Real–Time Traffic Simulation of the German Autobahn Network

M. Rickert[a,b], P.Wagner[a], Ch. Gawron[a]

[a] *Center Of Parallel Computing, University Of Cologne, Germany*
[b] *TSA-DO/SA, Los Alamos National Lab, USA*

This work is part of our ongoing effort to design and implement a traffic simulation application capable of handling realistic problem sizes in multiple real–time. A 16–CPU SGI Power Challenger offers real–time for the whole German Autobahn network. On a workstation cluster we have reached multiple real–time for the Autobahn network of the state Nordrhein–Westfalen. In this paper we present the parallel architecture and techniques used in our implementation. We also give an upper-bound estimate for the scaling behavior of this type of simulation.

1 Introduction

Over the last decade the phenomenon of individual vehicular traffic has become more and more important since transportation is one of the basic conditions for economic wealth. The number of vehicles, however, has increased so quickly that disadvantages start to outweigh the benefits. Traffic not only causes noticeable damage to the environment it also causes billions in losses through delayed transport of goods and people. That is why there is an increasing demand for tools capable of realistic traffic simulation. Conventional applications, however, usually fail to reproduce the phenomena found in real life traffic because either (a) the resolution of the simulation is not fine enough or (b) the considered area is too small.

Only high-end parallel computer architectures in conjunction with high-speed physical methods (e.g. cellular automata) deliver the computational performance necessary to tackle these problems.

This work is part of our ongoing effort of the project "Nordrhein–Westfalen Research Cooperative Traffic Simulation and Impacts on the Environment (NRW–FVU)" [1] to design a flexible, high–performance simulation tool for vehicular traffic. The problem size was given by the current Autobahn network (called "map FRG") of Germany which amounts to approximately 75,000 kilometers of road lane. Estimating an average occupancy of 10% results in 1,000,000 vehicles, each following an individual route plan during the simulation. We also used the Autobahn network of Nordrhein–Westfalen (called "map NRW", a sub–set of the latter) with approximately 11,500 kilometers road lane at an occupancy of 5% with 78,000 vehicles.

The traffic model used in this implementation was developed by Nagel and Schreckenberg [2] as a single lane version and later extended by Rickert, Nagel, Schreckenberg, and Latour [3] to a multi-lane version. Rickert [4] implemented a parallelized traffic simulation using cellular automata (CA) techniques running multiple real–time for the German Autobahn network on an Intel Paragon. This simulation, however, was not capable of executing individual route-plans, but used turning probabilities at intersections instead. Moreover it did not include dynamic load balancing. Nagel [5] used a two CPN [a] topology to run a parallel net simulation using individual route-plans with a single lane CA.

A more detailed description of the microsimulation presented here can be found in [6]. It includes a summary about how the classical CA model is extended to include street network elements (e.g. ramps, intersections) and route-plans which are necessary in many fields of applied traffic research.

Several other research groups are currently involved in large-scale traffic simulation projects. The TRANSIMS [7] [8] group at the Los Alamos National Lab uses a modified version of the Parallel Toolbox [15] to simulate large urban areas, resolved down to individual intersections on workstation clusters. The PARAMICS [9] [10] group in Edinburgh (EPCC) simulates the whole federal road network of Scotland on a Cray T3D (512 DEC Alpha nodes).

In the remaining part of this section we present some basic aspects of traffic simulation and describe the underlying CA model of traffic flow. In section 2 we outline the parallel implementation scheme of our application, which will be followed by some remarks out the parallel environment. We conclude by presenting the benchmarks for two computer hardwares (section 4) and some performance estimates for larger numbers of CPN (section 5).

1.1 Traffic Simulations

Generally, all road traffic simulation models can be classified into *microscopic* or *macroscopic* (fluid-dynamical) models.

The microscopic (high-resolution) simulation uses individual cars, each of them equipped with a route-plan to be followed. Their dynamics are modelled on very different levels of fidelity [b]. One of the most detailed models uses a com-

[a] Computational Node: a general term for one unit of a large computer system solving a part of a distributed parallel problem.

[b] In this context *fidelity* is often used as a synonym for *accuracy*.

plicated delay differential equation for every car [11], in which the acceleration of the car depends on the distance and velocity difference to the car ahead. Although such models perform nicely when compared to measurements, they are computationally very demanding, and have a large set of parameters to be adapted to reality.

At the low end of complexity, we arrive at a model in which both space and velocity are discrete. The dynamics of the cars are reduced to a few simple rules, which are controlled by a small set of parameters. Models of this kind are called cellular automata (CA) for traffic simulation, and can be understood as a minimal microscopic model for simulating traffic. To have only a small set of parameters to calibrate a CA turns out to be a tremendous advantage when coping with more complicated situations such as the lane changing behavior, where it is difficult to calibrate the more complicated models [12].

The macroscopic (low-resolution) models [13] describe the movement of blocks of cars, according to some rules which utilize the continuity equation, together with the empirically measured relation between the car density ρ and flow q. They can be understood as a spatial and time discrete version of a partial differential equation describing a particle flow. Macroscopic models are able to simulate road traffic on very large networks, e.g. it is possible to simulate the German freeway network on a single workstation [14] in real-time. Their main disadvantage, in our view, is their inability to handle a large number of route plans. So, any investigations which rely on route-plans, such as in telematics applications, cannot be done.

Based upon the considerations made above, we think that the microscopic modeling is the most natural way to simulate road traffic.

1.2 Traffic Simulation using CA

Let us briefly summarize, how to simulate traffic [2]. Space, time and velocity are discrete, each cell is either occupied by a car or is empty. The length of a cell is 7.5 $[m]$, which is interpreted as the length of a car plus the gap between cars in a traffic jam. One time-step corresponds to 1 $[s]$, which is of the order of the reaction time of humans. Velocity ranges from $0, \ldots, v_{\max} = 5$, corresponding to a maximum velocity of approximately 120 $[km/h]$.

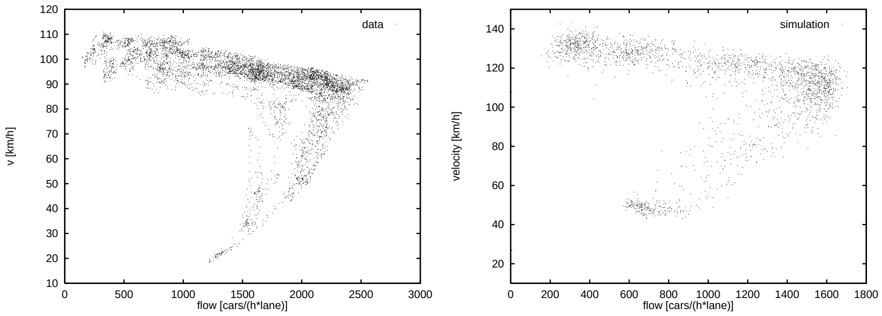

Figure 1: *Left:* Measured fundamental diagram. Data are from a Californian highway. *Right:* Simulated fundamental diagram. Data are a combination of different ρ-values.

Let n denote the current time-step, $(\Delta x)_n$ the front bumper to front bumper distance between the car we are looking at and the car ahead, and let v_n, x_n be the current speed and position, respectively. We obtain the following set of rules, which are updated in parallel for all vehicles:

$$v = \min(v_n + 1, (\Delta x)_n - 1, v_{\max}), \tag{1}$$

$$v_{n+1} = \begin{cases} \max(0, v-1) & \text{with probability} \quad p_{\text{brake}} \\ v & \text{otherwise} \end{cases}, \tag{2}$$

$$x_{n+1} = x_n + v_{n+1}. \tag{3}$$

The first rule summarizes the interaction between two cars and their tendency to drive at maximum speed, while there is no other car ahead. The interaction is constructed to avoid any accident. The second rule accounts for different kinds of inaccuracy in human driving behavior, making the model a stochastic CA. The third rule simply advances the cars v_{n+1} sites.

The original model works with one maximum velocity only. However, it is simple to introduce a distribution of velocities and different car types (e.g. trucks).

Despite its simplicity, the model yields quite realistic behavior. It describes the spontaneous generation of traffic jams, it produces space-time plots of traffic flow, which look very similar to aerial views of real traffic, and it yields realistic fundamental diagrams. A fundamental diagram is the graphical representation of the relation between average local speed $\langle v_l \rangle$ and flow q. An example is shown in the two plots of figure 1, where it is difficult to distinguish between the simulated and the measured fundamental diagrams, showing that the CA is capable of reproducing the observed macroscopic behavior.

Even in more complicated situations the CA-model exhibits remarkably realistic behavior. Examples are the mixing or weaving of two traffic flows or the lane changing behavior of the model. We have found a set of rules, which leads to the correct lane-usage behavior: on German freeways, the left lane has a higher occupancy than the right lane, even for moderate values of the flow. This behavior can be reproduced with the CA-model. More details can be found in [13].

2 Parallel Architecture

The traffic network is associated with a graph of vertices and edges. Nodes represent network elements like network terminators [c], ramps, and intersections. Edges represent open road segments.

We use a straightforward SIMD master slave architecture with a geometrical distribution of the graph. The simulation is designed for computer topologies with distributed memory and message passing, even though one architecture used in our benchmarks is a native shared memory architecture. All control functions necessary for parallel execution as well as the load balancing are handled by the Parallel Toolbox [15] developed by one of the authors.

2.1 Master and Slave Functionality

On a computer hardware with p CPN the simulation is started as a single-CPN application representing the master, which reads the road network data and performs the initial distribution of all graph elements (vertices and edges). Afterwards the master will spawn $p-1$ slave-processes using library calls of the communication library PVM [16]. At that time, all elements only exist in an *inactive* state, which is to say that only the elements themselves have been instantiated, but no secondary data (e.g. CA grids for edges and intersection control for vertices) is associated with them. All slaves receive a copy of their respective sub-networks plus some neighboring vertices of inter–CPN edges (see fig. 2). Note that the master also performs all actions of a slave and thus will receive a sub-network of its own.

As the last step of the initialization phase, the master broadcasts an activation message prompting the slaves to activate the network elements by creating secondary data. The master will keep an inactive representation of the whole network (mainly for graphics and global statistics), which results in slightly

[c] physical boundaries of the road network in contrast to logical boundaries between CPN

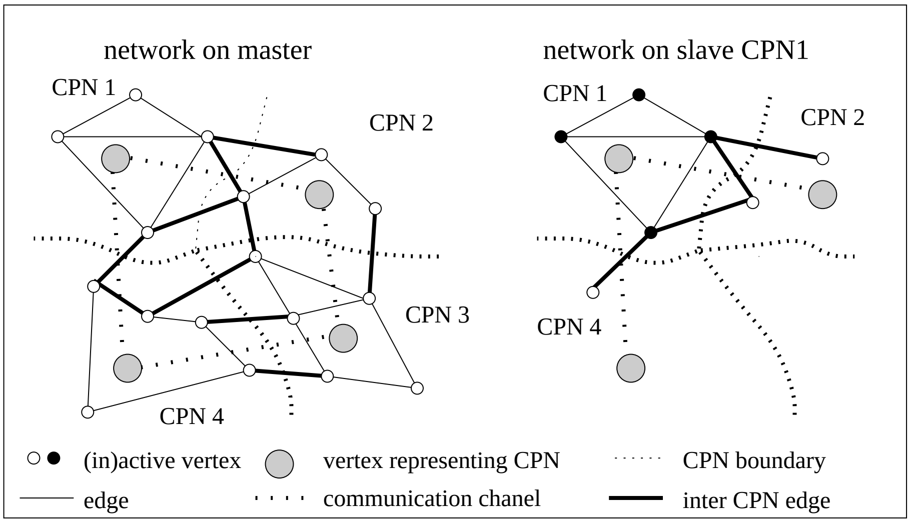

Figure 2: *Global and local sub–networks*

increased memory requirements.

2.2 Initial Distribution

Based on the Euclidean coordinates of the network elements we perform an iterative orthogonal recursive bisection of the vertices onto the available CPN. The weights (by which the vertices are bisected) are computed by assigning each vertex with half of the weights of its incident edges, which are estimated to be proportional to the Euclidean lengths of the edges. Fortunately the simulation speed of the CA model only exhibits a weak dependency on the vehicle density. Over the density range $0.01 \ldots 0.3$ the time required to perform one million updates only varies by a factor of two (see fig. 3 which was obtained from simulation runs of an excerpt of the NRW map).

For the time being, the assignment of sub-networks to CPN does not take the physical communication topology into account, that is, we use a trivial, linear mapping of sub-networks onto CPN. On the topologies used so far, Ethernet (Sparc-cluster) and memory bus (SGI), this does not result in a performance decrease, since all communication channels can be regarded as equivalent.

Also note that due to the fact that CPN can have inhomogeneous performance

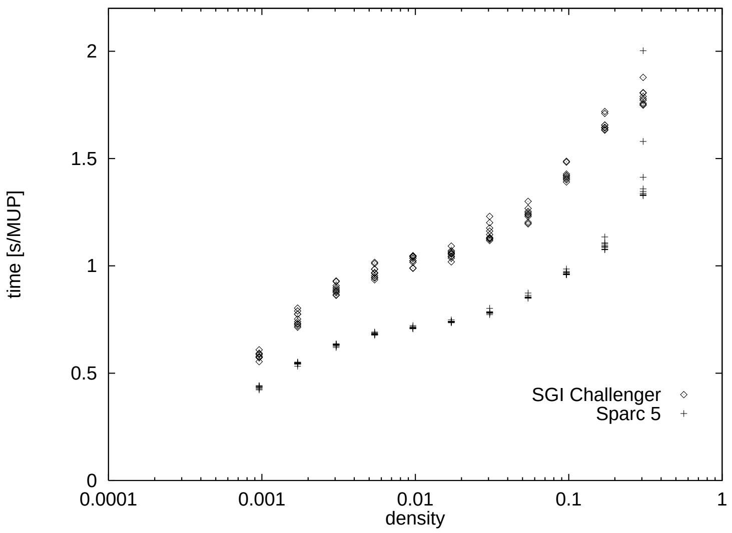

Figure 3: *Update-Time versus Vehicle Density*

and not necessarily a number of CPN equal to a power of two is used, the expression 'bisection' is not to be taken literally. In particular we try to split the number of available CPN, then compute the sum performance of the two subsets and assign two sub–networks which have the same ratio of estimated loads as the ratio of performances of the CPN subsets.

We do not put much effort into this part, since the dynamic load balancing compensates for an insufficient initial distribution.

2.3 Boundaries

Inter–CPN edges exist in two copies on either CPN they refer to (see fig. 4). They are split in the middle with one half being active on the first CPN, the other being active on the second CPN. Boundaries have a length (measured in sites) as large as the maximum interaction range of the CA rule set, which is currently v_{max} (or $v_{max} + 1$ in some cases). They are requested from the edges on both CPN, encoded and transferred to remote the CPN, where they are appended to the active range 'faking' an infinite system to the CA. Note that the time spent on retrieval, transfer, and storing of boundaries is considerable, compared to a more heterogeneous application which is able to provide

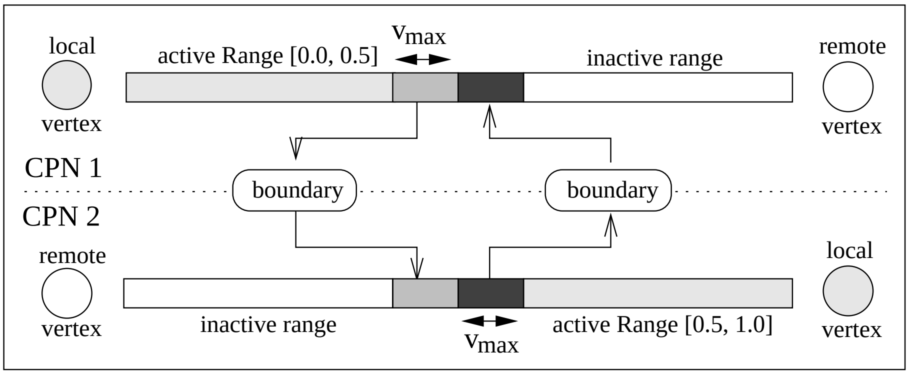

Figure 4: *Inter–CPN edge with boundaries*

boundaries as e.g. linear arrays. We will take this extra effort into account as *application-level* communication (see section 5).

2.4 Simulation Control

The simulation is basically time–step driven since the built-in CA needs all information of a previous time–step to compute the current one. This results in boundary communication along inter–CPN edges for each time–step [d]. We use the boundary exchange to control the timing on each slave. A so called *simulation sequence* is executed as follows:

1. The master assumes that all slaves are synchronized at on a (not yet executed) time-step t_n.
2. The master initiates a sequence of length s by broadcasting a message to all slaves. It falls back into slave mode.
3. Each slave sends out boundary information for time–step t_n to all its neighbors [e] and then waits for incoming boundaries.
4. After a CPN has received all boundaries from its neighbors, it executes time–step t_n. If the end of the simulation sequence has been reached, it

[d] In fact we exchange boundaries twice for each time–step since one CA update is split into two sub steps

[e] Two CPN are regarded as neighbors (connected by a communication channel), iff there is at least one inter–CPN edge between them.

sends a message to the master. If not, it continues to send out boundaries for time–step t_{n+1}.

5. As soon as the master has received messages from all slaves acknowledging the end of the simulation sequence, it resumes master mode to initiate another sequence or to terminate the simulation. Note that at this time all slaves are synchronized on time step t_{n+s}.

2.5 Dynamic Load balancing

Since there is boundary communication before each time–step, it must be the goal of the dynamic load balancing to equilibrate the execution time of all CPN. We have implemented a straight-forward *local decision, local migration* ($LDLM_S$, see [17]) strategy which we will describe next.

Estimating Load

During execution each CPN i uses wall-clock timers to monitor the time t_i needed to execute one time–step of the local sub–network. Moreover it can request an estimate l_i (in arbitrary units) of the load of its local sub–network. These weights are computed the same way as described for the initial distribution by simply summing over the actual road lengths.

At the same time it receives equivalent information from its neighbors in intervals of t_{lb} time-steps, which is used to compute a performance value $P_i = l_i/t_i$. CPN i stores those values for a certain period of time $t_{monitor}$, from which it retrieves the minimum P_i^{min} as its *safe performance value.* A *safe work load* for this CPN is thus $L_i = l_i/P_i^{min}$, which has the unit of a time period and is used to determine the load differences with respect to its neighbors.

Note that the advantage of using the minimum P_i^{min} instead of the current value P_i lies in preventing moving load to a CPN that is *only temporarily* running idle. CPN that are used interactively show such behavior.

Transferring Load

Whenever there is significantly[f] more load on a CPN i than on one of its neighbors j, CPN i will try to initiate a transfer of vertices by requesting a local synchronization with CPN j. The actual amount of load l_t to be cast off

[f] exceeding a certain predefined threshold

is computed by comparing the safe work loads L_i, L_j and trying to reach equal execution times t_i, t_j on both CPN as follows:

$$L_i - \frac{l_t}{P_i^{min}} = t_i \stackrel{!}{=} t_j = L_j + \frac{l_t}{P_j^{min}}$$

Isolating l_t and introducing a dampening factor c yields:

$$l_t = c\,\frac{l_i P_j^{min} - l_j P_i^{min}}{P_i^{min} + P_j^{min}}$$

The dampening factor $c = 1/n_n < 1$ is used to prevent a CPN from receiving l_t load units from more than one of its n_n neighbors. Vertices to be offloaded are selected along the common inter–CPN edges until l_t is reached. To maintain well shaped sub–networks those vertices furthest away from the center of gravity of the sub–network are favored.

Whenever a vertex is offloaded the local copy either a) falls back into the inactive state if it still serves as a dummy connected to an inter–CPN edge, or b) is completely discarded (see fig. 2). The latter is also true for edges that have been transferred completely.

Despite the fact that in principle each load transfer is restricted to the offloading CPN A and the receiving CPN B. There may be other *third party* CPN which have to be informed about the transfer. In case a vertex is transferred that is also used by another third party CPN C there is a chance of incoming boundaries from C which are addressed to A. The receiving inter–CPN edge, however, now resides on B, so that the boundary has to be forwarded. Therefore, CPN A handles a list of edges which have been transferred lately.

Timing

In order to spread the communication due to load transfers over the interval t_{lb} the connections between CPN are colored with 4 colors representing time slots within the l_{lb} interval. At the beginning of the simulation all connections are colored by the master. New connections that are created due to transfers receive a flag denoting an invalid color and preventing any transfer. The CPN involved try as soon as possible to negotiate a new color which is chosen from those that are currently not used by either of them. If no such color can be found an arbitrary color is used.

3 Environment

We use C++ as an object oriented programming language for our implementation. The parallelization is handled by a toolbox which supplies base classes for vertices and edges, as well as slave and master control objects. The traffic microsimulation itself is defined by supplying descendant objects overwriting virtual class methods. The toolbox uses PVM as a message passing library.

3.1 Parallel Environment

The simulation should operate on all machines that support both PVM and a current version of GNU C++. We have successfully ported the simulation to the architectures SGI5, SGI64, SUN4, SUN4SOL2, and LINUX. The RS6K architecture only works without optimization. The shared memory versions SGIMP64, SUNMP, and ALPHAMP as well as the dedicated version for the Intel Paragon PGON can be built, but need further debugging.

3.2 Fault Tolerance

The simulation does not provide any protection against CPN failures, since there are no redundant copies of data available. However, the current design offers[g] the option to *dynamically* add or remove CPN on the fly. The toolbox will automatically switch between raw encoding for CPN topologies with homogeneous binary data representation and XDM encoding for heterogeneous topologies.

Adding a CPN

The PVM shell allows to interactively add a new CPN to the parallel machine. The master is informed by the PVM `pvm_notify` mechanism about this new resource. It will try to insert the CPN k after termination of the current simulation sequence as follows:

(1) The master will scan its global load data to find the CPN i that is most heavily loaded. (2) CPN i checks the load of its neighbors and selects the CPN j which is most heavily loaded. (3) CPN i transfers one seed vertex on a common boundary with j to k. (4) The master continues as usual.

[g] Within the restrictions imposed by PVM, which is to say that the master process can never be terminated without terminating the whole simulation.

Choosing the CPN like this ensures that the new CPN has at least two neighbors inclined to offload part of their sub-networks during the upcoming load balancing intervals.

Removing a CPN

Unfortunately, PVM shell does *not* permit the removal of a CPN since that action would terminate the PVM daemon corrupting the application immediately. Any slave CPN, however, can be removed from the parallel machine as follows:

(1) After reception of the signal `SIGHUP` it initiates a forced offload of all vertices except one. (2) It sends a message to the master expressing the wish to be removed. (3) The master waits until the end of the current simulation sequence and prompts the CPN to offload the remaining vertex. (4) The master continues as usual. (5) Now, the PVM daemon can be removed using the PVM shell.

4 Benchmark Results

The simulation was tested on an SGI Power Challenger system with 16 processors (Irix 6 with PVM architecture SGI5) and on a workstation cluster consisting of 12 Sun workstations (Sparc 5 and Sparc 20 under Solaris using PVM architecture SUN4SOL2). We performed several runs each lasting between 300 and 600 simulation time-steps. Note that in all cases the best performance was reached after about 100 time-steps (equivalent to about about 15 load-balancing transfers).

In figure 5 the simulation reaches an equilibrium at a real time ratio of about 1 for map FRG on the SGI with 16 CPN and on the cluster with 12 CPN. For map NRW a real time ratio of 3.75 can be achieved on a workstation cluster with 8 CPN.

The efficiency of all runs was at least 0.85 after the initial load balancing period (see fig. 6).

5 Performance Analysis

We have made a first attempt to deduce an upper-bound for the efficiency $e(p)$ of a large scale traffic simulation running on p CPN. It is based upon a set of parameters which can be retrieved from simple measurements. We only cite

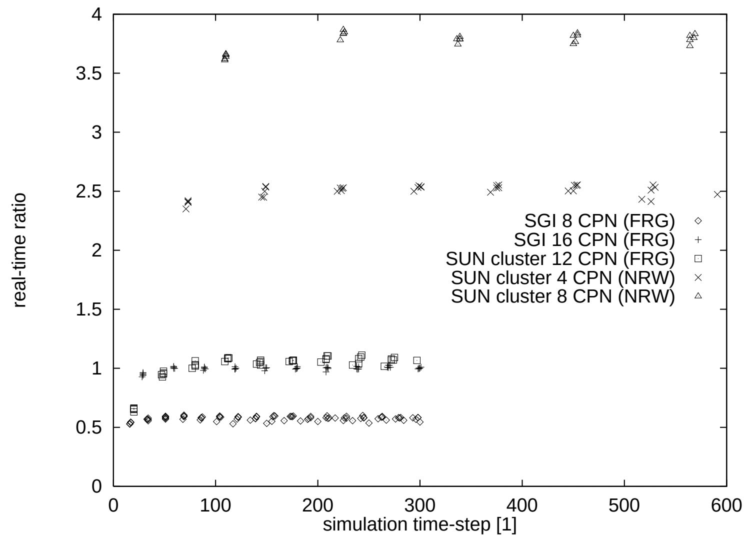

Figure 5: *Real Time Ratio*

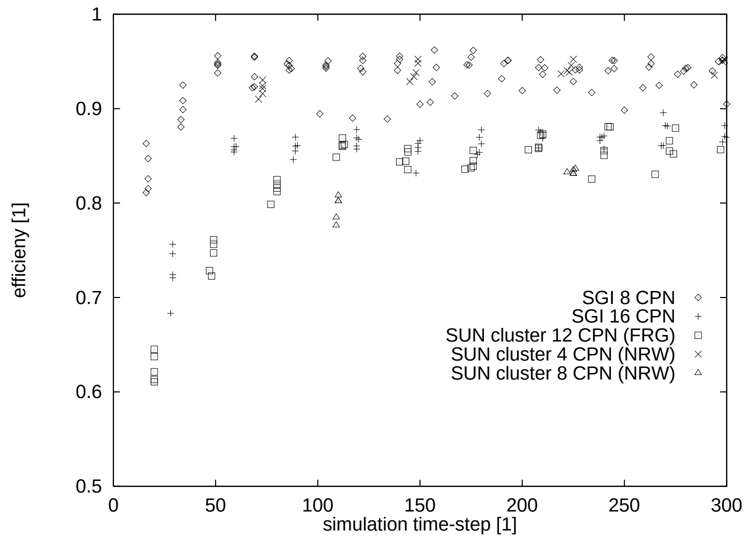

Figure 6: *Efficiency*

results here. A more detailed version (including an additional estimate for a two-dimensional communication topology) can be found in [18].

Assuming a time of $T(1)$ required for one time-step on a single-node machine, the necessary input parameters turn out to be:

- the size of the street network (number of edges, number of nodes) resulting in estimates for the number of neighbors $N_n(p)$ (with an average number of $n_n(p)$ per CPN) and the number of boundaries $B(p)$ (with an average number of $b(p)$ per CPN),
- the number of sub-time-steps n_{sub} per time-step causing a relative performance loss for administration overhead $f_{adm}(n_{sub})$ and additional communication volume,
- the average boundary length b_{size} and the boundary message header size b_{header}, the application-level boundary transmission time t_{c1} and transmission latency t_{cl} both given as fractions of $T(1)$,
- the low-level communication bandwidth C_{net} (measured in byte per $T(1)$) of the computer network, and
- the relative load gradient $f_{grad}(p)$ generated by the granularity of the street network.

Using these parameters we define four major contributions to the time spent on one time-step which are plotted in figures 7 and 8:

- The *raw simulation* fraction mainly represents the traffic simulation itself, although it includes the administrative overhead for multiple sub-time-steps. It is equivalent with the efficiency $e(p)$ of the simulation if $n_{sub} = const.$
- The *load-gradient* fraction represents the loss of execution time due to the load gradient which builds up throughout the CPN network. We assumed a relative exponential gradient of 0.01 per layer.
- The *application-level (a-l) communication* fraction represents the time spent on retrieving, coding, transferring, decoding, and storing boundary data.
- Finally, the *low-level (l-l) communication* represents the additional time spent on low-level communication due to the saturation of the underlying communication network.

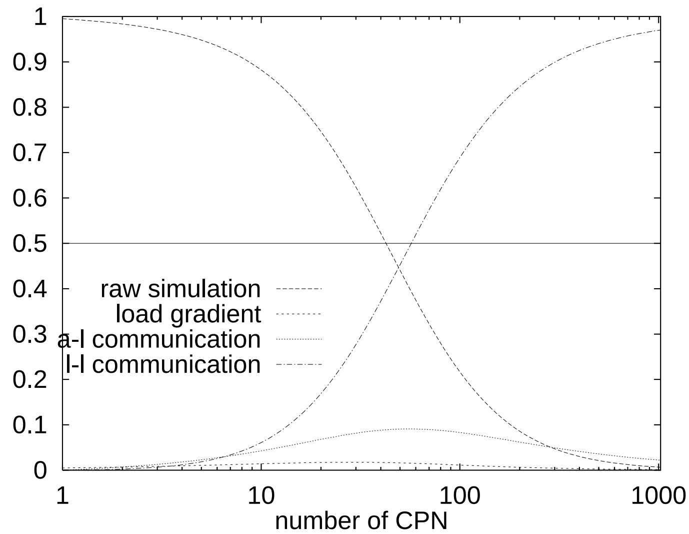

Figure 7: *Sparc-5 Cluster with Ethernet (Bus-Topology)*

The overall efficiency turns out to be

$$e(p) = \Bigg(\underbrace{n_n(p)t_{cl}}_{\text{latency}} + \underbrace{b(p)t_{c1} + \frac{N_n(p)b_{header} + B(p)b_{size}}{C_{net}}}_{\text{bandwidth}} + \\ + \frac{1}{p}\Big(1 + \underbrace{f_{adm(n_{sub})}}_{\text{overhead}} + \underbrace{f_{grad}(p)}_{\text{gradient}}\Big)\Bigg)^{-1}$$

Figure 7 depicts the estimate for a cluster of Sparc 5 workstation connected by Ethernet. Efficiency quickly drops below 0.5 for 30 CPN due to the network saturation by *low-level communication*. We used the parameters: overall density $\varrho = 0.05$, $T(1) = 13.6[s]$, $c_{net} = C_{net}/T(1) = 1.0[Mbyte/s]$, $T_{c1} = T(1)t_{c1} = 0.62[ms]$, $T_{cl} = T(1)t_{cl} = 3.64[ms]$, $b_{header} = 64[byte]$, and $b_{size} = 118[byte]$.

Figure 8 shows the same qualitative behavior for the SGI challenger although an efficiency of at least 0.5 can still be obtained for up to 100 CPN. We used the parameters $\varrho = 0.1$, $T(1) = 10.2[ms]$, $C_{net} = 5.0[Mbyte/s]$, $T_{c1} = 0.31[ms]$, $T_{cl} = 2.16[ms]$, $b_{header} = 64[byte]$, and $b_{size} = 160[byte]$.

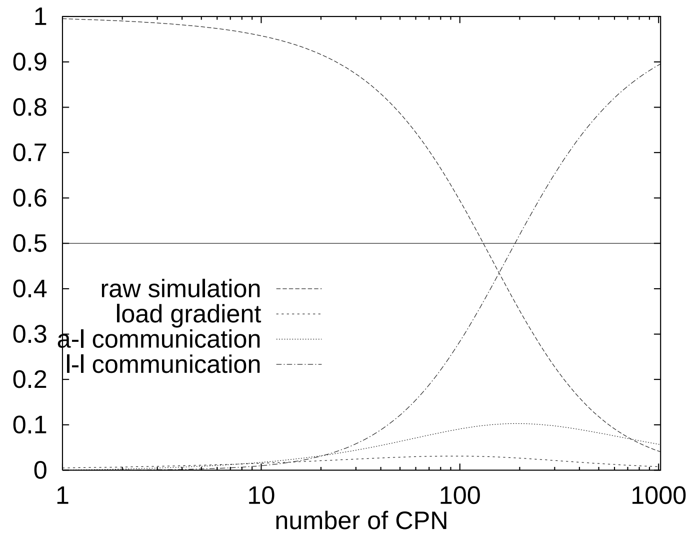

Figure 8: *SGI-Challenger with Shared Memory(Bus-Topology)*

6 Discussion and Outlook

We have presented a real–time simulation application portable to various parallel computer architectures. On the platform SGI we can reach real–time for a complex street network with 1,000,000 individually routed objects, which represents a considerable increase of performance compared to conventional systems.

On the one hand we exploit the simple geometric structure of the given road network for geometric distribution. On the other hand we have to deal with the strong time dependence of the sub–graphs making the implementation of a dynamic load balancing desirable. Note that in the future we will need dynamic load balancing not only because of performance fluctuations imposed by other users in a workstation cluster, but also by the simulation itself: It may be necessary to run different parts of the traffic network at different levels of fidelity depending on what type of output statistics one is interested in. In that case, the low fidelity CA model would be complemented by a — computationally more demanding — high fidelity driver model. During run-time the simulation would switch between these models according to some yet to be defined criteria.

So far we have not investigated the parameter space defining the load balancing behavior, like the length of the monitoring period $t_{monitor}$, the threshold for offloading, or the dampening factor c. This will be covered in future publications.

We will also try to integrate message passing and shared memory techniques into a new toolbox, which will run on hybrid computer systems. This is to remain independent of the message passing technology which has lost part of its significance over the last couple of years. Nevertheless, porting the simulation to dedicated machines with a high-end communication architecture will definitely be one of the goals of our group in the near future.

7 Acknowledgements

We thank A. Bachem, R. Schrader and C. Barrett for supporting MR's work as part of the traffic simulation efforts in Cologne ("NRW-FVU") and Los Alamos (TRANSIMS). We also thank them, K. Nagel, S. Krauß for help and discussions. Computing time on the SGI-1 Challenger of the Regionales Rechenzentrum Köln and on the workstation cluster at TSA-DO/SA is gratefully acknowledged. We further thank all persons in charge of maintaining the above mentioned machines.

The work of MR was supported in part by the "Graduiertenkolleg Scientific Computing Köln/St. Augustin".

References

1. URL. http://www.zpr.uni-koeln.de/Forschungsverbund-Verkehr-NRW/.
2. K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *J. Physique I*, 2:2221, 1992.
3. M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour. Two lane traffic simulations using cellular automata. accepted by Physica A, 1995.
4. M. Rickert. Simulation zweispurigen Verkehrsflusses auf der Basis zellularer Automaten. Master's thesis, Universität zu Köln, 1994.
5. K. Nagel. *High–speed Microsimulations of Traffic Flow.* PhD thesis, Universität zu Köln, 1994.
6. M. Rickert and P. Wagner. Parallel real-time implementation of large-scale, route-plan-driven traffic simulation. accepted by Int.J.Mod.Phys. C, 1996.
7. K. Nagel, C. Barrett, and M. Rickert. Parallel traffic micro-simulation

by cellular automata and application for large scale transportation modeling. Technical report, TSA-DO/SA, Los Alamos National Lab, New Mexico, USA, and Santa Fe Institute, Santa Fe, New Mexico, USA, and Center for Parallel Computing, University of Cologne, Germany, 1996. submitted to Transportation Research C.

8. TRANSIMS URL. http://studguppy.tsasa.lanl.gov/.
9. D. McArthur. The *PARAMICS* Model: Present and Future Directions. Technical report, SIAS Ltd., Edinburgh, 1994.
10. Paramics URL. http://www.epcc.ed.ac.uk/epcc-projects/paramics/.
11. R. Wiedemann. Simulation des Straßenverkehrsflusses. Technical Report Heft 8, Institut für Verkehrswesen der Universität Karlsruhe, 1974.
12. M. McDonald and M. A. Brackstone. Simulation of lane usage characteristics on 3 lane motorways. In *Proceedings of the 27th International Symposium on Automotive Technology and Automation (ISATA)*, 1994.
13. P. Wagner. Traffic simulation using cellular automata: Comparison with reality. In *Proceedings of the conference "Traffic and Granular Flow"*, 1995. to be published.
14. J.T. Pfenning. *Beiträge zum Einsatz von "Workstation Clustern" als Parallel-Rechner.* PhD thesis, University of Cologne, 1994.
15. M. Rickert. Parallel Toolbox 1.0. Technical report, Center for Parallel Computing, Cologne, Germany, and TSA-DO/SA, Los Alamos National Lab, USA, 1995.
16. PVM URL. http://www.epm.ornl.gov/pvm/pvm_home.html.
17. R. Lüling, B. Monien, and F.Ramme. Load balancing in large networks: A comparative study. In *3rd IEEE Symposium On Parallel And Distributed Processing*, pages 686–689, 1991.
18. M. Rickert. Estimating parallel efficiency of large-scale traffic simulations. in preparation, 1996.