A high-performance traffic flow microsimulation for large problems

Report

Author(s):
Charypar, David; Balmer, Michael; Axhausen, Kay W.

Publication date:
2008

Permanent link:
https://doi.org/10.3929/ethz-a-005652298

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
Working Paper / IVT 509
A High-Performance Traffic Flow Microsimulation for Large Problems

David Charypar
Michael Balmer
Kay W. Axhausen

Working paper 509
Transport and Spatial Planning 2008
A High-Performance Traffic Flow Microsimulation for Large Problems

David Charypar
Institute for Transport Planning and Systems
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
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
ETH Zurich
CH-8093 Zurich
phone: +41-44-633 39 43
fax: +41-44-633 10 57
axhausen@ivt.baug.ethz.ch

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. In this paper, we present how we extend our previous event-driven queue-based microsimulation to run efficiently on parallel computers. Using appropriate load balancing and minimizing communication interfaces, we are able to simulate a test scenario involving 7 million simulated person days on a road network with 28k links in 87 seconds on 64 CPUs. Furthermore, we add support for signaled intersections that makes the model well suited for application to urban street networks. Finally, we show that our resulting model reproduces a reasonable relation between traffic flow and density similar to fundamental diagrams extracted from real world counts data.

Keywords
Traffic flow microsimulation, Event driven simulation, Parallelization, Green time fractions, Large-scale scenario

Preferred citation style
1 Introduction

In transport planning, traffic flow simulation plays an important role as it takes the demand generated by an earlier process as input and produces derived quantities like traffic densities, flow volumes, speeds, and travel times as output. These quantities are in turn important for instance to analyze the state of the traffic system or to enable iterative demand generation based on one or multiple of these values.

There are several choices when it comes to deciding on the type of flow simulation: macroscopic aggregated models are available just as mesoscopic and microscopic approaches. For resolution and accuracy, microsimulation would clearly be the method of choice here. Furthermore, using a microscopic point of view, the advantages of advanced techniques in demand generation based on the individual, like activity based (e.g. Bowman et al., 1999; Vovsha et al., 2002) or agent-based (e.g. MATSim-T, 2008) demand models, can be fully exploited by utilizing the same individuals throughout.

The downside of microsimulation is the computational burden that comes with these methods. One explanation for this is that data with very high temporal and spatial resolution must be processed as well as produced. This inevitable property represents a major challenge when trying to use such a method for large scale scenarios. Another disadvantage is arguably the added complexity in the demand generation process for creating a virtual population: zone-resolution is no longer sufficient since microscopic flow simulations need the individual traffic objects (i.e. the cars). On the other hand, tedious procedures known from macroscopic modeling, like zone definition, placement of connector links, and generation of OD-matrices, can be avoided this way. Perhaps as a result of these properties, microsimulations are seldom used for large scale scenarios today. Instead, mostly macroscopic models and sometimes mesoscopic models are employed.

One way to reduce the computation time of microsimulations is by running them in parallel on multi-CPU computers and large computer clusters. Unfortunately, it turns out that this approach cannot be generally applied to each microsimulation model, as often the resulting parallel performance of such approaches is poor. With the advance of multi-core CPUs in desktop computers this fact is currently leading to the paradoxical situation that increases in processing power often do not accelerate traffic flow microsimulations.

In this work, we present a microsimulation program that realizes decent single CPU and parallel performance making it suitable for very large scale scenarios. This speed is achieved by bearing in mind implementation and parallelization issues already during model design. However, looking at performance only is certainly not sufficient for traffic analysis. We need to make sure that the model also shows reasonable flow properties. Analysis of such properties therefore forms an important part of this work. The third focus of this paper is the functional extension
of introducing traffic lights into the model. This extension allows the transport modeler to cope with the intrinsic heterogeneity of simulating large scale scenarios with rural as well as urban areas.
2 Traffic Flow Simulation Approaches and Related Work

In this section, we give a brief overview of different classes of traffic flow simulation approaches. Their level of abstraction has strong influence on the resulting accuracy, resolution and simulation speed.

2.1 Physical Microsimulation Models

Physical microsimulation models represent the most accurate and expensive type. Their aim is generally to mimic real traffic dynamics by imitating human car following, simulating lane changing behavior, employing sophisticated intersection dynamics, etc. Space is often represented continuously and simulated time is advanced in small time-steps. Examples are AIM-SUN (Barceló et al., 1998; AIMSUN, 2006), MITSSIM (Yang, 1997; MITSIM, 2006), and VISSIM (VISSIM, 2006). Barceló et al. (1998) demonstrated a parallel implementation using globally accessible data which ran 3.5 times faster on 8 processors.

For large scenarios however, physical simulation methods are still too expensive; to become faster, one has to “trade in” some of the accuracy and resolution for speed.

2.2 Cellular Automata

In cellular automata (CAs, see e.g. Nagel and Schreckenberg, 1992; Chowdhury et al., 2000; Brilon and Wu, 1998), roads are discretized in cells each of which can be either empty or occupied by a car. Each car controls its driving speed based on the available head space to the front of it. CAs are faster than physical models while retaining a fair amount of spatial resolution.

A CA is adopted for example in TRANSIMS (Nagel et al., 1998; TRANSIMS, 2006). Nagel and Rickert (2001) and Rickert and Nagel (2001) showed a parallel version using message passing between processors, running midsized scenarios on 32 CPUs. Although the running times were about ten times faster than real time, ethernet latency problems and speed handicaps due to the use of cellular automata were reported.

What limits CAs’ speed is the necessary update of each car’s position in every time-step. This becomes impractical for large numbers (>1 million) of cars.
2.3 Queue-Based Simulations

The model can be further simplified and thereby accelerated by representing links as queues. This approach was followed in MATSim-T (e.g. Cetin, 2005; MATSim-T, 2006). The computational advantage comes from the shift from the cars being the main simulated elements to links that “process” the cars. This reduces the number of simulated units (links vs. cars) and thus simulation performance increases by a factor of 10 to 100, depending on network resolution. The links collaborate to move the cars through the network while adhering to various constraints (capacity, free speed travel time, intersection precedence, and space availability).

Using message passing between cluster nodes, the queue-based model presented in (Cetin, 2005; Cetin et al., 2003) achieved a speed-up of 32 using 64 CPUs when simulating a peak period. Charypar et al. (2007) presented a single processor queue-based simulation based on Cetin (2005), that replaces the constant time-steps by an event-driven approach improving efficiency by more than a factor of 10. The idea of advancing time as necessary from event to event can be found in Mahut (2000) as well, where the entry and exit times of vehicles are computed directly.

2.4 Mesoscopic Models

In contrast to microsimulation, mesoscopic models (e.g. METROPOLIS (Marchal, 2001; de Palma and Marchal, 2002), DynaMIT (Ben-Akiva et al., 1998; DynaMIT, 2006), DYNASMART (Chang et al., 1994; DYNASMART, 2006), DYNEMO (Schwerdtfeger, 1984; Nökel and Schmidt, 2002), ORIENT/RV (Axhausen, 1988)) use aggregates to compute travel times and speeds.

METROPOLIS is able to simulate large scenarios efficiently by using a parallel implementation based on threads. DynaMIT does not parallelize the traffic flow simulation itself but uses task parallelization i.e. different modules are run in parallel. DYNEMO was run in parallel (Nökel and Schmidt, 2002) by using a message passing technique on 19 CPUs for simulating small scenarios. Larger numbers of CPUs were reported to be inefficient.

2.5 Macroscopic Models

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).
3 Traffic Flow Model

This section summarizes the traffic flow model we presented in Charypar et al. (2007), which forms the base for the extensions presented in the following three sections: support for traffic lights, analysis of the flow properties of the model, and the parallelization of the model to run efficiently on many CPUs.

3.1 Event-Driven Queue-Based Model

The main overall aim during the development of the presented model was computational speed. As mentioned earlier, we are targeting very large scenarios with up to 10 million simulated agents. It is important to realize that in this range of size we are competing mostly with macroscopic models and not with high resolution microsimulations of the physical or cellular automata type. For this reason, it was decided to develop a queue-based model to achieve the maximal possible speed.

At the base of the simulation model there are the road segments which represent the built infrastructure and a population of agents that represent the people inside cars that use the infrastructure. Between agents there exist gaps which are represented explicitly:

- The road segments are directed links between a starting and an end point. They have a length, an amount of available space, a minimum travel time and two flow capacities, for in- and outflow, respectively.

- Each agent has a complete 24-hours activity plan and represents a person traveling by car. The plan consists of an ordered list of activities, corresponding activity start times and durations, and the route to travel between subsequent activity locations.

- The population of virtual agents together with their activity plans represent the travel demand.

- In the opposite direction of agents gaps move upstream to produce the necessary delays between a car leaving a congested link at the end and a new car entering that link at its starting point.

During the course of the simulation agents and road segments work together by means of message passing. The basic idea is simple: In order to enter or leave a specific road segment, each agent needs to ask the segment for approval first. Approval is not given until execution of the corresponding action will not violate any of the following requirements:
Agents can only leave the road segment in the same order as they entered.

The road segment must not hold more agents than a certain maximum.

There is a minimum gap between two consecutive cars according to inflow capacity.

Outflow is constrained analogously.

To enter a road segment, there must be a gap available at the inlet. Gaps travel upstream at a constant predefined speed.

An agent cannot leave the road segment before the free speed travel time has elapsed.

If a road segment has available space, an agent’s request is accepted immediately. Only if the road is full, approvals are delayed, thereby leading to a reduction in travel speed.

One crucial observation with queue-based models in general is that basically all information about the course of simulation is contained in the entry and exit events. Between subsequent entry and exit events for a given car, its exact position is unknown. As a result, simulating the time between such a pair of events with time-steps only increases simulation costs.

Furthermore, on a heavily loaded link, one can even say that one event triggers the next. Assume for example an agent trying to enter a road segment that is completely filled with cars. The agent sends an entry request message to the road segment and waits until the next space becomes available. At some point in time, the leaving event of the front car produces a gap that travels upstream at constant speed. When it reaches the upstream end, it becomes available for the new car to enter the link. This predetermination between subsequent events can be exploited for simulation efficiency. It makes it possible to avoid constant time-steps and instead use variable time-steps to advance time from one event to the next. This has a couple of advantages:

- Computational events correspond to traffic events. That means, that computation time is spent proportionally to the traffic flow. This denotes an adaptive assignment of processing resources according to instantaneous traffic flow.

- Since the event processing rate is more or less constant on a given machine it is easy to predict how long it will take to simulate a given traffic demand.

By using variable time-steps based on events, one can realize an overall speedup factor of at least 10 over the time-step-based implementation.

To coordinate the necessary variable time-steps, we introduce a centralized clock object. All agents and road segments use this clock to register timers as necessary for the event handling.
Figure 1: The traffic flow protocol: the vertical lines indicate how time runs during the course of the simulation; the arrows between these lines represent messages sent between the objects of the simulation; the numbers in circles denote the corresponding protocol step of the message.

process described above. When such a timer expires, the corresponding object gets informed and is then able to take the corresponding action. The rules that define these objects’ interactions form the so-called flow-protocol, which is illustrated in Figure 1 for the case of an agent traveling along a line of road segments.

Source: Charypar et al. (2007)
4 Green Time Fractions

The simulation of 24 hours scenarios on urban networks with many signaled intersections often involves a special problem: The observable link flow capacities on crossing roads do not always keep a fixed ratio as the signal control parameters may be time of day dependent. This type of intersections cannot be handled correctly with constant flow capacities. To alleviate this problem, we introduce green time fractions that modulate the links outflow capacity. These green time fractions can be varying over time, enabling the simulation of different traffic control schemes over the day.

4.1 Model Extension

The general idea is to introduce on each link an explicit representation of the outgoing capacity reduction based on the fraction of time the associated traffic light is switched to green. We simply assume the average capacity realized on such a link is proportional to the relative green time. For example on a link with its traffic light switched to green during 30\% of the time on average, we expect to observe an average realized capacity of 30\% of the free flow capacity. Admittedly, there probably is no linear relation between green time fractions and realized capacity. Still we choose this model to make the simulation easy to understand. After all, we can switch to an interpretation of the data where we replace relative green time by relative useable green time, rendering our linear model correct.

In our model, the green time fractions per link are not constant but are merely modulated, each individually, by a piecewise linear function of time. Other models would have been possible (including piecewise constant modulation functions), but we think it is a beneficial property to have a continuously changing capacity on the links to avoid undesired shocks in the system. After all, if desired, very quick changes in capacity can still be realized.

The question might arise why we model averaged green time fractions instead of individual green phases. There are multiple reasons to mention:

- The necessary data is difficult to obtain and process, especially for very large scenarios with hundreds or even thousands of traffic lights. These scenarios often cross administrative boundaries resulting in a lot of heterogeneity in the input data.
- The data cannot be easily made up and small errors in these numbers may produce totally different realized flows on crossing links.
- For similar reasons, it is very difficult to test new policies using individual green phases compared to the relative ease of adapting the average green time fractions on a road.
Note, that if there is, for some reason, the need to simulate individual green phases on certain intersections, our model is flexible enough to mimic such a behavior by changing the green times fraction periodically. For example, a link’s green time fraction can be set to alternate from 100% green for 73 seconds to 0% green (i.e. red) for 128 second by defining a corresponding modulation function in the input data.

This input data is provided as a separate file (the green time fraction file) in a human readable XML format. In this file, for each link incorporating traffic lights, there is a list of data points, specifying the green time fraction on that link at a given point in time. Furthermore, the simulation assumes that the values repeat periodically according to a period defined in the same file. In Figure 2 you can see a plot of example green time fractions for one link.

4.2 Technical Description

Internally, for each of the road segments holding green time fractions data an instance of a special modulator object is created. Apart from holding the data, this object is also responsible for computing the car waiting times on the associated road segment.

The data is stored in a sorted data structure allowing quick access to any data point needed during computations. Each data point consists of a time stamp and a green time fraction.
When a car wants to leave a signaled road segment, just as before without signalling, it requests the next available time slot. This value depends on the time when the last car left the link and on the out flow capacity of the road segment, which is now time dependent. Technically, the car has to wait until the integral of the time dependent capacity reaches $1[\text{car}]$. This integral of the piecewise linear capacity function $q(t)$ is the piecewise quadratic volume function $V(t)$. If $t_l$ is the time when the last car left the road segment, the requested next leaving time $t_n$ can be computed by solving the equation

$$V(t_n) - V(t_l) = 1$$

for $t_n$. This involves inverting the volume function which is not trivial, since it is a piecewise quadratic function, and it is not clear which piece will contain the $t_n$ to be computed. Our implementation solves this problem by starting at $t_l$ and integrating each piece of $q(t)$ sequentially until the integral exceeds the required $1[\text{car}]$. At this point, it is clear that the last covered piece contains the $t_n$ in question and this can be easily found by solving a quadratic equation.

### 4.3 Demonstration of New Simulation Possibilities

If inclusion of new links into existing road networks is considered, for instance the creation of a city by-pass, accompanying measures are often investigated. Changing the timing and prioritization of traffic lights is often one of these measures tested and the presented extension of our traffic flow model provides a way of rendering such ideas in the simulation.

Our model was used at our institute in a recent project where the effects of opening a city by-pass tunnel around the city of Zurich, Switzerland, were investigated (see Balmer et al., forthcoming). The scenario involved a road network with 60k links, used by around 670k agents each of which executing its 24-hours activity plan. One of the questions under investigation was how changes in traffic flow patterns could be influenced by modifying the traffic lights policies around relevant points in the network. Figure 3 visualizes how peak hour traffic volumes change if traffic flow into and out of the city center is restrained by using traffic lights. The section shows an area around the freeway exit “Wollishofen” near the east entrance of the tunnel under investigation. One can see clearly that traffic starts to avoid the restrained road and starts to use alternative routes.
Figure 3: Effect of changing traffic lights policy near the freeway exit Wollishofen. The numbers indicate the change in hourly traffic volumes of the peak hour. The circle in the center indicates where the traffic lights were changed, thickness of the roads indicates the number of lanes.
5 Flow Properties

To be able to use our model in a way that it produces the best possible simulation results, it is important to know its inherent properties. One especially important property is the relation between traffic density and traffic flow and how much this relation depends on external properties like the network resolution and the travel demand. With this information available, it is possible to tell if and how the model should be applied to certain classes of simulation problems. Can the model be used for the simulation of freeway traffic? Does the accuracy of the result change if we split long links into pieces? Such questions are ultimately under investigation in this section.

5.1 Averaged Characteristics

To find the relation between average flow and average density, we perform a virtual experiment on a circular test network. This circle is loaded accordingly to produce all relevant average traffic densities. The average traffic flow is then measured and the corresponding values plotted against each other. In Figure 4, we show the resulting flow-density plot. The graph has a trapezoidal shape, the left, upper, and right edges of which can be clearly associated with independent parameters of our queue-based model with gaps: The slope of the left edge is con-
trolled by the free speed defined on a road segment, the height of the top flat part corresponds to the flow capacity on the link, and the cut-off point together with the slope of the right edge are a result of the maximum density (fully congested density) and the backward traveling gap speed. Interestingly, increasing the network resolution by a factor of 500 does not change the average flow-density relation at all. Another interesting observation: all observed states on the network lie on the border of the graph. No state of medium average density together with medium average flow was observed in any of our tests. It seems that our model is not only able to produce the limit values, but it also will produce exactly these limit values if used on a ring topology.

5.2 Instantaneous Flow Characteristics

We are not only interested in the relation between average flow and average density: We also want to know, how these values relate locally and on a short time scale. To extract the desired values, the test setup was slightly modified by introducing a short measurement strip. Subsequently, density and flow at any time were measured on this sensor strip and no temporal smoothing was applied. For this test setup, the demand was left the same as in the previous case. Interestingly, the resulting instantaneous flow-density plot looked very similar to the average flow-density plot. The only noticeable difference was a certain clustering at high and low densities, so it seems medium densities are rare local observations. Maybe this can be explained through transitions from congested flow conditions (high density) to free flow conditions (low density) back again to congestion. This maybe can be interpreted as stop-and-go traffic.

5.3 Stochastic Flow characteristics

With the test scenario describe above we have found that our traffic flow model basically reproduces the envelope of what could be termed a stylized fundamental diagram.

Interestingly, the “inner” states, as they are often seen in real world fundamental diagram are never observed. To figure out if this is a property of our model we have created a new test case, basically an isolated short road segment that is fed by two stochastic and independent processes: On the upstream end, one such process is randomly producing cars trying to enter the road segment and at the downstream end another one is randomly producing outflow slots for cars arriving. This setup corresponds to the (admittedly unrealistic) situation where cars enter and leave a road segment with the highest possible fluctuation.

The resulting flow and density patterns show the same trapezoidal envelope as before, but now, the inner states are realized regularly.
5.4 Findings of Flow Characteristics Investigations

Based on the three conducted tests, we draw the following conclusions:

- The interrelation of traffic flow and density on the simulated road segments shows a trapezoidal envelope. This envelope is defined by four independent parameters of the queue-based model: free speed, capacity, gap travel speed, and maximum density.

- The network resolution of a test ring scenario has no effect on the resulting averaged density and flow patterns. We conclude that it is sufficient to simulate freeway traffic by using relatively long road segments.

- If the demand is deterministic the instantaneous state of the road segments almost never reaches points inside the envelope (at least in the ring test scenario).

- If there is sufficient fluctuation in the demand and in the backward traveling gaps (which are also induced by the demand) essentially every traffic condition can be reached (i.e. also inner states are possible).
6 Parallelization

Using queue-based link dynamics and an event-driven approach (see Section Traffic Flow Model) 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. This is especially true if the microsimulation is to be used for relaxation of an integrated demand model. Here, according to experiences made in the MATSim project (MATSim-T, 2008) at least 20 iterations need to be executed. That means 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.

6.1 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 (see Figure 5 for an illustration). For this process, we use load data available from a previous iteration. This decomposition of the simulation domain is a simple, practical, and efficient algorithm that can be implemented quickly. The domain decomposition also handles 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.
Figure 5: 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.

6.2 Communication

The only parts of our parallel program that involve communication between processors are road segments crossing the boundaries between subdomains (indicated as black links in Figure 5). 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. Periodically, 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 took place. 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 the 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 depends solely on the speed of information propagation across this specific road segment. As the only information transported across road segments is the agents traveling down the link and the gaps moving 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.

Here, it can be seen how the introduction of gaps into our model facilitates parallelization: Without gaps, information would travel upstream at very high speeds, basically one link per time-step, impairing parallel performance as in each time-step communicating across CPU boundaries would become necessary. Limiting the upstream propagation of information though a relatively low gap travel speed results in much longer times between subsequent synchronization messages. This increases the maximum parallel performance of the program.

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.

### 6.3 Technical Description

Our software is implemented as an explicitly parallel program. We have used the Message Passing Interface (MPI) ([MPI-Forum](https://www.mpi-forum.org)) 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.

### 6.4 Test Setup

We used the following test scenario to demonstrate the parallel performance of our simulation software:

- The road network of the federal states of Germany Berlin and Brandenburg consisting of 11.6k nodes and 27.7k links.
• The synthetic population of the area consists of 7.05M people. Each person has a complete 24-hours activity plan with multiple activities and trips. That is, we simulate 7.05 million 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.

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

6.5 Parallel Performance

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[6] 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.
Figure 6: The parallel microsimulation speed 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.
7 Summary

In this paper, we have presented a parallel event-driven traffic flow microsimulation supporting signaled intersections. The simulation model’s basic elements are the road segments. When cars drive on these segments, the dynamics is approximated by queues with backwards traveling gaps. To make the simulation of real world scenarios easier, the model is extended with support for traffic lights by modeling the average flow capacity according to green time fractions. We have shown that, while the used model is simple, the resulting flow-density plot reproduces a simplified fundamental diagram. The flow model with backwards traveling gaps has the additional advantage that it enables efficient parallelization: By using a suitable domain decomposition that balances the load on the processors and minimizes communication interfaces, we succeeded in running our software on a parallel computer with 64 CPUs with a speedup factor of 57.
References


