scale traffic simulation scenarios. In the next subsection the traffic simulation model underlying the different simulation implementations is discussed.

The Traffic Simulation Model

The general traffic simulation approach used in MATSim is based on queues. In this approach links are the active element, which move around cars. Each link contains a queue which stores the entry time of each car. Adjacent links collaborate with each other to assure that the different

FIGURE 1 Co-evolutionary simulation process of MATSim

traffic parameters and elements are simulated correctly. For example link capacity, free speed travel time, intersection precedence and space available on the next link are parameters which are taken into account by the simulation.

In the following subsections the different implementations of this approach are discussed.

QueueSim

The first micro-simulation for MATSim (called *QueueSim*) was based on a fixed-increment time advance model (7) and developed in C++ programming language. The vehicles were moved along in fixed time steps of one second. Although the model is quite flexible, for larger simulations it is too slow because of the fixed simulation time step. A parallel version of QueueSim was implemented leading to significant speed up (8).

DEQSim

A major performance breakthrough was achieved with a new micro-simulation, called *Deterministic Event-Driven Queue-Based Traffic Flow Micro-Simulation* (DEQSim, 9), also implemented in C++. Instead of performing the simulation along fixed time steps, an event based model is used performing only discrete actions which are relevant to the model (i.e. entering and leaving roads). Furthermore, DEQSim has been parallelized making it one of the fastest large scale transport micro-simulations available (10).

JQueueSim

To improve maintainability of the code, MATSim was re-implemented in Java including the single CPU version of QueueSim (here called *JQueueSim*). Furthermore, the performance of JQueueSim has been improved significantly over the last few years while the main model (time step based approach) still remains the same.

Graphical Processing Units

Graphical Processing Units (GPUs) on computer graphic cards perform many more operations in the same amount of time as CPUs on computer boards. A first successful implementation of QueueSim on GPUs rendered a speedup of 67 times (11) compared to JQueueSim. The main

drawback of GPUs is, that the interface between the graphic card and the rest of MATSim modules poses a bottleneck. Furthermore, current GPUs have an limited amount of memory. For example the traffic simulation of Switzerland with 7.3 Million agents requires around 60 Gi-gabyte of memory, depending on different settings. Graphic cards today have often less than 1 GB of memory.

Re-implementing DEQSim in Java

The implementation of MATSim modules in Java means a major speed disadvantage for DEQSim: As DEQSim is implemented in C++ the communication of DEQSim with the other MATSim modules is bridged by a slow file input/output (I/O) interface. Furthermore, extensions in DEQSim are difficult to maintain because of the different programming languages. Even though, DEQSim is still faster than JQueueSim, JQueueSim has become the de facto standard micro-simulation for MATSim.

In order to benefit from the speed superiority of DEQSim over QueueSim, a redesign and re-implementation of DEQSim in Java is presented in the next section (called *JDEQSim*). Furthermore, the interface between the micro-simulation and MATSim (called *event handling*) has been redesigned and improved. At last, first preliminary results of a parallel version of JDEQSim are shown.

PERFORMANCE IMPROVEMENTS

Implementation of JDEQSim

Re-implementing DEQSim in Java provided the opportunity to restructure and redesign the code. The C++ DEQSim code was only taken as a specification requirement and fully ignored for the implementation of JDEQSim. The initial design of JDEQSim was influenced by OM-Net++ (12), which is a modular and open-architecture discrete event communication network simulator. The JDEQSim implementation consists of the following three parts:

- **Simulation Units** Vehicles and links are the basic building blocks of the traffic simulation.
- **Messages** *Simulation units* communicate with each other by exchanging different kinds of *messages*. Each message contains a time stamp, e.g. when a vehicle is allowed to enter the next link or when a car should start a leg. Each newly created message is sent to the *scheduler*.
- **Scheduler** The *scheduler* contains a message priority queue, which is ordered after message time and message type. Each received message is put into this queue. In the beginning of the simulation, the queue is initialized: The first leg of each agent is scheduled in the queue. Thereafter the scheduler fetches the first message and executes it. Often this produces a new message, which is put into the queue. The scheduler processes always only the first message, until all messages have been processed, which terminates the micro-simulation.

To a certain extent many elements used in JDEQSim are similar to concepts presented by Axhausen (13).

Gaps in a Queue

From the beginning DEQSim had an additional feature making the model more realistic, which is still missing in JQueueSim: Gaps travelling backwards as a queue is dissolving (10). This makes the traffic model more realistic as when the front car in a queue starts driving, it leaves behind a gap which travels with a constant speed backwards. Therefore cars behind in the queue have to wait until such a gap reaches them, before they can start driving.

Prevention of Gridlock

At intersections, during the simulation a gridlock can happen, so that vehicles wait for each other forever. In the real world when such a situation arises, where it is not clear who has right of way, humans interact to resolve the ambiguity. In the micro-simulation this is resolved in the following way: Whenever a situation is detected where a potential gridlock could happen, cars continue moving to prevent the gridlock (14). This is achieved by temporary allowing more cars to enter a link than its capacity. This is not problematic in MATSim as agent plans where this situation occurs are penalized by the scoring module, resulting in fewer gridlocks in the next iteration.

Transportation Modes

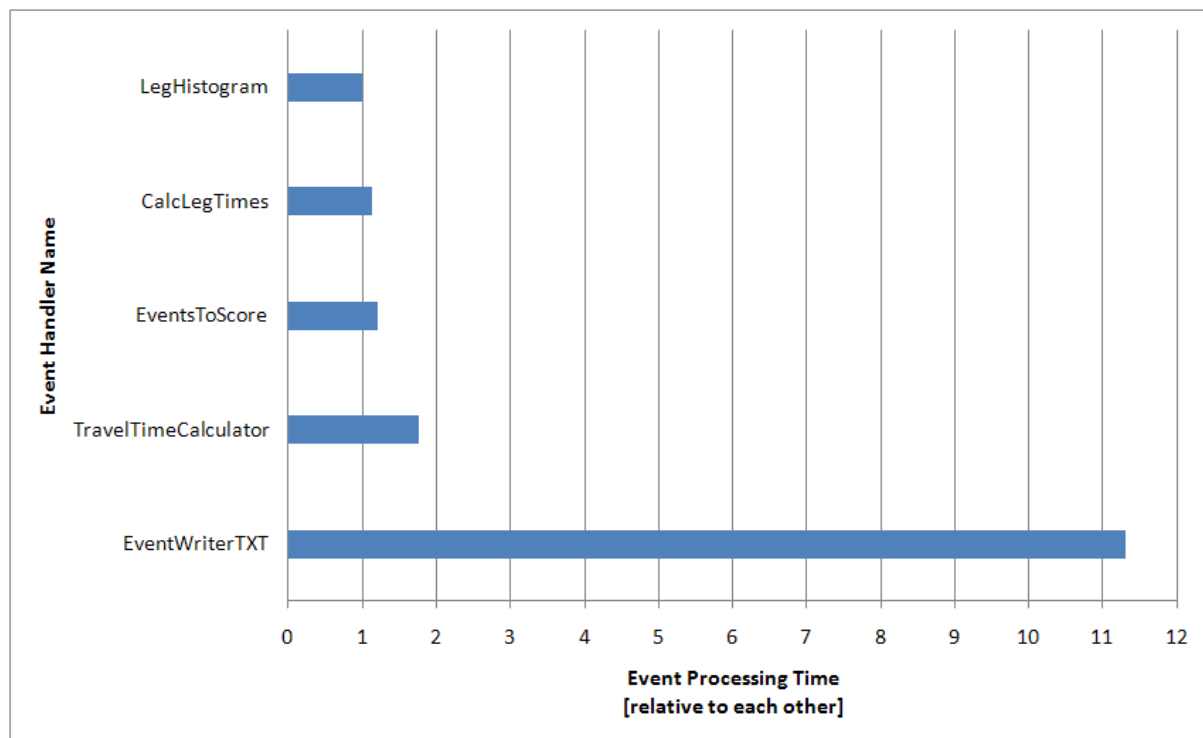
JQueueSim allows simulation of multiple transportation modes which is not implemented in DEQSim. For example an agent can drive to work and at lunch time walk by foot to a restaurant. Or a person might take a bike to ride to the bus stop and travel further by bus. JDEQSim has profited from these developments and has incorporated all these features into the simulation model from the beginning.

Parallel Event Handling

The communication between the micro-simulation and the rest of MATSim happens via *events*. Events have different types such as *enter/leave link* or *start/end activity*. Furthermore, the event also contains information on the location and time where the event occurred and which agent was involved. These events can be processed by various modules of MATSim. This is called *event handling*. For example by using event handling, travel time statistics can be calculated for the simulation or events can be written out to a file for later processing. Also modules such as the scoring module make use of event handling.

During the implementation of JDEQSim and its parallelization it was observed that event handling is executed on the same CPU as the micro-simulation. Running event handling in parallel to the micro-simulation has two advantages. Firstly, the micro-simulation can run faster, because if event handling contains a file writer handler (as by default), this will slow down the micro-simulation. Secondly when applying MATSim on certain scenarios, custom handlers are needed which may require lots of computation time. If multiple cores are available on a machine (as mostly the case nowadays), event handling can be further distributed among them.

At the moment five event handlers are present in MATSim by default. Figure 2 shows the relative time proportions of these five handlers to each other. As expected the event writer (*EventWriterTXT*) is the most heavy weight event handler in terms of processing time.

FIGURE 2 The execution time of different default event handlers relative to each other

Implementation of Parallel Event Handling

A common way in Java to use multiple CPUs (or cores) is to make use of *threads*. For parallel event handling the user can specify how many threads should be dedicated to parallel event handling in addition to the micro-simulation. The current implementation applies a round robin approach (15) to assign event handlers to threads. This means it tries to assign the same number of event handlers to each thread. It is obvious from Figure 2 that this approach is suboptimal, because it would be best to put the event handler writing out events on a separate thread and the others on a second one. Furthermore, it may seem that even by applying this improved method, writing events to the hard disk is so slow so that using more than two threads would not improve performance.

There are several reasons, why the presented approach was chosen and why more than two threads can actually make the simulation faster. First of all, writing out events to a file is not part of the communication interface between JDEQSim and MATSim (which is different in DEQSim). This means, if MATSim is running for 60 iterations only each 10th iteration needs to be written out. By doing so, full use of parallelization can be made during 9 of the 10 iterations. One reason for using a simple implementation for parallel event handling is that the existing framework did not need to be changed and the new implementation could just wrap the existing implementation into threads.

Parallelization of JDEQSim

In Charypar *et al.* (10) the parallelization of DEQSim is described. This is done by partitioning the traffic network into several pieces, which are assigned to a separate CPU of a shared mem-

ory machine. The Message Passing Interface (MPI, 16) is used for communication between CPUs. When an agent travels from the network area assigned to one CPU to a different one, MPI is used for passing the agent data between CPUs. This operation includes periodically synchronizing the state of links at the border of each network partition. In Java, threads (17) are commonly used as a basis for parallel programming. In order to pass data between two threads the *synchronized* keyword is used. Unlike MPI, the *synchronized* keyword does not allow to explicitly specify which data should be transferred between which CPUs.

The advantage of the Java *synchronized* keyword is, that no explicit data structures have to be built for transferring data between threads. But in the context of parallelizing JDEQSim, this is also a major disadvantage: Whereas MPI allows to explicitly specify, which data to transfer, it is not always obvious what data will be exchanged due to a *synchronized* statement in Java. Furthermore usage of the *synchronized* keyword is a quite expensive operation in Java in terms of computation time, because often more data is being synchronized between threads than actually needed. As many elements of data transfer are hidden and handled by the Java Virtual Machine (JVM, 18), the programmer has little control over them.

Before describing the successful implementation of the parallelization of JDEQSim two implementation ideas which failed are described (*Single Scheduler - Multiple Message Executors* and *Multiple Scheduler - Multiple Message Executors*). This might help to better follow why certain elements are present in the final adapted approach (*Time Delta between Threads*).

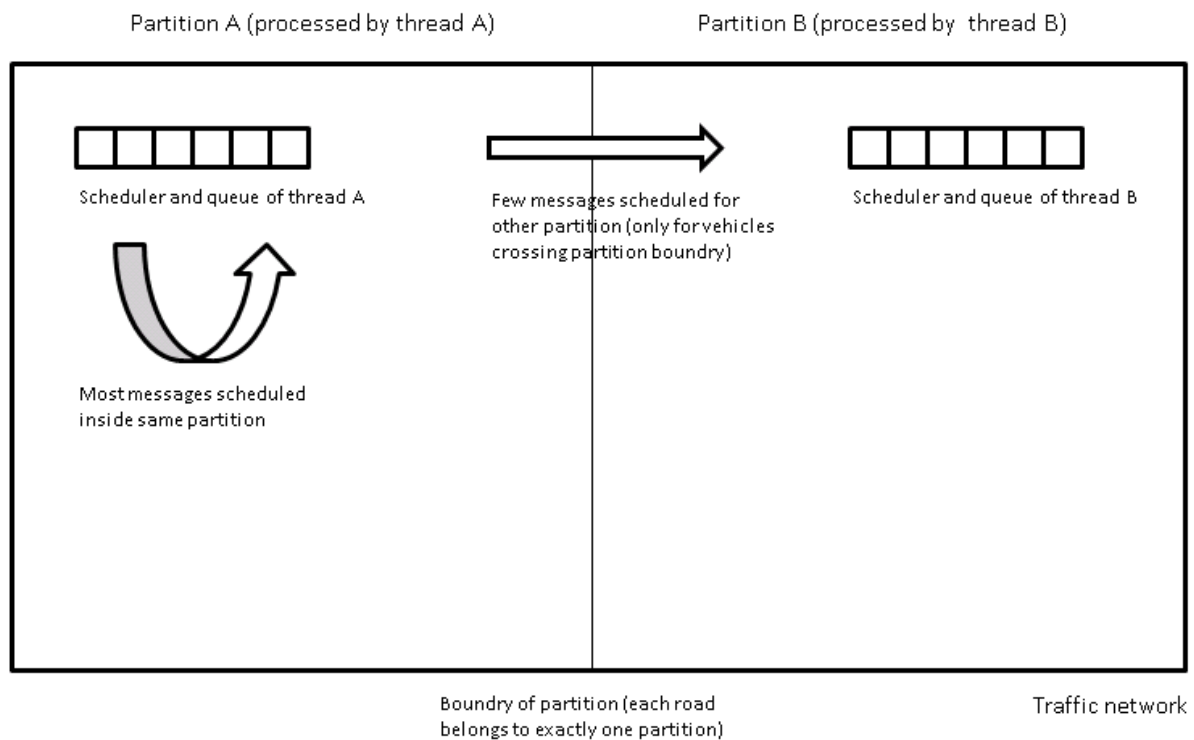
Single Scheduler - Multiple Message Executors

In Java each thread can access all data within the JVM. One possible approach might be to just use one scheduler within the JVM, where all the messages are queued. Multiple *message executor* threads can be used to process messages (e.g. moving vehicles on links). Of course appropriate synchronization between threads is needed for link and vehicle objects. Although this approach might seem promising, the scheduler with the message queue poses a bottleneck: For insertion of messages into the scheduler queue the *synchronized* keyword has to be used. Event adding per thread message buffers into the scheduler does not help much to resolve this problem.

Multiple Scheduler - Multiple Message Executors

Another approach might be to partition the network into pieces similar to what is done in the parallelization of DEQSim. In this case each partition is assigned to a separate message executor thread and has its own scheduler. The different threads need to synchronize periodically for two reasons. First of all for consistency of data, vehicles which enter a new partition of the network must be synchronized. Secondly border links must be synchronized periodically at predefined intervals to ensure that one thread does not advance the simulation too much. This synchronization interval is determined by the travel time needed to travel border links (link adjacent to a link in a different partition). This is almost identical to how the parallelization of DEQSim is performed.

Unfortunately this approach does not perform well because of the periodical waiting on other threads. There are several higher level constructs in Java to wait on other threads, such as *wait* and *notify* or *barriers* (17). Furthermore, the most primitive approach is to do busy waiting (do some dummy computations until the condition is met). All of these approaches did not render satisfactory results for the problem at hand.

FIGURE 3 Parallelization of JDEQSim

Time Delta between Threads

The Method In Figure 3 the successful parallelization approach is depicted. The prototype has been implemented for two CPUs. The traffic network is thereby divided in two parts in vertical direction, so that about the same number of events will occur during the day in each part of the network. This number can be estimated from the journey plans of the agents which are given as input to the micro-simulation by MATSim. A (message executor) thread is assigned to each of these partitions, which has a separate scheduler. Border links have been defined, which are adjacent to links in the other partition. The vehicles objects which pass over border links are synchronized between the threads. No synchronization of links is required as messages concerning a certain link are only processed by the thread to which the link belongs.

Scheduling of messages is done in the following way: As long as the link specified in the message belongs to the same thread it was scheduled by, no synchronization is required. The message can simply be put into the corresponding message queue. But if one thread schedules a message for the other thread, because a vehicle is passing between partitions, synchronized access to the queue object is required. This is further optimized by using a buffer inside the queue. As the proportion of vehicles moving between network partitions is much smaller than those travelling inside the same network partition, relatively few synchronization statements between threads are required.

One problem remains to be handled that is, one thread should not advance too much in time. At this point the question arises, what does correctness of a parallel simulation mean? The parallel simulation does not have to generate the same sequence of events as a sequential simulation in order to be correct. As often events have identical time stamps, the simulation

can arbitrary select, which of these events to process first. For example, if we have two agents going to work at exactly 6am in the morning, it is up to the simulation to decide, which event should be processed first. If both of them might want to enter the same link then one gets to go first. This decision by the micro-simulation changes the event time for the involved agents but does not have any significant effect on the overall simulation.

So, for a parallel simulation to be considered correct, it should not change the aggregate traffic properties of a sequential simulation on average. Keeping this fundamental property in mind, it is possible to define a small time delta (for example 10 seconds), which a thread is allowed to advance more than the other threads. This might produce a very small number of cars, which get lucky at border link intersections and thereby advance one place in the queue of the next link. But these low probabilistic events should not change the overall properties of the simulation significantly. It has been verified that the number of events and their logical sequence is not affected by adapting this heuristic. Nevertheless, this phenomenon needs to be investigated further.

By allowing a small time delta between different threads, there is also the advantage, that all messages that are within this delta, can be fetched at once and processed by the message executor thread. Therefore the number of synchronization statements used within the scheduler are reduced.

Open Issues and Ongoing Work

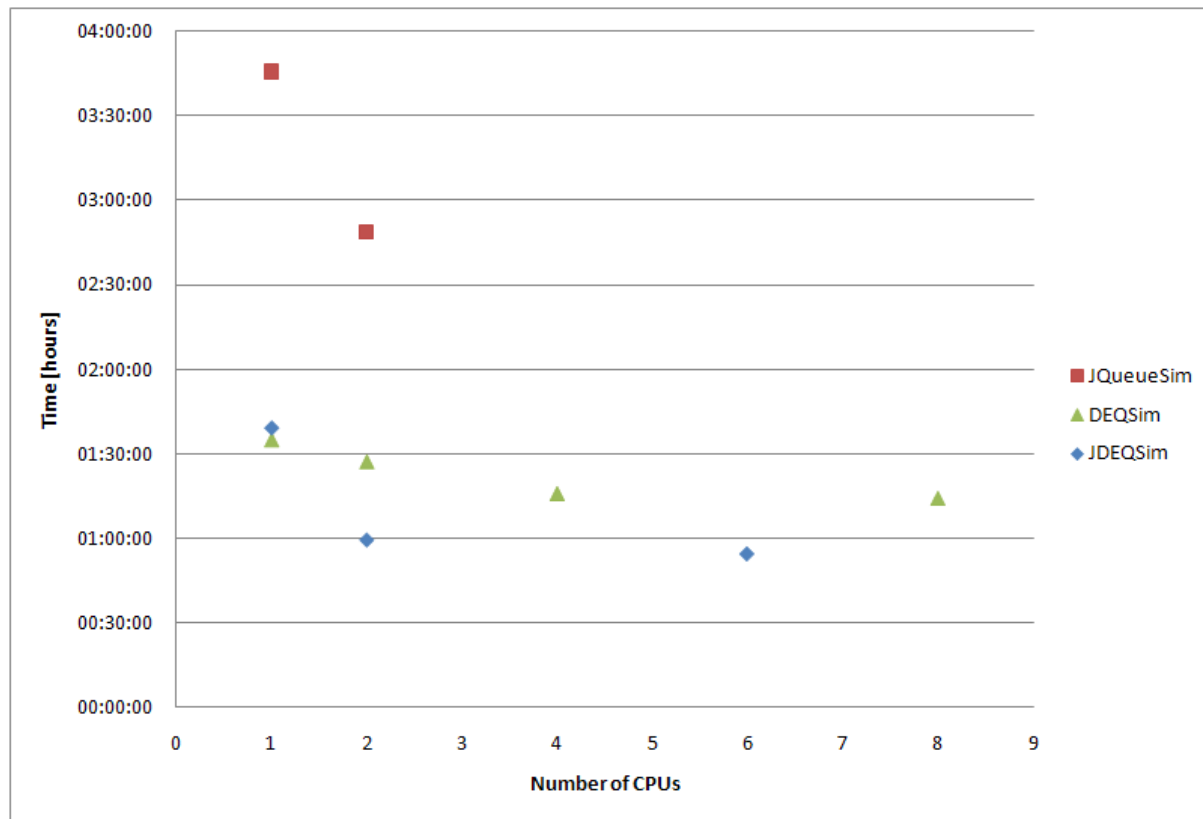
Load Balancing The agents' plans help estimating the number of events that will occur in a network partition. But the traffic in each area changes over the day. Defining new borders during the simulation of the day might help in this regard, but its possibility and impact needs to be further investigated.

The different message types require different amount of processing time. This could also be taken into account in future. For the experiments presented in this paper, the partitioning of the network was calibrated manually. This should be automated in future.

Time Synchronization between Threads Although the approach regarding synchronization of time between threads seems intuitive, experiments are required to validate that this approach does not have any significant effect on the relevant parts of the model. For example, it needs to be investigated, what portion of vehicles have advanced into the other partition before the time they would have entered it, if a smaller time delta was chosen.

Generalization of the Approach to N CPUs The prototype has been implemented only for two CPUs at the moment, but it can be generalized to N CPUs. This can be done by splitting the link network vertically in N partitions so that each partition contains the same number of events. Then each of these partitions is assigned to a separate thread.

Adaptation of Parallel Event Handling Required The events processed by event handling must have an ascending time stamp. When the events are being produced in parallel by JDEQSim, buffering and sorting of the events is required before they can be sent for event processing. This requires the implementation of an efficient input buffer at the parallel event handler.

FIGURE 4 The computation time with different micro-simulation settings for a single iteration

EXPERIMENTS AND RESULTS

Overall Speedups

Experiments to compare JQueueSim, DEQsim and JDEQSim were performed on the NAVTEQ road network (19) for Switzerland (around 882K links). A population sample of the people surrounding the city of Zurich which drives cars was used (around 614K agents). The hardware used for this experiment is a Sun Fire X4600 M2, with 16 cores (8 dual core CPUs) and 128GB of memory. For the same setup also the effects of parallel event handling were analyzed. In Figure 4 runtime measurements for JQueueSim, DEQsim and JDEQSim are shown for the micro-simulation including event handling. All JQueueSim and JDEQSim runs with more than one CPU are using parallel event handling. E.g. the JDEQSim run with 2 CPUs is using one CPU for the micro-simulation and one for parallel event handling.

Parallel Event Handling

The Figure 4 clearly shows the significant impact of parallel event handling on both JQueueSim and JDEQsim. Currently DEQSim cannot use parallel event handling, because the integration is not implemented. But the impact of parallel event handling with DEQSim would not be that significant: By default DEQSim runs have already turned off the event handler which would write events to the disk again (because this is already part of the interface between DEQSim and MATSim).

The figure also shows, that for the default event handler setting in MATSim using more than one thread for event handling gives only a small additional speedup.

Amdahl's Law

Amdahl's law (20) describes the maximum achievable speedup of a parallel program. It says, that if a certain portion of a program cannot be parallelized (e.g. because of writing out events to files), then the maximum achievable speedup is limited even with unbounded computation power. To give an example of Amdahl's law: If 5% of a program cannot be parallelized, then it is not possible to achieve a speedup of more than 20 (even by having one million CPUs).

The figure shows the implication of Amdahl's law for DEQSim within the MATSim context. Because of the I/O overhead of the communication between the micro-simulation and MATSim a speedup of even two seems impossible. This means more than 50% of the micro-simulation consists of parts, which have to be executed sequentially.

Relative Micro-Simulation Speeds

In order to compare the speed of JDEQSim and JQueueSim micro-simulations only the case where parallel event handling is turned on is relevant (this takes away the fraction of event handling). This shows that JDEQSim is around 3 times faster than JQueueSim for the given scenario. The JDEQSim micro-simulation is slower than the DEQSim simulation (without considering the I/O overheads), which can depend on many factors among others that JDEQSim is running within a JVM, whereas DEQSim is compiled to native code.

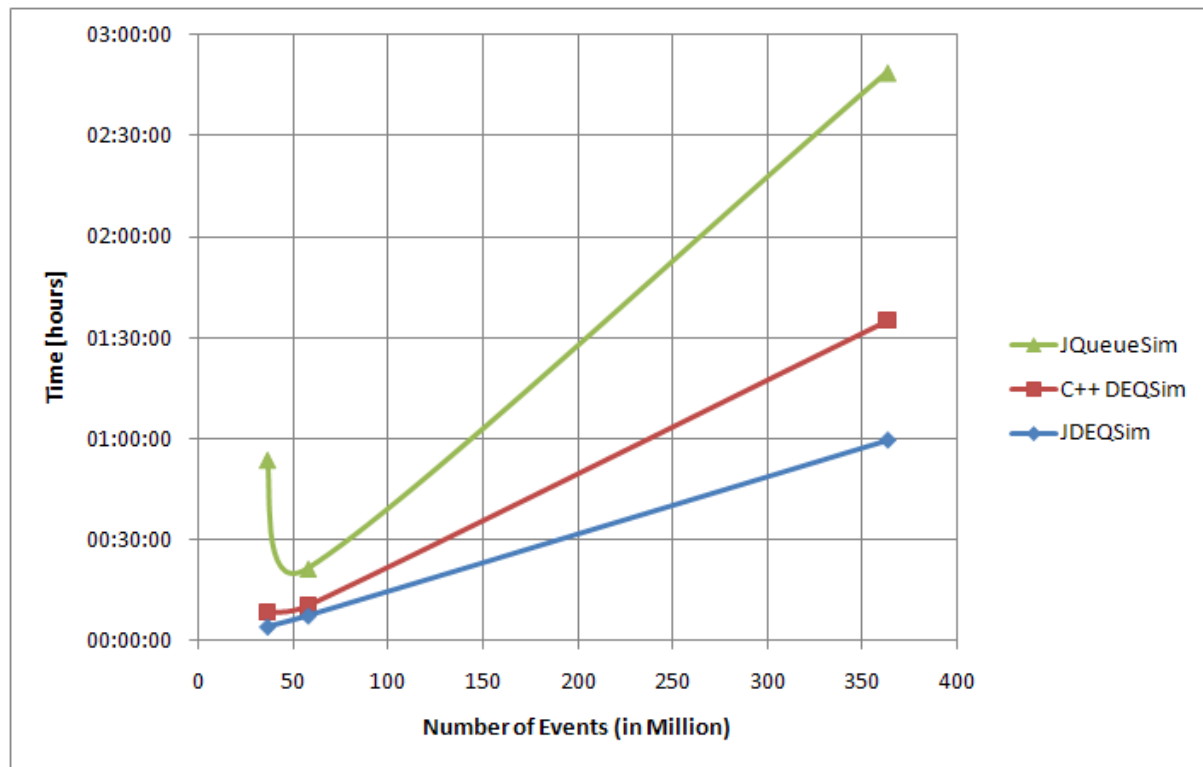
Most Efficient Configuration

The figure clearly shows, how to make most efficient use of CPUs: Running JDEQSim with one parallel event handling thread. As the machine used has around 128GB RAMs and the run above uses less than 15GB of RAMs, up to 8 JDEQSim runs can run in parallel. Furthermore, as most new desktop computer today have at least two cores, the same solution can provide significant speed up as till now only JQueueSim was the option for them.

Influence of Network Size

In the previous experiment JDEQSim was three times faster than JQueueSim. But this cannot be generalized, because if the network is congested, then JDEQSim can be tens of times faster than JQueueSim, because the run time of JQueueSim is directly correlated to the duration of the simulated period, which is not the case with JDEQSim. Furthermore, the speed of JQueueSim also varies with the ratio between network size and population (number of agents), which is highlighted in this experiment. In this experiment all micro-simulations are running using one CPU. Furthermore parallel event handling is turned using a single thread. The network capacity is chosen in such a way that no congestion should happen. The three scenarios which are considered in Figure 5 are:

- Scenario A: Network with 882K links and 61K agents (36M events)
- Scenario B: Network with 61K links and 616K agents (58M events)
- Scenario C: Network with 882K links and 614K agents (363M events)

FIGURE 5 The influence of network size on the different micro-simulations

This experiment clearly shows, that C++ DEQSim and JDEQSim scale linearly with the number of events. Only in scenario A for DEQSim the I/O overhead of loading the network is immense compared to the actual simulation time.

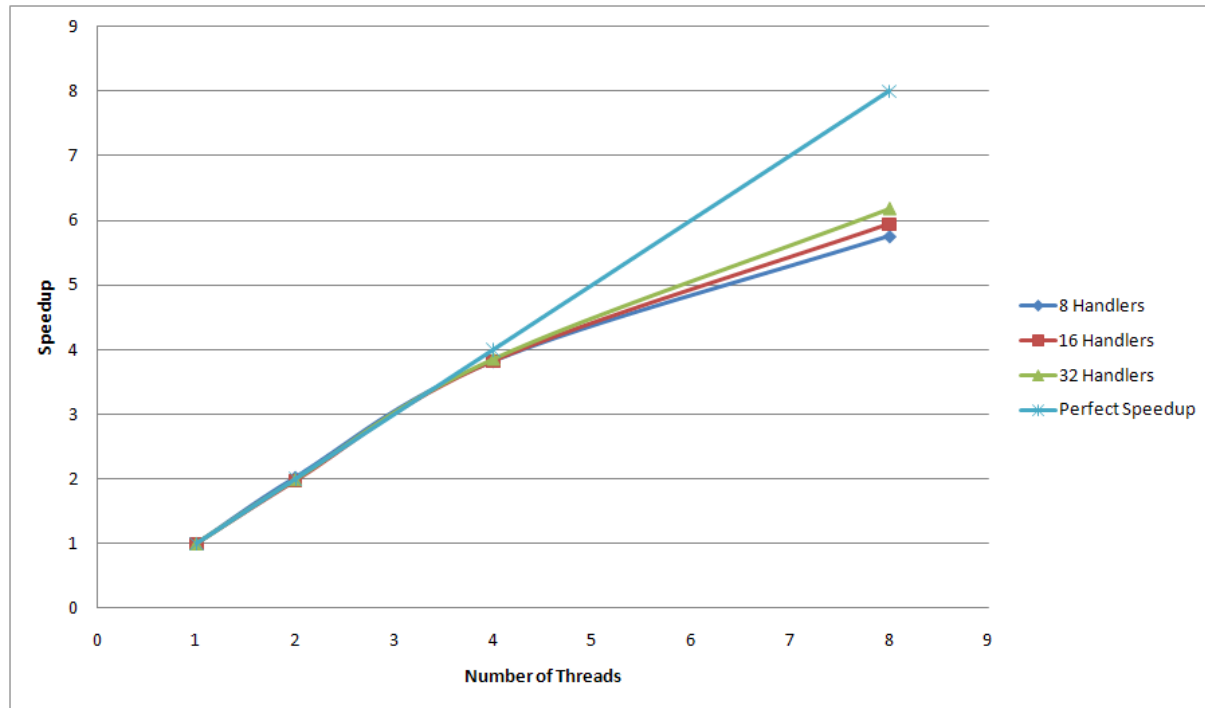
Scenario A and B have the same magnitude of number of events. But the ratio between network size and population size is rather different. Therefore in scenario A JQueueSim performs extremely bad. In fact, JDEQSim is more than twelve times faster than JQueueSim.

Multiple Event Handler - Scalability

To test the scalability of parallel event handling, a dummy handler is used which only performed CPU intensive tasks and did not have any disk I/O. This handler is then added for event handling several times. This experiment was performed for different numbers of threads and handlers to find the speedup, which is shown in Figure 6.

Currently only a small number of event handlers are present by default in MATSim, but many applications are under development and planned in the MATSim community (1), which require additional event handlers. The good news is, that the more handlers are added to parallel event handling, the better the speedup. This is expected, because adding more work to the handler reduces the relative penalty of synchronization between threads.

Parallel event handling scales linearly up to 4 threads. Although parallel event handling only makes little use of Java's synchronized keyword, still the performance drops with 8 CPUs already significantly (speedup only around 6). Similar programs written in C++ with MPI could easily achieve speedups of around 8 in this case (9).

FIGURE 6 Scaling of Parallel Event Handling with the number event handlers and threads

Speedup for Parallel JDEQSim

An experiment with JDEQSim for the whole population of Switzerland (7.3 million agents) was performed on a network with around one million links. The agents travelled using different transportation modes, such as car, bus, bike and by foot. The experiment was done with parallel event handling (single thread) and took around 3 hours and 16 minutes for a single iteration (only micro-simulation and event handling). As MATSim is an iterative process, often more than 60 iterations containing the micro-simulation execution are needed to reach a relaxed state (21). Therefore new ways to accelerate the simulation even further (such as parallelizing JDEQSim) are required in order to simulate even larger scenarios in a reasonable time frame.

As described earlier, a first prototype of the parallelization of JDEQSim for two CPUs has been implemented. As event handling has to be adapted to properly function with parallel JDEQSim only measurements of the micro-simulation were done (with event handling turned off). The experiment consisted of 1.62 million agents residing in the area of Zurich city. The network contained 163K links. This experiment took 29 minutes 40 seconds with JDEQSim, while on parallel JDEQSim (2 CPUs), the experiment only took 18 minutes 37 seconds. This means a speed up of 1.6, which is quite encouraging.

CONCLUSIONS AND FUTURE WORK

In this paper JDEQSim is presented, which accelerates MATSim substantially. Furthermore, JDEQSim can simulate bigger runs with much fewer CPUs than required till now. This makes it suitable both for running several runs on machines with lots of CPUs and to simulate MATSim runs faster than before on machines with fewer CPUs.

As discussed in the section on experiments, the parallel version of DEQSim is limited by Amdahl's law within the MATSim context. On the other hand JDEQSim is missing MPI and has to cope with the JVM threading model and different JVM implementations, which have different properties on different machines. But nevertheless it is expected, that future JVMs should be able to handle locking more efficiently, so that it makes sense to pursue the path of parallelizing JDEQSim further.

This paper only considered the performance gains of the micro-simulation and event handling in MATSim, but there are also other ways to improve the performance of MATSim. For example some replanning modules such as rerouting are relative slow and could be improved. Furthermore, by improving the way replanning is being used one can reduce the number of iterations until the equilibrium is reached and as such the runtime for MATSim overall.

ACKNOWLEDGEMENT

Special thanks to Yu Chen who was the first to try out JDEQSim and gave feedback regarding first bigger runs. Furthermore, thanks to Marcel Rieser who gave technical advice regarding MATSim. Thanks also to Prof. Kai Nagel for various conceptual discussions related to MATSim.

REFERENCES

1. MATSim-T (2008) Multi Agent Transportation Simulation Toolkit, webpage, <http://www.matsim.org>.
2. Holland, J. H. (ed.) (1992) *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, Cambridge.
3. Casas, J. and C. Arce (1999) Trip reporting in household travel diaries: A comparison to GPS-collected data, paper presented at *the 78th Annual Meeting of the Transportation Research Board*, Washington, D.C., Jan, Oliver 1999.
4. Du, J. and L. Aultman-Hall (2007) Increasing the accuracy of trip rate information from passive multi-day GPS travel datasets: Automatic trip end identification issues, *Transportation Research Part A: Policy and Practice*, **41** (3) 220–232.
5. Schüssler, N. and K. W. Axhausen (2009) Processing GPS raw data without additional information, paper presented at *the 88th Annual Meeting of the Transportation Research Board*, Washington, D.C., Jan, Oliver 2009.
6. Balmer, M., A. Horni, K. Meister, F. Ciari, D. Charypar and K. W. Axhausen (2009) Wirkungen der Westumfahrung Zürich: Eine Analyse mit einer Agenten-basierten Mikrosimulation, *Final Report*, Baudirektion Kanton Zurich, IVT, ETH Zurich, Zurich, February 2009, <http://www.ivt.ethz.ch/vpl/publications/reports/ab550.pdf>.
7. Raney, B., N. Cetin, A. Völlmy, M. Vrtic, K. Axhausen and K. Nagel (2003) An agent-based microsimulation model of Swiss travel: First results, *Networks and Spatial Economics*, **3** (1) 23–41.

8. Cetin, N. (2005) Large-scale parallel graph-based simulations, Ph.D. Thesis, ETH Zurich, Zurich.
9. Charypar, D., K. W. Axhausen and K. Nagel (2007) An event-driven queue-based traffic flow microsimulation, *Transportation Research Record*, **2003**, 35–40.
10. Charypar, D., K. W. Axhausen and K. Nagel (2007) An event-driven parallel queue-based microsimulation for large scale traffic scenarios, paper presented at *the 11th World Conference on Transportation Research*, Berkeley, June 2007, <http://www.ivt.ethz.ch/vpl/publications/reports/ab425.pdf>.
11. Strippgen, D. and K. Nagel (2009) Using common graphics hardware for multi-agent traffic simulation with cuda, paper presented at *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 1–8, ICST, Brussels.
12. OMNeT++ (2009) OMNeT++, webpage, <http://www.omnetpp.org>.
13. Axhausen, K. W. (1988) Eine ereignisorientierte Simulation von Aktivitätenketten zur Parkstandswahl, Ph.D. Thesis, University of Karlsruhe, Karlsruhe.
14. Balmer, M. and K. Nagel (2006) Shape morphing of intersection layouts using curb side oriented driver simulation, in J. P. van Leeuwen and H. J. P. Timmermans (eds.) *Innovations in Design & Decision Support Systems in Architecture and Urban Planning*, 167–183, Springer, Eindhoven.
15. HAHNE, E. (1991) Round-Robin scheduling for max-fin fairness in data networks, *IEEE journal on selected areas in communications*, **9** (7) 1024–1039.
16. Snir, M., S. Otto, D. Walker, J. Dongarra and S. Huss-Lederman (1995) *MPI: The complete reference*, MIT Press Cambridge, MA, USA.
17. Lea, D. (1999) *Concurrent Programming in Java.: Design Principles and Patterns*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
18. Lindholm, T. and F. Yellin (1999) *Java virtual machine specification*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
19. NAVTEQ (2009) NAVTEQ, webpage, <http://www.navteq.com>.
20. Amdahl, G. (1967) Validity of the single processor approach to achieving large scale computing capabilities, paper presented at *Spring joint Computer Conference*, 483–485, NY, USA.
21. Balmer, M., M. Rieser, K. Meister, D. Charypar, N. Lefebvre and K. Nagel (2009) MATSim-T: Architecture and simulation times, in A. L. C. Bazzan and F. Klügl (eds.) *Multi-Agent Systems for Traffic and Transportation Engineering*, 57–78, Information Science Reference, Hershey.