A Large-Scale Agent-Based Traffic Microsimulation Based On Queue Model

Nurhan Cetin, Dept. of Computer Science
Adrian Burri, Dept. of Computer Science
Kai Nagel, Dept. of Computer Science

STRC 03 Conference paper
Session Microsimulation

STRC | 3rd Swiss Transport Research Conference
Monte Verità / Ascona, March 19-21, 2003

A Large-Scale Agent-Based Traffic Microsimulation Based On Queue Model

Nurhan Cetin, Adrian Burri, Kai Nagel
Department of Computer Science
ETH Zürich
Zürich

Phone: +41 (0) 1 - 632 27 54
Fax: +41 (0) 1 - 632 13 74
eMail: cetin@inf.ethz.ch, nagel@inf.ethz.ch, burriad@student.ethz.ch

Abstract

We use the so-called queue model introducted by Gawron as the base of the traffic dynamics in our micro-simulation. The queue model describes the links with a *flow capacity* that limits the number of agents that can leave the link and a *space constraint* which defines the limit of the number of agents that can be on a link at the same time. *Free flow speed* is the third key component of traffic dynamics in the model.

Flow capacity and space constraint together model physical queues, which can spill back beyond the end of the link. A consequence of this is that fairness between the incoming traffic streams becomes an issue, since in a spill-back situation they cannot be served at their full rate. We implement and verify a simple solution to this; the solution is much simpler than the one chosen in many other models.

The traffic micro-simulation is "large-scale" which means the simulation is capable of modeling the behavior of millions of agents simultaneously. We utilize a parallel implementation to speed up the computation. In this implementation, the data is distributed onto a number of computing node, each of which runs a smaller portion of the data. Data distribution and communication among the computing nodes are achieved by freely available software libraries.

We test this simulation on two different scenarios using the road network of Switzerland. One of them is aimed to see how the simulation handles the congestion whereas the other one is based on real data of the daily activities of the Swiss people. The parallel version on the bigger scenario gives a runtime that is about 800 times faster than real time on 64 computing nodes using Myrinet. The maximum number of vehicles simultaneously in that simulation is about 160 000.

Keywords

Large-scale Traffic Simulation – Queue Model – Parallel Computing – 3rd Swiss Transport Research Conference – STRC 03 – Monte Verità

1. Introduction

The traditional four step process consists of (1) trip generation, (2) trip distribution, (3) modal choice, and (4) traffic assignment. An important feature of this approach is that at no point are travelers treated as individual entities – all information is aggregated into traffic streams. For example, the result of trip distribution is an origin-destination matrix, which gives constant streams from each origin to each destination.

There is some agreement that a higher level of realism is desirable for these reasons:

- Temporal dynamics. The four step process makes the approximation that traffic streams are static, which corresponds to the approximation that the underlying particle process is steady-state. This means that temporal effects such as peak spreading or time-dependent congestion spillback are not covered.
- Disaggregation. Many aspects of choice behavior, such as modal choice, become more realistic when, say, demographic data is included. However, there is no access to such variables in the four step process.

The second aspect could in principle be overcome by replacing the three first steps by a method which generates activity-based travel demand for a whole (synthetic) population. However, activity-based travel demand generation is time-dependent, and it is difficult to see how the resulting demand could be approximated by a time-independent steady-state process so that it would fit into static traffic assignment. This leads to the consequence that either both shortcomings of the four step process need to be overcome simultaneously, or one has to start by making traffic assignment dynamic.

"Dynamic traffic assignment (DTA)" is indeed a technique which has emerged over the last two decades (e.g. [18, 19, 24, 6]). The problem can be stated in the way that one has a time-dependent demand and a certain traffic dynamics, which moves vehicles along links (roads) and across nodes (intersections). For each individual one wants to find the route so that no traveler would be better off by selecting a different route. This is just a Nash Equilibrium statement for the dynamic problem.

Since no analytical solution is known for traffic dynamics which includes spillback, many groups have resorted to simulation. The typical simulation approach is systematic relaxation, via a variant of the following procedure:

1. Make some initial guess for the routes.
2. Execute all route plans simultaneously in a traffic micro-simulation. This is sometimes called the *network loading*.
3. Re-adjust some or all of the routes using the knowledge from the network loading.
4. Goto 1.

This can be interpreted as a multi-agent *learning* method. Many variations of this are possible, such as varying the fraction of routes which are replanned [42], using a probabilistic route choice

based on, say, multinomial logit or probit [18], or using different network loading algorithms [4]. It is also straightforward to add departure time choice [17], or other aspects of activity-based demand generation as indicated above.

This approach looks similar to relaxation methods for static assignment, where the simple link cost function is replaced by the network loading. What is lost when going from from static to simulation-based dynamic assignment is the mathematical knowledge. For example, for static assignment one knows that (under certain assumptions) the solution is unique (in terms of path flows) and therefore any method that converges toward a mathematically correct solution will converge toward the same solution.

Investigations into the theoretical nature of the dynamic traffic assignment problem have made some progress [10, 6], but so far no general statement about the nature of the problem, including uniqueness, is available. One *can*, however, show that the deterministic variants of above solution method will have (attractive or repulsive) fix points in very high dimensional phase space, and that stochastic variants will, under certain conditions, converge to a steady-state density in the same very high dimensional phase space [6]. Under additional conditions, it will even be ergodic [10].

One needs to notice however that the conditions for these statements are fairly restrictive when confronted with simulation reality. The main problem with these requirements is that they assume that the whole phase space is given at the start of the simulation. For example, one needs to know in advance every route that any traveler will eventually use throughout the procedure. This makes the use of explorative algorithms, which generate new routes as they go, inconsistent with the mathematical formulation. Put in a different way, it is clear that a simulation system that keeps exploring new strategies is not in the steady-state. Similarly, theoretical ergodicity is only valid for the limit of the number of iterations going to infinity, while in practice one is restricted to about 50 iterations. For such small numbers of iterations, effects such as broken ergodicity [36] are to be expected. Broken ergodicity refers to the property of a system to be mathematically ergodic but to remain in sub-areas of the phase space for arbitrarily long periods of time.

Given this state of affairs, computation will remain one of the tools to be used, both for research and for real-world applications. Real-world applications are typically large scale, with on the order of 10 million travelers. This means that also some of the research should be done with large-scale systems, both in order to understand the computational problems, and to check if results obtained for smaller-scale systems also hold for those larger-scale systems. The main focus of our research will be: (1) exploration of computational problems and solutions for large scale problems; and (2) exploration of the general dynamics of a multi-agent learning system.

The paper is organized as follows: We start by reviewing parallel computing aspects for transportation simulations (Sec. 2). In Sec. 3, possible traffic models are discussed, and the selection of the queue model is justified. Details of our queue model implementation are discussed in Sec. 4. Sec. 5 then describes the parallel computing implementation of the queue model, and reports computational speed results. The paper is concluded by a discussion and a summary.

This paper is essentially a copy of [12], except that the performance values are improved in the present version, from a real-time-ratio of maximally 200 to nearly 800 in the present paper. The contents of the present paper are also similar to [3], but computational performance numbers were even lower then, and also referred to an artificial scenario (the “gotthard” scenario) rather than to the realistic one used in the present paper.

2. Parallel Computing For Transportation Simulations

2.1 Discussion of Parallel Computing with a Special View Toward Transportation Simulation

Ultimately, we are interested in the agent-based simulation of large scale transportation scenarios. A typical scenario would be the 24-hour (about 10^5 seconds) simulation of a metropolitan area consisting of 10 million travelers. Typical computational speeds of micro-simulations with 1-second update steps are 100 000 vehicles in real time [33, 31, 42]. This results in a computation time of $10^5 \times 10^7/10^5 = 10^7$ seconds ≈ 100 days. This number is just a rough estimate and subject to the following changes: Increases in CPU speed will reduce the number; more realistic driving logic will increase the number; smaller time steps [38, 47] will increase the number.

This means that such a traffic simulation is too slow for practical or academic treatment of large scale problems. In addition, computer time is needed for activity generation, route generation, learning, etc. In consequence, it makes sense to explore parallel/distributed computing as an option.

The idea behind parallel computing is that a task can be achieved faster if it is divided into a set of subtasks, each of which is assigned to a different processor. A possible parallel computation environment is a cluster of standard Pentium computers, coupled via standard 100 Mbit Ethernet LAN (Local Area Network). In order to generate a parallel program, one must think about (i) how to partition the tasks into subtasks, and (ii) how to provide the data exchange between the subtasks. Since (i) depends on (ii), the discussion will be started with (ii).

With respect to communication, there are in general two main approaches to inter-processor communication. One of them is called *message passing* between processors; its alternative is to use *shared-address space*, where variables are kept in a common pool where they are globally available to all processors. Each paradigm has its own advantages and disadvantages.

In the shared-address space approach, all variables are globally accessible by all processors. Despite multiple processors operating independently, they share the same memory resources. The shared-address space approach makes it simpler for the user to achieve parallelism but since the memory bandwidth is limited, severe bottlenecks are unavoidable with an increasing number of processors, or alternatively such shared memory parallel computers become very expensive. For those reasons, our work concentrates on message passing.

In the message passing approach, there are independent cooperating processors. Each processor has a private local memory in order to keep the variables and data, and thus can access local data very rapidly. If an exchange of the information is needed between the processors, the processors communicate and synchronize by passing messages which are simple *send* and *receive* instructions. Message passing can be imagined to be similar to sending a letter. The following phases happen during a message passing operation.

1. The message needs to be packed. Here, one tells the computer which data needs to be sent.
2. The message is sent away.
3. The message then may take some time on the network until it finally arrives in the receiver's

inbox.

4. The receiver has to officially receive the message, i.e. to take it out of the inbox.
5. The receiver has to unpack the message and tell the computer where to store the received data.

There are time delays associated with each of these phases. It is important to note that some of these time delays are incurred even for an empty message ("latency"), whereas others depend on the size of the message ("bandwidth restriction"). We will come back to this later.

The communication among the processors can be achieved by using a message passing library which provides the functions to send and receive data. There are several libraries such as MPI [30] (Message Passing Interface) or PVM [37] (Parallel Virtual Machine) for this purpose. Both PVM and MPI are software packages/libraries that allow heterogeneous PCs interconnected by a computer network to exchange data. They both define an interface for the different programming languages such as C/C++ or Fortran. For the purposes of parallel traffic simulation, the differences between PVM and MPI are negligible; we use MPI since it has slightly more focus on computational performance. – In principle, CORBA (Common Object Request Broker Architecture; www.corba.org) would be an alternative to MPI or PVM, in particular for task parallelization (see below); in practice, our own and other people's experiences are that it is difficult to use and because of the strict client-server paradigm is not well suited to our simulations, which assume that all tasks are on equal hierarchical levels.

Because of cost/benefit reasons, our work concentrates on clusters of coupled PCs. We expect this to be the dominant technology in the area for many years to come. Near the end of this paper, it will be explored what performance gains can be expected of when improving the communication hardware via Myrinet. Two general strategies are possible for parallelization on such an architecture:

- **Task parallelization** – The different modules of a transportation simulation package (traffic simulation, routing, activities generation, learning, pre-/postprocessing) are run on different computers. This approach is for example used by DYNAMIT [1] or DYNASMART [2].

  The advantage of this approach is that it is conceptually straightforward, and fairly insensitive to network bottlenecks. The disadvantage with this approach is that the slowest module will dominate the computing speed – for example, if the traffic-simulation is using up most of the computing time, then task parallelization of the modules will not help.

- **Domain decomposition** – In this approach, each module is distributed across several CPUs. In fact, for most of the modules, this is straightforward since in current practical implementations activity generation, route generation, and learning are done for each traveler separately. Only the traffic simulation has tight interaction between the travelers. This will be considered in the following.

For clusters of PCs, the most costly communication operation is the initiation of a message ("latency"). In consequence, one needs to minimize the number of CPUs that need to communicate with each other. This is achieved with a *domain decomposition* (see Fig. 3(c)) of the traffic network graph. As long as the domains remain compact, each CPU will in the average have

at most six neighbors (Euler's theorem for planar graphs). Since network graphs are irregular structures, one needs a method to deal with this irregularity. METIS [29] is a software package that specifically deals with decomposing graphs for parallel computation.

The quality of the graph decomposition has consequences for parallel efficiency (*load balancing*): If one CPU has a lot more work to do than all other CPUs, then all other CPUs will need wait for it, which is inefficient. For our current work with $\sim$ 100 CPUs and networks with $\sim$ 20 000 links, the "latency problem" (see below) always dominates load balancing issues; however it is generally useful to use the actual computational load per network entity for the graph decomposition [32].

For shared memory machines, other forms of parallelization become possible, for example based on individual network links or individual travelers. One could have a dispatcher that distributes links for computation in a round-robin fashion to the CPUs of the shared memory machine [26]; technically, one would use threads [43] for this. This would be called *fine-grained parallelism*, as opposed to the *coarse-grained parallelism* that is more appropriate for message passing architectures. As said before, the main drawback of this method is that one needs an expensive machine if one wants to use large numbers of CPUs.

Important numbers for parallel implementations are real time ratio, speed-up, and efficiency:

- **Real time ratio (RTR)** – describes how much faster than reality the simulation is. For example, an RTR of 100 means that 100 minutes of traffic are simulated in 1 minute of computing time. This number is important no matter if the simulation is parallel or not.
- **Speed-up** – describes how much faster the parallel simulation is when compared to a simulation on a single CPU. For the single CPU algorithm one either uses the parallel algorithm running on a single CPU, or an algorithm specifically tailored to a single CPU system.
- **Efficiency** – is obtained by dividing Speed-up by the number p of CPUs that were used.

These numbers are all related, but they carry different meanings. As we will describe in more detail later, the main restriction that we are up against is the latency of the Ethernet communication hardware. Under the assumption of 1-second time-steps and 2 message exchanges per time step, this latency imposes a limit on the real time ratio of about 150. That is, *no matter what kind of simulation model one uses, how good the load balancing is, or how many CPUs one adds, that number is a fixed quantity* for this type of simulation and this type of parallel computer.

2.2 Parallel Computing in Transportation Simulations

Parallel computing has been employed in several transportation simulation projects. One of the first was PARAMICS [8], which started as a high performance computing project on a Connection Machine CM-2. In order to fit the specific architecture of that machine, cars/travelers were not truly objects but particles with a limited amount of internal state information. PARAMICS was later ported to a CM-5, where it was simultaneously made more object-oriented. In ref. [9], a computational speed of 120 000 vehicles with an RTR of 3 is reported, on 32 CPUs of a Cray T3E.

About at the same time, our own work showed that on coupled workstation architectures it was possible to efficiently implement vehicles in object-like fashion, and a parallel computing prototype with "intelligent" vehicles was written [31]. This later resulted in the research code PAMINA [42], which was the technical basis for the parallel version of TRANSIMS [32]. In our tests (with Ethernet only, on a network with 20 000 links, about 100 000 vehicles simultaneously in the simulation), TRANSIMS ran about 10 times faster than real time with the default parameters, and about 65 times faster than real time after tuning (Fig. 1. These numbers refer to 32 CPUs; adding more CPUs did not yield further improvement. The parallel concepts behind TRANSIMS are the same as behind the queue model from this paper, and in consequence TRANSIMS is up against the same latency problem as the queue model. However, for unknown reasons the computational speed is a factor of two to three smaller than predicted by latency alone.

Some other early implementations of parallel traffic simulations are [13, 34]. A parallel implementation of AIMSUN reports a speed-up of 3.5 on 8 CPUS using threads (which uses the shared memory technology as explained above) [20, 5].

DYNEMO is a macroparticle model, similar to DYNASMART described below. A parallel version was implemented about two years ago [35]. A speed-up of 15 on 19 CPUs connected by 100 Mbit Ethernet was reported on a traffic network of Berlin and Brandenburg with 13738 links. Larger numbers of CPUs were reported to be inefficient. The speed-up corresponded to a real time ratio of 6, implying that it was not the latency that caused the inefficiency, since the latency inefficiency should not set in before the RTR reaches 150 as explained several times in this text.

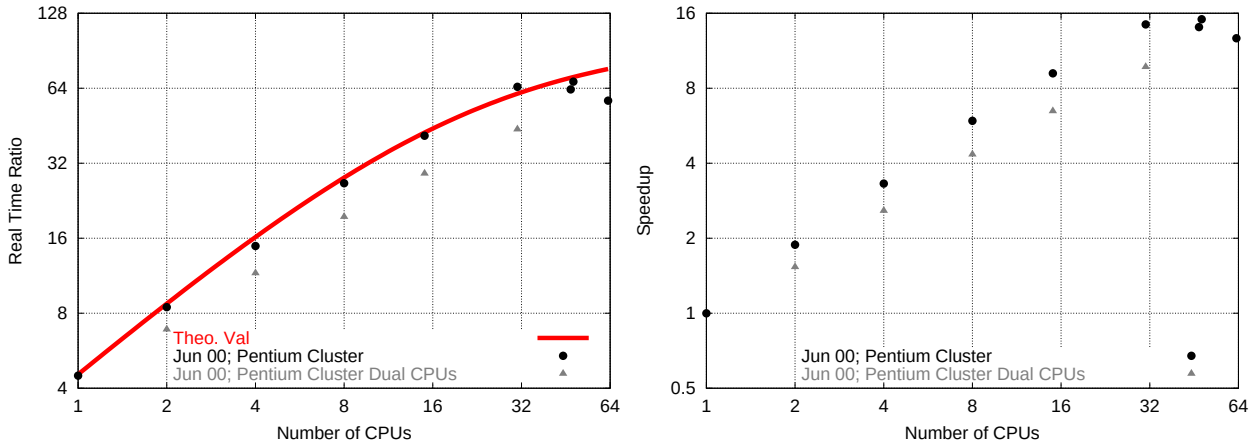
DynaMIT uses functional decomposition (task parallelization) as a parallelization concept [1]. This means that different modules, such as the router, the traffic (supply) simulation, the demand estimation, etc., can be run in parallel, but the traffic (supply) simulation runs on a single CPU only. Functional decomposition is outside the scope of this paper. DYNASMART also reports the intention to implement functional decomposition [2].

One should note that in terms of raw simulation speed, the values that will be presented in this paper are more than an order of magnitude faster than anything listed above. And this is not achieved by a smaller scenario size, but by diligent model selection, efficient implementation, and by hardware improvements based on knowing where the computational bottlenecks are. That is, our approach makes it possible to do very large scale scenarios as everyday research topics, rather than to have them as the result of very compute intensive studies only.

3. Possible Traffic Models

From the previous discussions it follows that we want the following conditions to be fulfilled for a traffic simulation:

- The model should have individual travelers/vehicles in order to be consistent with all agent-oriented learning approaches.
- The model should be simple in order to be comparable with static assignment and in order to allow concentration on computational issues rather than modeling issues.



(a) (b)

Figure 1: ***(a)* RTR and *(b)* Speedup curves of Portland EMME/2 Network. The results are from TRANSIMS simulator.**

This includes the fact that in the future we want to be able to include task parallelization into the software.

- The model should be computationally fast so that scenarios of a meaningful size can be run within acceptable periods of time.

For our own work, we also state a third condition:

- The model should be somewhat realistic so that meaningful comparisons to real-world results can be made.

These conditions make the use of existing software packages, such as DYNAMIT, DYNASMART, or TRANSIMS, difficult, since these software packages are already fairly complex and complicated. An alternative is the selection of a simple model for large scale microscopic network simulations, and to re-implement it. If one wants queue spill-back, there are essentially two starting points: queuing theory, and the theory of kinematic waves.

In queuing theory, one can build networks of queues and servers ([46, 14, 44]). Packets enter the network at an arbitrary queue. Once in a queue, they wait, typically in a first-in first-out (FIFO) queue until they are served; servers serve queues with a given rate. Once the packet was served, it will enter the next queue.

This can be directly applied to traffic, where packets correspond to vehicles, queues correspond to links, and serving rates correspond to capacity. The decision which link to enter after a vehicle was served is given by the vehicle's route plan.

A shortcoming of this type of approach is that it does not model spill-back. If queues have size restrictions, then packets exceeding that restriction are typically dropped [46]. Since this is not realistic for traffic, an alternative is to refuse further acceptance of vehicles once the queue is full (*"physical queues"*). This however means that the serving rate of the upstream server is influenced by a full queue downstream. Such a model was for example used by Gawron [22]. A detailed algorithmic description is given in Fig. 2(a).

An important issue with physical queues is that the intersection logic needs to be adapted. Since without physical queues (i.e. with "point queues") the outgoing links can always accept all incoming vehicles, the maximum flow through each incoming link is just given by each link's capacity. However, when outgoing links have limited space, then that space needs to be distributed to the incoming links which compete for it.

In the original algorithm (Fig. 2(a)), links are processed in arbitrary but fixed sequence. This has the consequence that the most favoured link in a given intersection is the one that is processed next after the congested outgoing link has been processed. This could for example mean that a small side road obtains priority over a large main road.

A better way to do this is to allocate flow under congested conditions according to capacity [16]. For example, if there are two incoming links with capacities 2 and 4 per time step, and the outgoing link has 3 spaces available, then 1 space should be allocated to the first incoming link and 2 to the second. Sec. 4.2 will explain in more detail how this is implemented.

A shortcoming of queue models is that the speed of the backwards traveling kinematic wave ("jam wave") is not correctly modeled. A vehicle that leaves the link at the downstream end *immediately* opens up a new space at the upstream end into which a new vehicle can enter, meaning that the kinematic wave speed is roughly one link per time step, rather than a realistic velocity. This becomes visible in the dissolution of jams, say at the end of a rush hour: If a queue extends over a sequence of links, then the jam should dissolve from the downstream end. In the queue model, it will essentially dissolve from the upstream end. More details of this, including schematic fundamental diagrams, can be found in [45, 22].

Despite this shortcoming, we will use the queue model for its simplicity. This point will be revisited in the discussion.

4. Queue Model

4.1 Original Queue Model

Each link has, from the input files, the attributes free flow velocity v_0, length L, capacity C and number of lanes n_{lanes}. Free flow travel time is calculated by $T_0 = L/v_0$. The storage constraint of a link is calculated as $N_{sites} = L \cdot n_{lanes}/\ell$, where ℓ is the space a single vehicle in the average occupies in a jam, which is the inverse of the jam density. We use $\ell = 7.5$ m.

The intersection logic by Gawron [23] is that all links are processed in arbitrary but fixed sequence, and a vehicle is moved to the next link if (1) it has arrived at the end of the link, (2) it can be moved according to capacity, and (3) there is space on the destination link (Fig. 2(a)). The three conditions mean the following:

- A vehicle that enters link a at time t_0 cannot leave the link before time $t_0 + T_0$, where T_0 is the free speed link travel time as explained above.
- The condition "vehicle can be moved according to capacity" is determined as

$$N < C \;\text{ or }\; \Big(N = C \;\text{ and }\; rnd < f\Big)$$

where C is the integer part of the capacity of the link (in vehicles per time step), f is the fractional part of the capacity of the link, and N is the number of the vehicles which already left the same link in the same time step. rnd is a random number such that $0 \leq rnd \leq 1$. According to this formula, vehicles can leave the link if the leaving capacity of the link has not yet been exhausted for this time step. If the capacity per time step is non-integer, then we move the last vehicle with a probability which is equal to the non-integer part of the capacity per time step.

- "Space on destination link": If the destination link is full, the vehicle will not move across the intersection.

4.2 Fair Intersections and Parallel Update

As discussed earlier, the problem with this algorithm is that links are always selected in the same sequence, thus giving some links a higher priority than others under congested conditions. One can modify the algorithm so that links are prioritized randomly proportional to capacity. That is, links with high capacity are more often first than links with low capacity. In contrast to Ref. [6], we serve *all* vehicles of an incoming link once that link is selected.

To make this work, the algorithm was moved from link-oriented to intersection-oriented (that is, the loop now goes over all intersections, which then look at all incoming links), and we have separated the flow capacity constraint from the intersection logic. The latter was done by introducing a separate buffer (see Fig. 3(a)) at the end of the link, which is of size $\lceil C_{link} \rceil$, i.e. the first integer number being larger or equal than the link capacity (in vehicles per time step). Vehicles are then moved from the link into the buffer according to the capacity constraint *and* only if there is space in the buffer; once in the buffer, vehicles can be moved across intersections without looking at the flow capacity constraints. This approach is borrowed from lattice gas automata, where particle movements are also separated into a "propagate" and a "scatter" step [21].

As a desired side effect, this makes the update in the algorithm completely parallel: If traffic is moved out of a full link, the new empty space will only open in the buffer and not on the link, and will thus not become available at the upstream intersection until the *next* time step – at which time it will be shared between the incoming links according to the method described above. This has the advantage that all information which is necessary for the computation of a time step is available locally at each intersection before a time step starts – and in consequence there is no information exchange between intersections *during* the computation of the time step. Further details are given in algorithmic form in Fig. 2(b).

In order to systematically test this intersection logic, an intersection test suite was implemented. This test suite goes through several different intersection layouts and tests them one by one to see if the dynamics behaves according to the specifications. The results typically look as shown in Fig. 3(b). In this particular example, one link with 500veh/sec and one link with 2000veh/sec merge into a link with a capacity of 500veh/sec. The curves are, for different algorithms, time-dependent accumulative vehicle numbers for the two incoming links. In this case, one sees that until approx time-step 3400, both links discharge at rates 400 and 100veh/sec, respectively. After that time, the first link is empty, and the second link now discharges at 500veh/sec. Not all algorithms are similarly faithful in generating the desired dynamics; the thick black lines denote results from the algorithm that got finally implemented. For further details, see [7].

Algorithm A – Handling Constraints - Original algorithm

```
for all links do
  while vehicle has arrived at end of link
  AND vehicle can be moved according to capacity
  AND there is space on destination link do
    move vehicle to next link
  end while
end for
```

(a)

Algorithm B – Links and Intersections separated:

```
// Propagate vehicles along links:
for all links do
  while vehicle has arrived at end of link
  AND vehicle can be moved according to capacity
  AND there is space in the buffer (see Fig 3(a)) do
    move vehicle from link to buffer
  end while
end for
// Move vehicles across intersections:
for all nodes do
  while there are still eligible links do
    Select an eligible link randomly proportional to capacity
    Mark link as non-eligible
    while there are vehicles in the buffer of that link do
      Check the first vehicle in the buffer of the link
      if the destination link has space then
        Move vehicle from buffer to destination link
      end if
    end while
  end while
end for
```

(b)

Algorithm C – Parallel computing implementation

```
According to Alg. B, propagate vehicles along links.
for all split links do
  SEND the number of empty spaces of the link to the other processor.
end for
for all split links do
  RECEIVE the number of empty spaces of the link from the other processor.
end for
According to Alg. B, move vehicles across intersections.
for  all split links  do
  SEND vehicles which just entered a split link to the other processor
end for
for  all split links  do
  RECEIVE the vehicles (if any) from the neighbor at the other end of the link.
  place these vehicles into the local queues.
end for
```

(c)

Figure 2: ***(a)*** **The original queue model.** ***(b)*** **Vehicle movement at the intersections. Note that the algorithm separates the flow capacity from intersection dynamics.** ***(c)*** **Parallel implementation of queue model.**

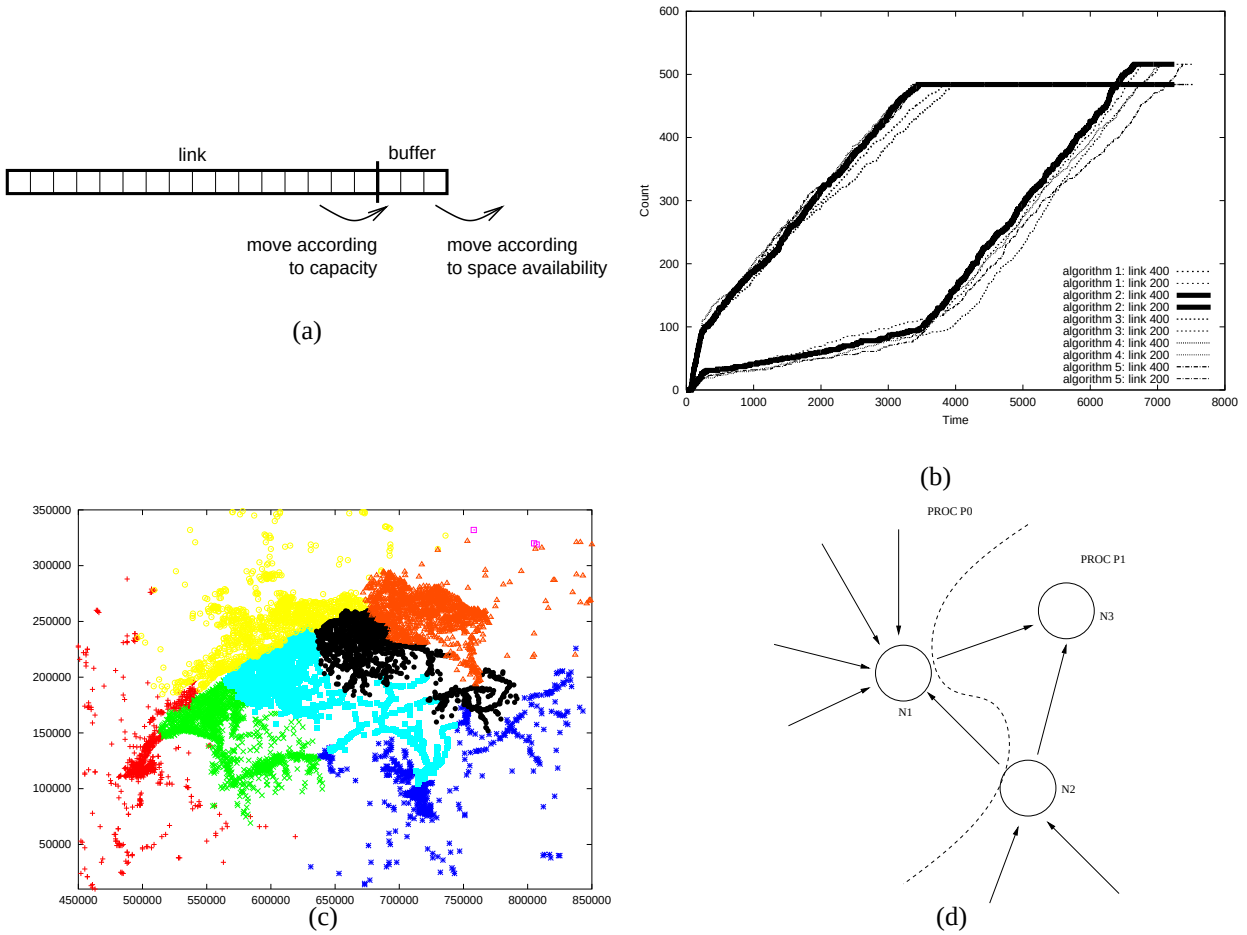


Figure 3: ***(a)* The separation of flow capacity from intersection dynamics. *(b)* Test suite results for intersection dynamics. The curves show the number of discharging vehicles from two incoming links as explained in section 4.2. *(c)* Decomposition of the Switzerland street network. Each color corresponds to a different processor. *(d)* Communication between nodes in the boundaries: Node N1 needs to communicate with N2 and N3. Since N2 and N3 are on the same processor, they do not need to establish a communication between themselves.**

5. Parallel Computing of the Queue Model

5.1 Parallel Implementation

As was discussed above, the parallel target architecture for our transportation micro-simulation is a PC cluster. As also discussed, the suitable approach for this architecture is domain decomposition, i.e. to decompose the traffic network graph into several pieces, and to give each piece to a different CPU. Information exchange between CPUs is achieved via messages.

Next, one needs to decide where to split the graph, and how to achieve the message exchange. Both questions can only be answered with respect to a particular traffic model, which is why they were not discussed previously. Nevertheless, lessons learned here can be used for other models.

In general one wants to split as far away from the intersection as possible. This implies that one

should split links in the middle, as for example TRANSIMS in fact does [32]. However, for the queue model "middle of the link" does not make sense since there is no real representation of space. In consequence, one can either split at the downstream end, or at the upstream end of the link. The downstream end is undesirable because vehicles driving *towards* an intersection are more influenced by the intersection than vehicles driving *away* from an intersection. For that reason, in the queue simulation we split them right after the intersection (Fig. 3(d)).

With respect to message passing, this implies that a CPU that "owns" a split link reports, via a message, the number of empty spaces to the CPU which "owns" the intersection from which vehicles can enter the link. After this, the intersection update can be done in parallel for all intersections. Next, a CPU that "owns" an intersection reports, via a message, the vehicles that have moved to the CPUs which "own" the outgoing links. See Fig. 2(c) for pseudo-code of how this is implemented using message passing. In fact, Algorithm B and Algorithm C together give the whole pseudo-code for the queue model traffic logic. For efficiency reasons, all messages to the same CPU at the same time should be merged into a single message in order to incur the latency overhead only once.

5.2 Performance Issues

Once the precise implementation of the parallelization is known, it is possible to predict the parallel performance. This has been done in detail in Ref. [32]. Since that reference refers to TRANSIMS instead of to the queue model, and since we also have results regarding Myrinet (www.myri.com), the most important aspects will be repeated here, with special focus towards the specific problems encountered in the current situation.

The execution time of a parallel program is defined as the total time elapsed from the time the first processor starts execution to the time the last processor completes the execution. During execution, on a PC cluster, each processor is either computing or communicating. Therefore,

$$T(p) = T_{cmp}(p) + T_{cmm}(p), \tag{1}$$

where T is the execution time, p is the number of processors, T_{cmp} is the computation time and T_{cmm} is the communication time.

For traffic simulation, the time required for the computation, T_{cmp} can be calculated roughly in terms of the run time of the computation on a single CPU divided by the number of processors. Thus,

$$T_{cmp}(p) \approx \frac{T_{cmp}(1)}{p}, \tag{2}$$

where p is the number of CPUs. More exact formulas would also contain the overhead effects and unequal domain size effects ("load balancing").

In deviation from Ref. [32], we now assume that the time for communication is well approximated by the latency only; this is in fact a good approximation for our system. Latency is incurred for each message that is sent. Since it is possible to pack all messages to the same processor into one message, one obtains

$$T_{lt} = N_{nb}\, t_{lt} \ ,$$

where N_{nb} is the number of neighboring processors (i.e. the processors which are reached via common split links), and t_{lt} is the latency per message.

The number of neighbors is zero at $p = 1$, and goes, for contiguous domains, to an average of six for $p \to \infty$. An interpolating formula is

$$N_{nb}(p) = 2\,(3\sqrt{p} - 1)\,(\sqrt{p} - 1)/p\,, \tag{3}$$

Latency of 100 Mbit Ethernet cards is about 0.5 ms; each processor sends messages twice per time step to all neighbors resulting in ~ 12 latency contributions or $6\,ms$ per time step. In other words, the cluster can maximally do $1000/6 = 167$ time steps per second; in practice we find 174 with 64 CPUs, see Sec. 5.3. If the time step of a simulation is one second, then this is also the maximum real time ratio of the parallel simulation, i.e. the number which says how much faster than reality the computer is. *Note that the limiting value does not depend on the problem size or on the speed of the algorithm; it is a limiting number for any parallel computation of a 2-dimensional system on a Beowulf cluster using Ethernet LAN.*

The only way this number can be improved under the assumptions that we made is to use faster communication hardware. Gbit Ethernet hardware is faster, but standard driver implementations give away that advantage [27]. In contrast, Myrinet (see www.myri.com) is a communication technology specifically designed for this situation. Interestingly, as we will see later, it will be possible to recoup the cost for a Myrinet network by being able to work with a smaller cluster.

5.3 Computational Performance

The parallel queue model is used as the traffic micro-simulation module within the project of a microscopic and activity-based simulation of all of Switzerland. In this paper, we only report computational performance results; validation results with respect to a real world scenario are reported elsewhere [39]. Also note that our queue model is fully calibrated and validated in terms of free speed link travel times and link flow capacity, since these numbers are taken from the input files and implemented exactly. The same would be true for link storage capacity if those numbers were available. It was described earlier (Sec. 4.2) how intersection priorities are modeled, and how the implementation was verified.

The following performance numbers refer to the morning rush hour in a road network with 10 564 nodes and 28 624 links. Demand contains *all* car trips in Switzerland which start between 6:00AM and 9:00AM; this results in 991471 trips. The largest number of vehicles simultaneously in the simulation is 162464 vehicles at 8:00AM. This is also called the "ch6-9" scenario.

The most important plot is Fig. 4(a). It shows our newest (and fastest) computational real time ratio (RTR) numbers as a function of the number of CPUs. One notices that, with 64 CPUs, we reach an RTR of nearly 800. *This means that we can simulate 24 hours of all car traffic in Switzerland less than two minutes!* This performance is achieved with Myrinet communication hardware; with 100 Mbit Ethernet hardware, performance levels out at about 170 as predicted earlier. The plot also shows two different graphs for achieving the performance with single-CPU or with dual-CPU machines; one notices that there are differences but they are less important.

To the right of this, in Fig. 4(b), the corresponding speed-up curves are shown. In a log-log plot, the speed-up curve can be obtained from the RTR curve by a simple vertical shift; this vertical shift corresponds to a division by the RTR of the single-CPU version of the simulation, which is about 5 here. Speed-up curves put more emphasis on the efficiency of the implementation and

less emphasis on the absolute speed. In our case, we reach super-linear speed-up, that is, the computational speed grows *faster* than the number of CPUs. This is due to the fact that for small numbers of CPUs, the simulation does not fit into computer memory, which is 1 GByte. For larger numbers of CPUs, some of the memory requirements get distributed across the CPUs, and so the memory requirement per CPU decreases.

Obviously, the speed-up curves show the same performance saturation for Ethernet as do the RTR curves. It should be noted that, while the level of saturation in the RTR curves is universal for this computer architecture and a one-second time step, for the speed-up curves this depends on the scenario size. Larger scenarios will reach higher speed-up, but saturate at the same RTR. For that reason, we consider the RTR number as more important than the speed-up.

The other curves in this figure show earlier performance numbers. Figs. 4(c) and 4(d) show the results of [12]. The curves are similar, but the performance is about 2-4 times slower. The reason is an improved implementation of the simulation logic in the newer results, in particular with respect to function inlining, etc.

Figs. 4(e) and 4(f) show the numbers that were presented last year at STRC [11]. In spite of a smaller scenario, the RTR curves saturate at a much smaller value of 128. Worse, the speed-up did not get larger than 4, implying a very inefficient implementation. And indeed, in that implementation there was communication of every domain to every other domain, even if there was no relevant information to send. This means that number of messages per time step *per CPU* grows with p (where p is the number of CPUs), meaning that once the latency saturation sets in, performance *decreases* essentially as $\sim 1/p$. This is visible as convergence to a line with slope -1 in the log-log plots.

One could ask if a different domain decomposition might make a difference. It was already argued earlier that no difference is expected once the latency saturation sets in. Nevertheless, in earlier investigations we had found that there were some differences when using so-called multi-constraint partitioning in METIS when applied to the "gotthard" scenario [41]. Yet, for the ch6-9 scenario, multi-constraint partitioning does not yield systematic improvement (not shown). Since this is consistent with the theoretical expectation that it should not yield a systematic improvement for spatially homogeneous scenarios, a further treatment of this is moved to the appendix.

It is interesting to compare two different hardware configurations:

- 64 single CPU machines using 100 Mbit LAN.

  Real time ratio 170.

  Cost approx $64 \times \$2k = \$128k$ for the machines plus approx $\$20k$ for a full bandwidth switch, resulting in $\$148k$ overall.

- 32 dual CPU machines using Myrinet.

  Real time ratio 770.

  Cost approx $32 \times \$4.5k = \$144k$, Myrinet included.

That is, the Myrinet set-up is not only considerably faster, but somewhat unexpectedly also cheaper. A Myrinet setup has the additional advantage that smaller scenarios than the one dis-

cussed will run even faster, whereas on the Ethernet cluster smaller scenarios will run with the same computational speed as the large scenarios.

6. Discussion

6.1 Traffic Model

Two arguments against the queue model often are that the intersection behavior is "unfair" in standard implementations, and that the speed of the backwards traveling jam ("kinematic") wave is incorrectly modeled. The first problem was overcome by a better modeling of the intersection logic, as described in Sec. 4.2. The second problem still remains. What can be done about it?

If one wants to avoid a detailed micro-simulation as is done in TRANSIMS for example, then a possible solution is to use what is sometimes called "mesoscopic models" or "smoothed particle hydrodynamics" [25]. The idea is to have individual particles in the simulation, but have them moved by aggregate equations of motion. These equations of motion should be selected so that in the fluid-dynamical limit the Lighhill-Whitham-Richards [28] equation is recovered [15].

The number of vehicles in a segment is updated according to

$$N(x,t+1) = N(x,t) + Q(x-\frac{1}{2},t) - Q(x+\frac{1}{2},t) + g(x,t) \;, \tag{4}$$

where $N(x,t)$ is the number of vehicles in segment x at time t, $Q(x-\frac{1}{2},t)$ is the flow of vehicles from segment $x-1$ into segment x at time t, and $g(x,t)$ is the source/sink term given by entry and exit rates.

What is missing is the specification of the flow rates $Q(x,t)$. A possible specification is given by the cell transmission model [15]:

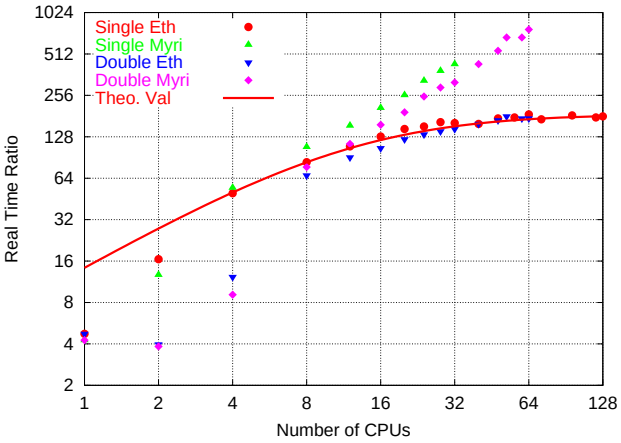
$$Q(x-\frac{1}{2},t) = \max[v_f\, N(x-1,t), c\,(N_{max} - N(x,t)), Q_{max}] \;, \tag{5}$$

where Q_{max} is the capacity constraint, c is the jam wave speed, v_f is the free speed, N_{max} is the maximum number of vehicles on the link and all other variables have the same meaning as before.

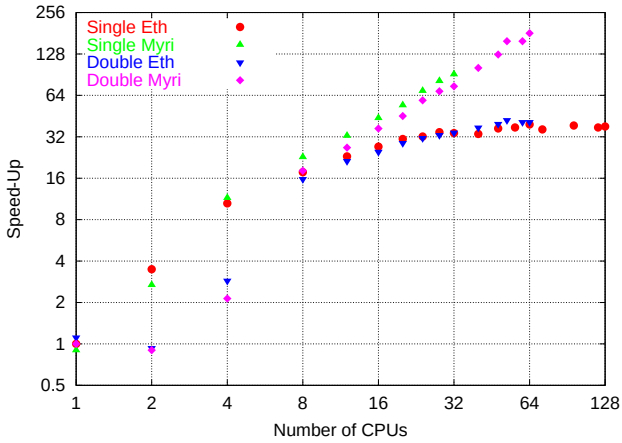
Note that this now exactly enforces the storage constraint by setting $Q(x-\frac{1}{2},t)$ to zero once $N(x,t)$ has reached N_{max}. In addition, the kinematic jam wave speed is given explicitely via c. There is some interaction between length of a segment, time step, and c that needs to be considered. The network version of the cell transmission model [16] also specifies how to implement fair intersections. The cell transmission model is implemented under the name NETCELL.

Other link dynamics are, for example, provided by DYNAMIT [18] or DYNASMART [19]. These are based on the same mass conservation equation as Eq. (4), but use different specifications for $Q(x)$. In fact, they calculate vehicle speeds at the time of entry into the segment depending on the number of vehicles already in the segment. The number of vehicles that can potentially leave a link in a given time step is in consequence given indirectly via this speed computation. Since this is not enough to enforce physical queues, physical queueing restrictions are added to this description. A further description of these models goes beyond the scope of this paper.

Feb 2003:

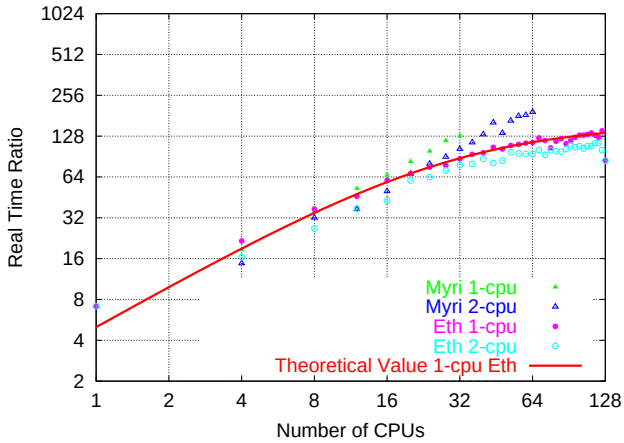


(a)

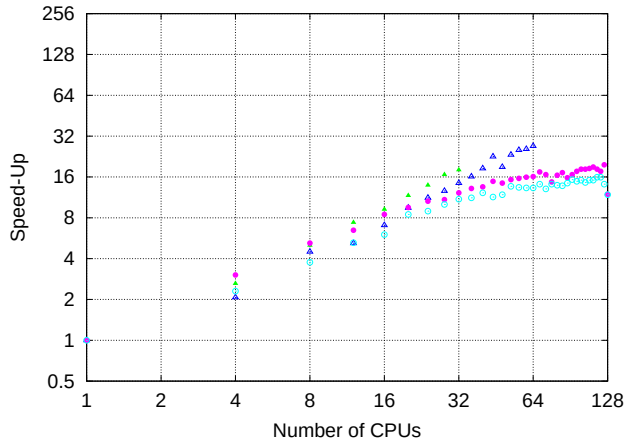


(b)

Nov 2002:

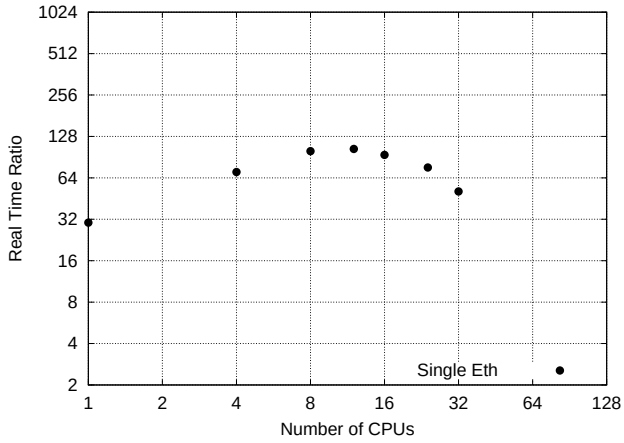


(c)

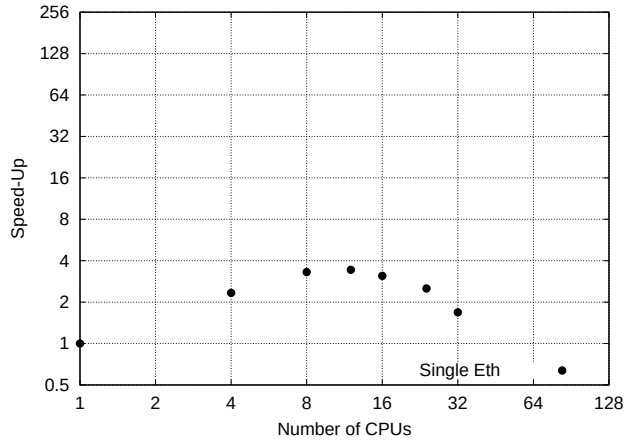


(d)

Mar 2002, also smaller scenario (“gotthard”):



(e)



(f)

Figure 4: ***(a)* RTR and *(b)* Speedup curves of the ch6-9 scenario from Feb 2003, with the default domain decomposition approach. *(c)* RTR and *(d)* Speedup curves of the ch6-9 scenario from Nov 2002, with the default domain decomposition approach. *(e)* RTR and *(f)* Speedup curves of Gotthard scenario (fewer vehicles than in the 6-9 scenario) in March 2002. These results are from a non-optimized and unscalable version of queue simulator.**

6.2 Parallel performance

Once more, the most important result of our investigations is that there is a natural limit to computational speed on parallel computers which use Ethernet as their communication medium, and that speed is about 150 updates per second. If a simulation uses 1-second time steps, then this translates into a real time ratio of about 150. We are not aware of any other traffic simulation which has approached this limit, which is probably the reason why it is not discussed more. It imposes important consequences both on real time and on large scale applications that need to be considered. And in contrast to other areas of computing, it seems that waiting for better comodity hardware will here *not* solve the problem: Latency, which is the technical reason for this limit, has not improved significantly over the last decade. Gbit Ethernet has not much better latency than 10 Mbit Ethernet.

One option to go beyond this limit is to use more expensive special purpose hardware. Such hardware is typically provided by computing centers, which operate dedicated parallel computers such as the Cray T3E, or the IBM SP2, or any of the ASCI (Advanced Strategic Computing Initiative) computers in the U.S. An intermediate solution is the use of Myrinet, which this paper shows to be an effective approach, both in terms of technology and in terms of monetary cost.

On the algorithmic side, the following options exist: First, for the queue simulation it is in fact possible to reduce the number of communication exchanges per time step from two to one. This should yield a factor of two in Speed-up. Next, in some cases, it may be possible to operate with time steps longer than one second. This should in particular be possible with the kinematic wave models, since in those models the backwards waves do no longer travel infinitely fast. The fastest time in such simulations would be given by the shortest free speed link travel time in the whole system. In addition, one could prohibit the simulation from splitting links with short free speed link travel time, leading to further improvement.

6.3 Task Parallelization

In Sec. 2, task parallelization was shortly discussed. There it was pointed out that this will not pay off if the traffic simulation poses the by far largest computational burden. However, after parallelizing the micro-simulation, this is no longer true. Task parallelization would mean that for example activity generator, router, and learning module would run in parallel with the traffic simulation. One way to implement this would be to not precompute plans any more, as is done in day-to-day simulations, but to request them just before the traveler starts. A nice side-effect of this would be that such an architecture would also allow within-day replanning without any further computational re-design. We are in fact working on such an implementation.

7. Summary

A simple queue model for traffic flow was investigated. The main difference to queue*ing* theory and to some dynamic extensions of static assignment is a hard storage constraint on the link; this generates spill-back. In contrast, the queue model is still simpler than the flow models of DYNASMART, DynaMIT, and the cell transmission model: The queue model does not divide

links into segments. The disadvantage of this is that jam wave speeds are no longer realistically modeled; the advantage is higher computational performance.

Hard storage constraints have the consequence that link outflows are constrained both by the flow capacity of the link itself and by space limitations on the receiving link. In contrast to earlier versions of the queue model, we implemented "fair" intersections, where, if space on the outgoing links is limited, that space is allocated proportional to the incoming links' capacity.

The intention of this work is threefold: (1) Investigate minimal extensions of static assignment. (2) Investigate parallel computing limitations of transportation simulations in general. (3) Use the queue model as a simpler and faster alternative to TRANSIMS-like micro-simulations in activity-based transportation modelling.

With respect to computing, it was demonstrated that affordable Beowulf clusters (clusters of Pentium computers with Linux operating system) can be used successfully for large scale problems. The latency of Ethernet communication sets a hard limit on computing speed to about 150 simulation time steps per second, no matter what the problem size. It was shows that the use of Myrinet communications technology overcomes that problem, and at no higher cost, since with Myrinet one can reach better computing speeds with a smaller numbers of computers than with Ethernet. A speed-up of 180 was reached on 64 CPUs, meaning in practical terms that a month of computing can be reduced to about a day.

The model was applied successfully to a dynamic traffic assignment simulation of car traffic of the whole country of Switzerland for the morning peak. The results of this are reported elsewhere [39, 40], including comparison to a VISUM assignment result and to field volume data.

Acknowledgments

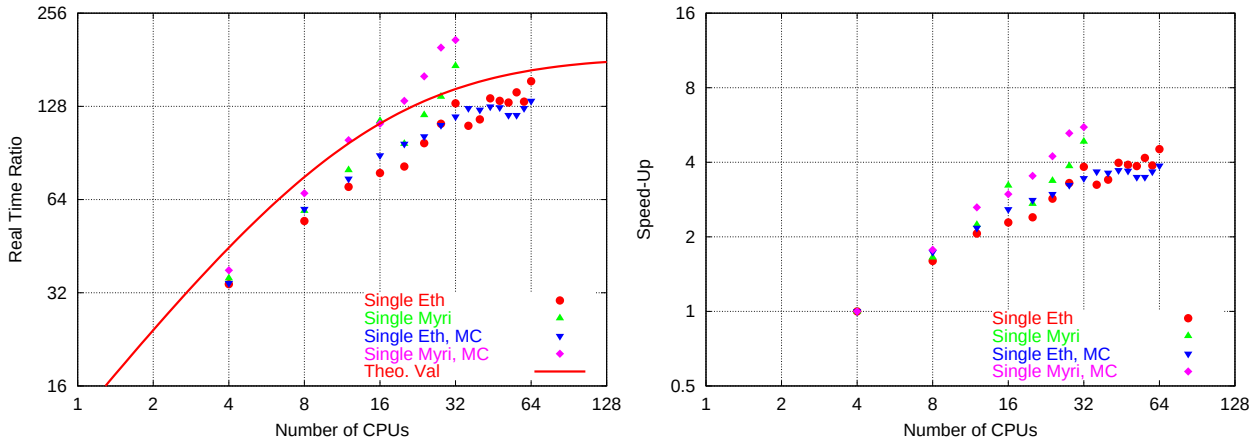
We thank ETHZ and the Department of Computer Science for making the Beowulf Cluster Xibalba including its Myrinet partition available to us. Marc Schmitt does a great job in maintaining the Linux environment of the cluster. This work was funded by ETHZ core funding and by the ETHZ project "Large scale multi-agent simulation of travel behavior and traffic flow".

A. Multi-constraint partitioning

In the "gotthard" scenario, 50 000 travelers, starting at random locations all over Switzerland, attempt to travel to Lugano as quickly as possible. The scenario is meant for testing the route assignment logic with respect to route equilibration in a single destination scenario. It was also used as a test scenario for parallel computing.

Partitioning in our case refers to the nodes of the road network. Since the work of the queue simulation mostly consists of computing the intersection dynamics, computational load is essentially proportional to the number of intersections. Standard partitioning thus attempts to spread the network nodes equally across all CPUs while maintaining contiguous domains.

Multi-constraint partitioning means that there is more than one weight per node. In our case,



(a) (b)

Figure 5: ***(a)* RTR and *(b)* Speedup curves of a 6-hour run of Gotthard scenario without and with multi-constraint ("MC") domain decomposition approach.**

this can be best understood by assuming that those weights refer to different time slices. For example, in the gotthard scenario, there is a lot of traffic all over Switzerland initially, but near the end of the simulation, only some nodes process traffic while all others are empty.

The question is if under such circumstances the network can be partitioned so that the load is equally balanced at all times. A counter-example would be a distribution where one CPU has nodes with a lot of traffic initially but no traffic later, and another CPU has no traffic initially, but a lot of traffic later. That simulation would run faster if both CPUs traded approximately half of their nodes: Then both CPUs would always be busy on about half of their nodes. This is exactly what multi-constraint partitioning attempts to achieve.

References

[1] Dynamit prototype description. See www.dynamictrafficassignment.org/dynamit.htm, 2000.

[2] Dynasmart-x prototype description. See www.dynamictrafficassignment.org/dsmart_x.htm, 2000.

[3] *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH, 2002. See www.strc.ch.

[4] V. Adamo, V. Astarita, M. Florian, M. Mahut, and J.H. Wu. A framework for introducing spillback in link based dynamic network loading models. In *Proceedings of TRISTAN III*, volume 2, San Juan, Puerto Rico, 1998.

[5] J. Barceló, J.L. Ferrer, D. García, and R. Grau. Microscopic traffic simulation for att systems analysis. a parallel computing version. Contribution to the 25th Aniversary of CRT, University of Montreal. See www.tss-bcn.com/documents.html, 1998.

[6] J.A. Bottom. *Consistent anticipatory route guidance*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2000.

[7] A. Burriad. Intersection dynamics in queue models. Term project report, Swiss Federal Institute of Technology, 2002. See sim.inf.ethz.ch/papers.

[8] G. D. B. Cameron and C. I. D. Duncan. PARAMICS — Parallel microscopic simulation of road traffic. In *J. Supercomputing* [9], page 25.

[9] G. D. B. Cameron and C. I. D. Duncan. PARAMICS — Parallel microscopic simulation of road traffic. *J. Supercomputing*, 10(1):25, 1996.

[10] C. Cantarella and E. Cascetta. Dynamic process and equilibrium in transportation network: Towards a unifying theory. *Transportation Science A*, 25(4):305–329, 1995.

[11] N. Cetin and K. Nagel. Parallel queue model approach to traffic microsimulations. In *Swiss Transport Research Conference*, Monte Verita, Switzerland, March 2002. See www.strc.ch.

[12] N. Cetin and K. Nagel. Parallel queue model approach to traffic microsimulations. Paper 03-4272, Transportation Research Board Annual Meeting, Washington, D.C., 2003. Also see sim.inf.ethz.ch/papers.

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

[14] Berksekas D. and R. Gallager. *Data Networks*. Prentice Hall, MA, U.S.A., 1991.

[15] C.F. Daganzo. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research B*, 28B(4):269, 1994.

[16] C.F. Daganzo. The cell transmission model, part II: Network traffic. *Transportation Research B*, 29B(2):79–93, 1995.

[17] A. de Palma and F. Marchal. Real case applications of the fully dynamic METROPOLIS tool-box: an advocacy for large-scale mesoscopic transportation systems. *Networks and Spatial Economics*, 2002.

[18] DYNAMIT. Massachusetts Institute of Technology, Cambridge, Massachusetts. See its.mit.edu. Also see dynamictrafficassignment.org.

[19] See www.dynasmart.com. Also see dynamictrafficassignment.org.

[20] J.L. Ferrer and J. Barceló. AIMSUN2: Advanced Interactive Microscopic Simulator for Urban and non-urban Networks. Internal report, Departamento de Estadística e Investigación Operativa, Facultad de Informática, Universitat Politècnica de Catalunya, 1993.

[21] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for navier-stokes equation. *Physical Review Letters*, 56:1505, 1986.

[22] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998.

[23] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998.

[24] C. Gawron. *Simulation-based traffic assignment*. PhD thesis, University of Cologne, Germany, 1998.

[25] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Royal Astronomical Society, Monthly Notices*, 181, 1977.

[26] P. Gonnet. A thread-based distributed traffic micro-simulation. Term project, Swiss Federal Institute of Technology ETH, Zürich, Switzerland, 2001.

[27] C. Kurmann, T. Stricker, and F. Rauch. Speculative defragmentation - leading gigabit ethernet to true zero-copy communication cluster computing. *Journal of Networks, Software Tools and Applications*, 4(4):7–18, 2001.

[28] M. J. Lighthill and J. B. Whitham. On kinematic waves. I: Flow movement in long rivers. II: A Theory of traffic flow on long crowded roads. *Proceedings of the Royal Society A*, 229:281–345, 1955.

[29] METIS library. www-users.cs.umn.edu/˜karypis/metis/.

[30] MPI: Message Passing Interface. See www-unix.mcs.anl.gov/mpi/mpich.

[31] K. Nagel. *High-speed microsimulations of traffic flow*. PhD thesis, University of Cologne, 1994/95. See www.inf.ethz.ch/˜nagel/papers.

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

[33] K. Nagel and A. Schleicher. Microscopic traffic modeling on parallel high performance computers. *Parallel Computing*, 20:125–146, 1994.

[34] W. Niedringhaus, J. Opper, L. Rhodes, and B. Hughes. IVHS traffic modeling using parallel computing: Performance results. In *Proceedings of the International Conference on Parallel Processing*, pages 688–693. IEEE, 1994.

[35] K. Nökel and M. Schmidt. Parallel DYNEMO: Mesoscopic traffic flow simulation on large networks. preprint, 2000.

[36] R. Palmer. Broken ergodicity. In D. L. Stein, editor, *Lectures in the Sciences of Complexity*, volume I of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 275–300. Addison-Wesley, 1989.

[37] PVM: Parallel Virtual Machine. See www.epm.ornl.gov/pvm/pvm_home.html.

[38] H. A. Rakha and M. W. Van Aerde. Comparison of simulation modules of TRANSYT and INTEGRATION models. *Transportation Research Record*, 1566:1–7, 1996.

[39] B. Raney, N. Cetin, A. Völlmy, and K. Nagel. Large scale multi-agent transportation simulations. In *Proceedings of the annual congress of the European Regional Science Association (ERSA)*, Dortmund, Germany, Aug 2002. Also see www.inf.ethz.ch/˜nagel/papers.

[40] B. Raney, N. Cetin, A. Völlmy, M. Vrtic, K. Axhausen, and K. Nagel. An agent-based microsimulation model of Swiss travel: First results. *Networks and Spatial Economics*, 3(1):23–41, 2003. Earlier version Transportation Research Board Annual Meeting 2003 paper number 03-4267.

[41] B. Raney and K. Nagel. Truly agent-based strategy selection for transportation simulations. Paper 03-4258, Transportation Research Board Annual Meeting, Washington, D.C., 2003. Also see sim.inf.ethz.ch/papers.

[42] M. Rickert. *Traffic simulation on distributed memory computers*. PhD thesis, University of Cologne, Germany, 1998. See www.zpr.uni-koeln.de/~mr/dissertation.

[43] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. John Wiley & Sons, Inc., 2001.

[44] H.P. Simão and W.B. Powell. Numerical methods for simulating transient, stochastic queueing networks. *Transportation Science*, 26:296, 1992.

[45] P. M. Simon and K. Nagel. Simple queueing model applied to the city of Portland. *International Journal of Modern Physics C*, 10(5):941–960, 1999.

[46] William Stallings. Queuing analysis. ftp://shell.shore.net/members/w/s/ws/Support/QueuingAnalysis.pdf, 2000.

[47] Planung Transport und Verkehr (PTV) GmbH. See www.ptv.de.