An Event-Driven Parallel Queue-Based Microsimulation for Large Scale Traffic Scenarios

2007-01-01

David Charypar

Institute for Transport Planning and Systems (IVT)

ETH Zurich

Switzerland

Phone: +41 44 633 35 62

Fax: +41 44 633 10 57

Email: charypar@ivt.baug.ethz.ch

Kay W. Axhausen

Institute for Transport Planning and Systems (IVT)

ETH Zurich

Switzerland

Phone: +41 44 633 39 43

Fax: +41 44 633 10 57

Email: axhausen@ivt.baug.ethz.ch

Kai Nagel

Transport Systems Planning and Transport Telematics (VSP)

TU-Berlin

Germany

Phone: +49 30 314 23308

Fax: +49 30 314 26269

Email: nagel@vsp.tu-berlin.de

Words: 5424

Tables and Figures: 5

ABSTRACT

Traffic flow microsimulations are interesting for transport planning problems due to their high temporal and spatial resolution. Unfortunately, most of them involve high computational costs making them impractical for running large scale scenarios. We present a parallel microsimulation software that uses queue-based link dynamics together with event processing instead of time-steps. By introducing appropriate load balancing and by minimizing interfaces between processors to reduce communication needs as far as possible, our simulator requires only 87 seconds on 64 processors to simulate a 24 hours test scenario with 7 million simulated person days on a network with 28k links.

INTRODUCTION

Traffic flow simulation is an important problem in transport planning, as it represents the final link between the description of travel demand and the emergence of flow densities, volumes, and travel speeds. In today's practice, traffic flow is most often simulated using aggregated models that are easy to use and well established in the community. However, compared to aggregated models, traffic flow microsimulation has certain clear advantages:

- Very high spatial and temporal resolution. Features like traffic jams and rush hours can be captured accurately.
- Traffic is represented in a natural way; vehicles traveling, roads, and intersections are simulated directly.
- Traffic flow microsimulation can be easily coupled with other microscopic approaches, e.g. agent-based demand modeling.
- Inverse analyses can be carried out to determine where certain cars are coming from and why they can be found at a specific road network location.

However, in many cases traffic flow microsimulations have prohibitively high computational burden especially for very large scale applications with more than one million person days simulated on high resolution networks. This is even true when using parallel machines to solve the computational task. As a result, microsimulations are most often only used for small applications and aggregated methods are used for large scale problems.

This work aims to present a new event-driven queue-based approach that is parallelizable and makes the traffic flow microsimulation of very large-scale

problems feasible on affordable computer hardware. To boost simulation speed and size of the scenario the option exists to execute the software on large parallel machines with processors added as necessary. This also means that our code profits from multi-core processors which are becoming common. We show that the software performance scales nicely with the number of processors for systems with up to 64 processors, which makes it possible to run traffic flow microsimulations 200 to 500 times faster than earlier queue-based approaches on single CPU computers.

RELATED WORK

In this section, we present a selection of previous work related to our microsimulation. There are many different traffic flow simulation approaches. The physically based microsimulations (e.g. AIMSUN (Barceló et al. 1998a, 1998b; AIMSUN web page), MITSSIM (Yang 1997; MITSSIM web page), VISSIM (VISSIM web page)) generally try to capture as many traffic flow phenomena as possible. They simulate car following and lane changing behavior and use a continuous representation of space and constant very small time-steps to simulate cars on the roads. AIMSUM (Barceló et al. 1998b) presented an implementation of parallel traffic simulation using threads. In their shared memory approach, all variables are globally accessible by all processors. They simulated small scale scenarios on 8 CPUs and reached a speed-up of 3.5 compared to the single CPU solution.

A different microscopic, but less physical, simulation approach is represented by cellular automata (e.g. Brilon and Wu 1998), used for example in TRANSIMS (Nagel et al. 1998; Nagel and Rickert 2001; Rickert and Nagel 2001; TRANSIMS web page). Here, cars move through cells that they can occupy. Although we have a

coarser level of detail in cellular automata, features like densities and travel speeds still emerge from the cars' simple direct interactions and are not computed at an aggregated level. The parallel version of TRANSIMS (Nagel and Rickert 2001; Rickert and Nagel 2001) used message passing between processors and was run on 32 CPUs using midsized scenarios. Although the run times were about 10 times faster than real time, ethernet latency problems and speed handicaps due to the use of cellular automata were reported. Using the same parallel concepts as in TRANSIMS, the queue-based model presented in Cetin (2005, 2003) achieved a speed-up of 32 using 64 CPUs.

In mesoscopic models (e.g. METROPOLIS (Marchal 2001, De Palma and Marchal 2002), DynaMIT (Ben-Akiva et al. 1998, DynaMIT web page), DYNASMART (Chang et al. 1994, DYNASMART web page), DYNEMO (Wiedemann and Schwerdtfeger 1987; Nökel and Schmidt 2002), ORIENT/RV (Axhausen 1989)), vehicles are still represented microscopically, but in contrast to the microscopic approaches described above, travel times and speeds are calculated using aggregates. By using a parallel implementation based on threads, METROPOLIS managed to simulate large scenarios efficiently. DynaMIT uses functional decomposition (task parallelization) as parallelization concept. Different modules are run in parallel, but the traffic simulation is executed on a single CPU only. The parallel implementation of DYNEMO (Nökel and Schmidt 2002) used message passing on 19 CPUs for simulating small scenarios. Larger numbers of CPUs were reported to be inefficient.

The highest level of abstraction can be found in macroscopic models that compute all traffic quantities on an aggregated level. One example of a traffic

simulation model as a one-dimensional incompressible fluid is NETCELL (Cayford et al. 1997).

The work presented here builds on the approach of MATSIM-T (Cetin 2005, MATSIM-T web page). While using simplified, queue-based dynamics to be computationally efficient, MATSIM-T still estimates all quantities microscopically and thus should be located somewhere between mesoscopic approaches and cellular automata. The same holds for the approach presented in (Mahut 2001) where the computational effort on links is minimized by only calculating the entry and exit times of vehicles.

Time-step based approaches have been more popular in the past than event-driven approaches. It is not clear why, but one reason might be the more straightforward implementation of time-step based simulations.

In this work, we extend an event-driven approach presented in (Charypar et al. 2007) using parallelization for the simulation of larger scale scenarios.

CLASSIFICATION OF TRAFFIC FLOW MICROSIMULATION METHODS

Cellular Automata

Cellular automata are used in traffic flow microsimulation to accurately model the behavior of cars traveling on a road network (for example Chowdhury et al. 2000). The basic idea is to discretize space in cells of equal size, each of which can be occupied by, at most, one vehicle. The cars drive through these cells by choosing their speed according to the space available in front of them. Cellular automata have the advantage of simplicity while still retaining a fair amount of spatial resolution, which – among other things – has the advantage that link capacities are generated from the properties of the links and the behavior of the drivers.

The major drawback of this method is its computational cost as every agent is simulated in every time step (usually one second). This becomes impractical if we are interested in large scale simulations.

Queue-Based Simulations

If the computational speed of a model is not sufficient for a certain kind of application, one solution is to switch to a simpler one. We follow this approach in MATSIM-T, (see for example Cetin 2005; MATSIM-T web page*)* where we use a queue-based approach to achieve higher performance. Here, the basic assumption is that the main features of (at least) urban traffic can be modeled by looking at the intersections alone, resulting in a model where:

- Links are simulated as they “process” cars traveling through the network.
- A queue stores cars traveling on each link together with their respective entry times.
- Each link’s capacity and space available for cars are parameters of the model.
- To move cars through the network, consecutive links collaborate in each time-step monitoring various constraints (capacity, free speed travel time, intersection precedence, and space available at the next link).

Queue-based models are faster than cellular automata mainly because the number of simulated units is smaller (links vs. cars), thus we gain roughly a 10 to 100 factor depending on the resolution of the network.

Event-Driven Queue-Based Simulations

To boost simulation performance further, we seek to eliminate inefficiencies in the queue-based approach. One such inefficiency can be observed when low density traffic regions of the simulation are investigated: While each link is processed in every time-step, most of the time no cars are really moved.

In order to achieve higher performance, we chose to substitute constant time-step for the direct treatment of actions occurring on the road network. Each such action is reflected by an event. Every time a car enters or leaves a link, a corresponding event is processed, meaning that the event processing rate is proportional to the traffic flow at any time. The two main advantages of the event-driven approach are:

- Processing time is assigned according to the traffic flow during any time of the day. As a result, most computational time is spent where traffic flow is maximal and almost no time is spent where the traffic network is empty.
- Prediction of processing time for a given travel demand is easy as it is proportional to the overall traffic volume simulated.

The event processing mechanism introduces an overhead compared to time-step based approaches that might be a concern in situations with large traffic flow. However, so far we have not come across a situation where the time-step based approach outperforms the event-driven simulation even when simulating only rush hours. For 24 hour scenarios, one can usually expect an overall speedup factor of 10 or more over the time-step-based implementation.

In METROPOLIS (see De Palma and Marchal 2002), a similar, event-based approach is used. The important difference there is that travel times and speeds are

estimated using aggregated values, while in our implementation, no aggregates are used.

A comparison of the simulation approaches' spatial and temporal discretization schema is shown in Table 1.

[[Table 1]]

MODIFICATIONS TO QUEUE DYNAMICS

In the process of implementing a parallel event-driven approach we have also added several features not present in former implementations of queue-based microsimulations:

- *Gaps, gap travel speeds*: The new implementation uses a concept of gaps that travel backwards through road segments at a constant speed. They are used to produce a delay between the event of a car leaving the road segment at the downstream end and the time the formed gap becomes visible at the upstream end. The benefit of this approach is mainly that it improves the way congestion dissolves.
- *Capacity constraints at inlet*: In addition to the outflow capacity, the inflow capacity of each link is constrained in the new implementation. This improves the behavior at converging Y-type connections where in former implementations breakdowns were predicted too far downstream.
- *Handling of gridlock*: All microsimulations have to cope with the issue of gridlock that happens if agents block each other at intersections. We solve this problem by guaranteeing a certain minimum inlet capacity for each road segment. This capacity is

available even if the road segment is already full allowing it to temporarily overfill. By permitting additional agents to enter a link they are removed from the problematic intersection which in turn averts gridlock.

SOFTWARE DESIGN: TECHNICAL DESCRIPTION

In the following section, we present the different parts of our event-driven queue-based traffic flow microsimulation and we explain how these parts work together to solve the simulation problem at hand. This only covers the single CPU implementation, the parallelization of which is discussed in the next section.

Description of Travel Demand

For the input travel demand to our traffic flow microsimulation we employ the formalism used in (Balmer et al. 2006):

- We use a population of agents, each having a complete 24-hour activity plan.
- An activity plan consists of an activity pattern describing the type and order of activities executed, location choice information describing where the activity in question should be executed, timing information defining the time of day when the activities are taking place, and routing information describing the sequence of road segments taken to travel between subsequent activities.

The Traffic Flow Model

The basic concept of our traffic flow model is that we only simulate transitions of agents from one road segment to the next. During the time spent on a specific road

segment, very little information about the position of the agent is available: only the position in the queue is stored and the earliest time that the car could leave the link according to free speed limitations. When the agent arrives at the end of a road segment, an event occurs and the agent leaves the road segment and enters the subsequent one according to the route stored in his complete activity plan. However, this can only happen if there is sufficient space left on the downstream link to take the oncoming agent. Otherwise, the agent has to wait until sufficient space is available.

Travel times derive from the behavior of the agents on the link: if a link is empty, the travel time equals the time needed to drive down the link at free speed. The travel time can lengthen if there are other agents traveling in front of the agent. In this case, the agent cannot leave the link before all agents in front of it have left the link. The reason for such a delay is usually high demand coupled with limited outflow due to limited outflow capacities or a blocked road network downstream. That means, as expected, large traffic volumes coupled with low link capacities will lead to long travel times

Simulation Output

The primary output of our traffic flow microsimulation is a log file that lists each event during the course of the simulation run. The complete description of an event consists of the absolute time of day when the event happened, an agent-ID and a link-ID specifying which agent was involved and where the event happened, respectively, and the type of event e.g. entry or exit. Using simple post processing, this *events file* can be converted into count data, hourly volumes on links, person diaries, or data similar to GPS floating-car data.

Actors During the Course of the Simulation

In our simulation software, three basic elements combine to compute traffic flow through the network (see Figure 1). These three elements are:

- The road segment that holds the cars during their travel,
- The agent that; 1. represents the object moving through the road network and 2. stores all information about the travel demand.
- The clock that handles registered timers and alerts the agents when an action must be undertaken.

[[Figure 1]]

The Traffic Flow Protocol

The actors mentioned above all interact and communicate according to a certain protocol explained and illustrated by the example in Figure 2, where one agent wants to travel along a certain sequence of road segments. The protocol consists of the following steps:

1. The agent informs the road segment that it wants to enter. The road segment stores this request chronologically together with all other requests of this kind.
2. As soon as it is clear when a gap will be available for the agent, the road segment sends this entry time to the agent.
3. The agent registers a timer for this time with the clock.
4. As soon as the timer expires, the clock sends a wake-up call to the agent and informs it about the time.
5. The agent then enters the road segment. With this step, the first part of the protocol finishes, and the agent continues to travel down the road. Note that

this does not require any actions to be taken by the road segment, the agent, or the clock.

6. When the agent moves up to first place in the road queue, the road segment computes the time the agent is going to arrive at the end of the road segment and sends this information to the agent.
7. Similar to step 3, the agent registers a timer with the clock.
8. After the timer expires, the clock calls the agent.
9. Now the agent is at the front of the road segment. It then checks with its activity plan for the next road segment to travel and registers there.
10. When it is clear at what time there will be a gap available for the agent to enter the next link, it receives a message with the predicted entry time.
11. The agent again registers a timer for this time.
12. When the timer expires, the clock wakes the agent.
13. The agent informs the last road segment that it is leaving.
14. It also informs the next road segment that it enters. From here on, steps 6 to 14 repeat until the last road segment of the trip is reached; then, step 14 is left out and the protocol stops.

PARALLELIZATION

Using queue-based link dynamics and an event-driven approach running large scale scenarios (with roughly one million person days simulated on a network with roughly 10k links) becomes feasible on single CPU desktop computers (Charypar et al. 2007). However, when going to even larger scenarios (roughly 10 million agents on high resolution networks with 100k links and more) the computational burden and memory consumption again become an issue. This is especially true if we want to

iterate 20 times or more during the search for a user equilibrium. To be able to do this for very large scale scenarios it is necessary to speed up the traffic flow simulation even more as we have less than 30 minutes for one simulation run if we want to finish overnight. To achieve this further reduction in processing time and also to cope with the memory demand we use parallelization to spread the simulation across multiple processors of a parallel computer.

Domain Decomposition

In order to distribute the computation across multiple processors we decompose the simulation domain (i.e. the network). The basic idea is to subdivide the network into parts and assign all nodes residing in one part to the same processor. Road segments that connect nodes assigned to the same processor are simulated entirely on that processor while road segments connecting nodes on different processors are simulated on those two processors jointly involving periodical communication.

To achieve reasonable parallel speedup it is important to have equal workload on all processors and to minimize the interfaces between individual parts of the domain to reduce the communication needs to a minimum. To achieve these two goals, the domain is partitioned using orthogonal recursive bisection, selecting the splitting plane such that the daily traffic volumes in both parts are equal. For this process, we use load data available from a previous iteration (Figure 3). This decomposition of the simulation domain is a simple, practical and efficient algorithm that can be implemented without too much effort.

[[Figure 3]]

The domain decomposition also manages how the memory demand of the whole simulation is distributed across processors: At any time each processor holds

solely agents currently residing on a link in care of that processor. This distributed handling of information allows to make use of the memory resources available on all processors together.

Communication

The only parts of our parallel program that involve communication between processors are road segments crossing the boundaries between sub domains (indicated as black links in Figure 3). In our program, such a road segment is split into two parts – the road start and the road end – which are then simulated separately on the two processors involved. From time to time synchronization messages are sent between the two parts of the same road segment. This is done to exchange information about agents that entered or left the road segment since the last synchronization. If agents travel across the simulation boundary between two processors these agents including their daily activity plan are packed into the synchronization message and thereby transferred to a different processor. This means that agents starting on one processor may go to any other processor during the course of the simulation.

The synchronization messages between the two parts representing a road segment have to be issued often enough such that the individual parts cannot become invalid. On the other hand, we want to communicate only as often as necessary in order to keep communication overhead as low as possible. Fortunately, the synchronization interval for each road segment (the time between two subsequent synchronization messages on one road segment) can be chosen independently of any other road segment. It solely depends on the speed of information propagation across this specific road segment. As the only information transported across road segments

are agents traveling and the gaps that travel in the opposite direction, it is the free speed travel time of agents and gaps that give us the desired synchronization interval. For a given road segment this value is given by the length of the road segment and the larger of the free speed travel time and the gap travel time.

In addition to the synchronization interval, we have to define at what specific times synchronization messages are sent. It is important that corresponding parts of the same road segment send and receive messages at the same time. Our solution to this problem is to start all processors at the same simulation time (the beginning of the period to be simulated, e.g. midnight) and communicate each time a synchronization interval has passed.

Technical Description

Our software is an instance of an explicit parallel program. We have used the Message Passing Interface (MPI) which is a quite flexible solution that makes it possible to run our code on all kinds of parallel computers including shared memory architectures and computer clusters. However, the performance is limited by the communication possibilities between processors and therefore it is to be expected that our software has larger potential for parallelization on shared memory computers than on computer clusters connected using cheap networking components.

RESULTS

Test Scenario

The test scenario we use to demonstrate the properties of our simulation software consists of the following parts:

- The road network of the federal states of Germany Berlin and Brandenburg. It consists of 11.6k nodes and 27.7k links.
- The synthetic population of the area consists of 7.05M people. Each person has a complete daily activity plan with multiple activities and trips. That means we are simulating 7.05M person days.

The average number of trips per agent in our demand is 2.02 and the average length of a trip is 17.5 links. This leaves us with an overall daily demand of 249M road segments to be traveled.

Computer System for Performance Analysis

All our tests were run on a shared memory parallel computer equipped with 64 dual-core Intel Itanium 2 processors with 1.6GHz and 256GiB of RAM.

Test Results

We ran the test scenario on the machine mentioned above. An average run using one CPU core took roughly 15 minutes to read in the travel demand, 25 minutes to produce the output file (events file) and 77 minutes to actually compute the traffic flow over the day. The following performance data refers to the computation time for the traffic flow, disregarding all input and output operations. Figure 4 shows how the performance of our simulation scales with the number of processor cores used. It can be seen that the system scales nicely using up to 64 processor cores where the performance is roughly 53 times the single core performance. The best parallel efficiency can be observed with 4 processor cores and with up to 16 cores the simulation runs with superlinear speedup. With 64 processor cores it is possible to run our test scenario in 87 seconds. Note that the speedup still increases around 64

processor cores and it might be possible to go beyond that point on even larger machines.

DISCUSSION AND OUTLOOK

We have shown that our method is able to simulate large scale scenarios as our test scenario of Berlin and Brandenburg at satisfactory speeds. However, in our current project, we aim at simulating a very large scale scenario where all people in Switzerland (roughly 7 million agents) are simulated on a high resolution network with a size in the order of 500k links. Using the property of our simulation that the processing time only depends on the total traffic volume, the processing time can be computed once this volume is known approximately. We estimate that the scenario describing all of Switzerland will be about a factor of 10 larger than the test scenario used in this paper. This would mean that it is feasible to compute the high resolution traffic flow for all of Switzerland within roughly 15 minutes on 64 processors. However, handling the tremendous amount of data related to these scenario sizes remains a major challenge. Currently, when running the simulation using 64 processors, more time is spent for input and output routines than for actually simulating traffic flow. Further work has to be done on this to reduce I/O needs of the simulation and to improve parallel I/O bandwidth.

Concerning functional extensions, we are working on the explicit modeling of intersections. This is not present in the current model where they are modeled implicitly. The first track followed here is the modeling of traffic signals. Our model can be extended to handle red phases by temporary reducing the outflow capacity of the corresponding link to zero and restoring the original capacity with the next green phase. The second track is to incorporate general intersection dynamics based on

direct interactions between cars. The idea here is to limit the total traffic flow through an intersection. As the maximum flow is limited, all road segments crossing at an intersection are competing. A large load in one direction will therefore reduce the maximal throughput in all other directions. We believe that these extensions will lead to more realistic flow patterns.

SUMMARY

We have presented a traffic flow microsimulation using a queue-based and event-driven approach jointly. We chose our approach to achieve a significant speedup compared to e.g. cellular automata, accepting a certain loss in spatial resolution to be able to solve large scale traffic simulation problems. Compared to earlier queue-based approaches, our event-driven simulation saves time in areas of the road network where the traffic load is moderate to small leading to a speedup compared to the time-step-based approach of more than 10 for 24 hour scenarios.

In addition to accelerating our method, several other modifications were undertaken. The simulation now handles backward-traveling gaps, limits the inflow capacities of all road segments in a meaningful way, and handles gridlock by using a notion of guaranteed minimal capacities.

By using a suitable domain decomposition that balances the load on the processors and minimizes communication interfaces, we succeed in parallelizing our software which makes it possible to use parallel computers to boost the performance by at least another factor of 50. The test scenario used was a large-scale 24-hour scenario with seven million simulated person days on a network with 28k links.

REFERENCES

AIMSUN. http://www.aimsun.com. Accessed November 2006.

Axhausen, K. W., 1989. Eine ereignisorientierte Simulation von Aktivitätenketten zur Parkplatzwahl. Ph.D. thesis. University of Karlsruhe, Germany, 1989.

Balmer, M., Axhausen, K. W., and Nagel, K., 2006. A demand generation framework for large scale micro-simulations. In *TRB 85th Annual Meeting Compendium of Papers.* CD-ROM. Transportation Research Board, Washington, D.C., 2006.

Barceló, J., Ferrer, L., Garcia, D., Florian, M., and Le Saux, E., 1998a. Parallelization of microscopic traffic simulation for ATT systems. In P. Marcotte and S. Nguyen, editors, *Equilibrium and advanced transportation modelling*. Kluwer Academic Publishers, 1998, pp. 1-26.

Barceló, J., Ferrer, L., Garcia, D., and Grau, R., 1998b. Microscopic traffic simulation for ATT systems analysis. A parallel computing version. Contribution to the 25th anniversary of CRT, University of Montreal, 1998.

Ben-Akiva, M., Bierlaire, M., Koutsopoulos, H., and Mishalani, R., 1998. Dynamit: A simulation-based system for traffic prediction. DACCORS Short Term Forecasting Workshop, The Netherlands, 1998.

Brilon, W. and Wu, N., 1998. Evaluation of cellular automata for traffic flow simulation on freeway and urban streets. In W. Brilon, F. Huber, M. Schreckenberg, and H. Wallentowitz, editors, *Traffic and Mobility: Simulation – Economics – Environment*, Springer, Berlin, 1998, pp. 163–180.

Cayford, R., Lin, W.-H., and Daganzo, C. F., 1997. *The NETCELL simulation package: Technical description.* California PATH Research Report UCB-ITS-PRR-97-23, University of California, Berkeley, 1997.

Chang, G. L., Junchaya, T., and Santiago, A. J., 1994. A real-time network traffic simulation model for ATMS applications: Part I— Simulation methodologies. *IVHS Journal*, 1(3), 1994, pp. 227–241.

Charypar, D., Nagel, K., and Axhausen, K.W., 2007. An event-driven queue-based traffic flow microsimulation, *86th Annual Meeting of the Transportation Research Board,* Washington, D.C., January 2007.

Cetin, N., 2005. Large scale parallel graph-based simulations. PhD thesis, Swiss Federal Institute of Technology ETH, 2005.

Cetin, N., Burri, A., and Nagel, K., 2003. A large-scale agent-based traffic microsimulation based on queue model. In Proceedings of Swiss Transport Research Conference (STRC), Monte Verita, CH, 2003.

Chowdhury, D., Santen, L., and Schadschneider, A., 2000. Statistical physics of vehicular traffic and some related systems. *Physics Reports*, 329(4–6), 2000, pp. 199–329.

De Palma, A., and Marchal, F., 2002. Real case applications of the fully dynamic METROPOLIS tool-box: An advocacy for large-scale mesoscopic transportation systems. *Networks and Spatial Economics*, 2(4), 2002, pp. 347–369.

DynaMIT. http://mit.edu/its/dynamit.html. Accessed January 2007.

DYNASMART. http://www.dynasmart.umd.edu. Accessed January 2007.

Mahut, M., 2001. Discrete flow model for dynamic network loading. Ph.D. thesis, University of Montreal, Canada

Marchal, F., 2001. Contribution to dynamic transportation models. Ph.D. thesis. University of Cergy-Pontoise, Cergy-Pontoise, France, 2001.

MATSIM, Multi Agent Traffic SIMulation. http://www.matsim.org. Accessed January 2007.

MITSIMLab. MIT ITS Program, Cambridge, MA. http://web.mit.edu/its/mitsimlab.html. Accessed January 2007.

Nagel, K., and Rickert, M., 2001. Parallel implementation of the TRANSIMS micro-simulation. Parallel Computing, 27(12), 2001, pp. 1611–1639.

Nagel, K., Wolf, D. E., Wagner, P., and Simon, P. M., 1998. Two-lane traffic rules for cellular automata: A systematic approach. Phys. Rev. E, 58(2), 1998, pp. 1611–1639.

Nökel, K., and Schmidt, M., 2002. Meso-scopic traffic flow simulation on large networks. Networks and Spatial Economics, 2(4), 2002, pp. 387-403.

Rickert, M., and Nagel, K., 2001. Dynamic traffic assignment on parallel computers in TRANSIMS. Future generation computer systems, 17(5), 2001, pp. 637–648.

TRANSIMS, TRansportation ANalysis and SIMulation System. Los Alamos National Laboratory, Los Alamos, NM. http://transims.tsasa.lanl.gov. Accessed January 2007.

VISSIM. http://www.ptv.de/cgi-bin/traffic/traf_vissim.pl. Accessed January 2007.

Wiedemann, R., and Schwerdtfeger, T., 1987. Makroskopisches Simulationsmodell für Schnellstraßennetze mit Berücksichtigung von Einzelfahrzeugen (DYNEMO). *Forschung Straßenbau und Straßenverkehrstechnik, Heft 500.* Typo-Druck, Bonn, 1987.

Yang, Q., 1997. A Simulation Laboratory for Evaluation of Dynamic Traffic Management Systems. PhD thesis. Massachusetts Institute of Technology, 1997.

TABLE 1 Discretization Schema of Different Microsimulation Approaches

	Spatial discretization	Temporal discretization
Cellular Automata	equidistant	constant time-step
Queue-based simulation	road segments	constant time-step
Event-driven, queue-based simulation	road segments	adaptive, event-driven

FIGURE CAPTIONS

FIGURE 1

The boxes represent actors in our simulation, the arrows stand for flow of information and message passing.

FIGURE 2

The vertical lines indicate how time runs during the course of the simulation. The arrows between these lines represent messages sent between the actors of the simulation. The numbers in circles denote the respective protocol step of the message. See text for more information.

FIGURE 3

Domain decomposition of the road network of the federal states of Germany Berlin and Brandenburg: On the left you can see a complete view while on the right a close up of the center section is shown. Each colored area corresponds to the same traffic load and represents the assignment to a specific processor for the parallel simulation using 16 processors. The thin black lines represent the domain boundaries. Thick black lines indicate links that cross subdomain boundaries and involve communication.

FIGURE 4

The software scales nicely with the number of processors used. Note the superlinear speedup for up to 16 processors probably due to better cache efficiency. For 64 processors the speedup reaches roughly 53.

FIGURE 1 Interaction Processes Between Elements of the Traffic Flow Microsimulation.

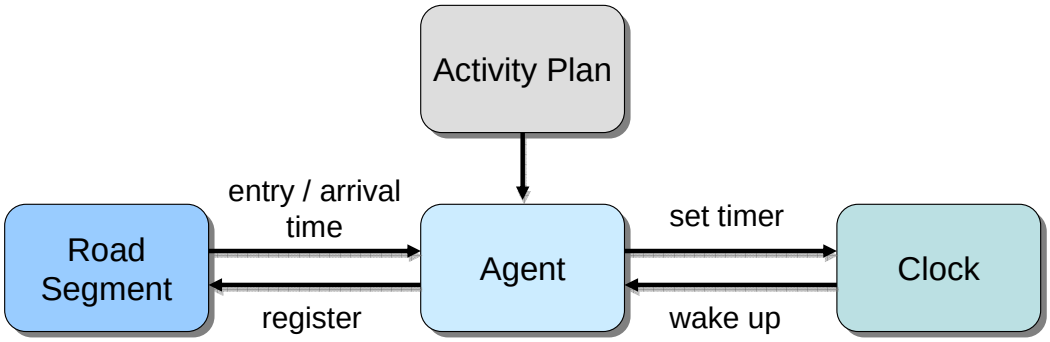

FIGURE 2 The Traffic Flow Protocol

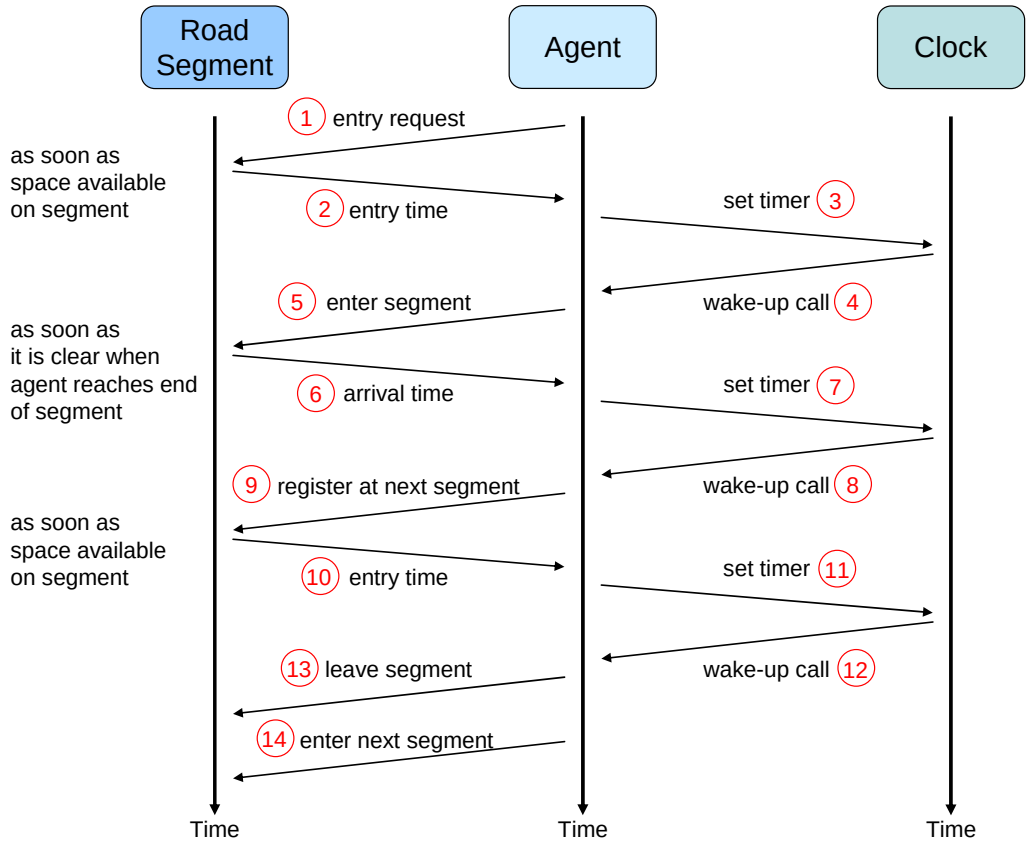

FIGURE 3 Domain Decomposition of the Network.

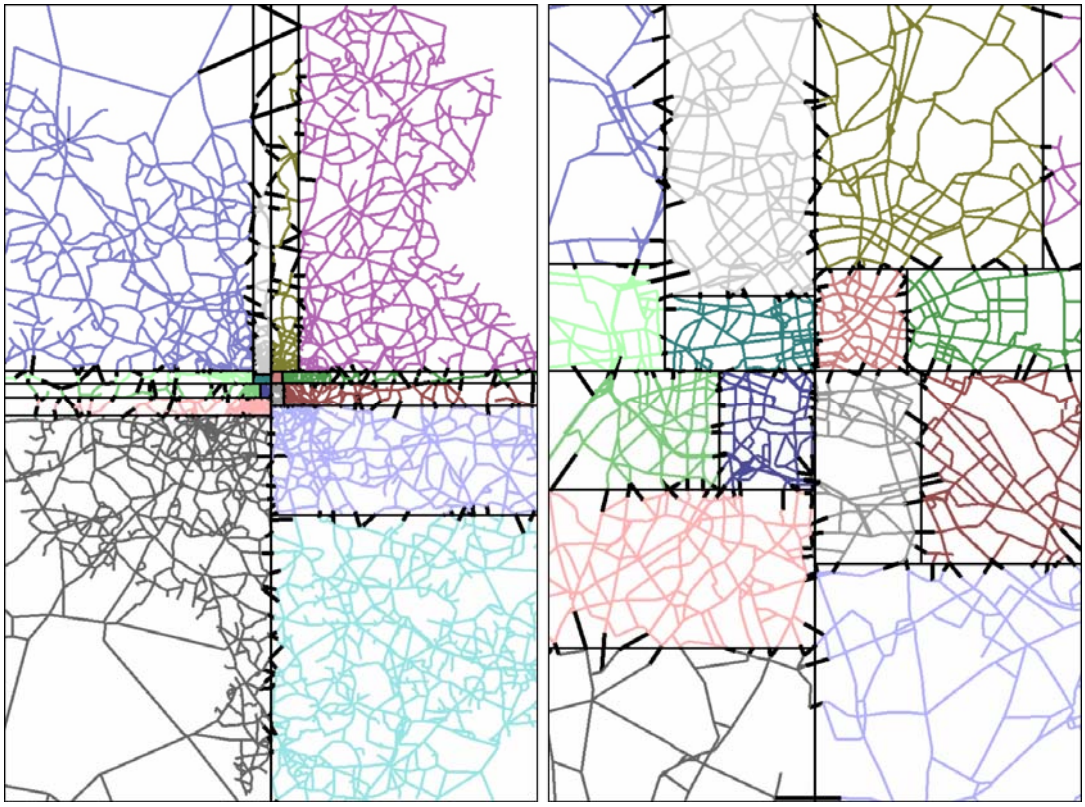

FIGURE 4 Speedup of the Parallel Simulation.