17th International Conference on the Application of Computer
Science and Mathematics in Architecture and Civil Engineering
K. Gürlebeck and C. Könke (eds.)
Weimar, Germany, 12–14 July 2006

PARALLELIZATION OF A MICROSCOPIC TRAFFIC SIMULATION SYSTEM USING MPI-JAVA

K. Erlemann, and D. Hartmann

Institute of Computational Engineering
Ruhr-University Bochum
Bochum, Germany
E-mail: kai.erlemann@ruhr-uni-bochum.de

Keywords: Microscopic Traffic Simulation, Behavioral Models, Reasoning Voting, Parallelization, MPIJava, PC-Cluster.

Abstract. *Traffic simulation is a valuable tool for the design and evaluation of road networks. Over the years, the level of detail to which urban and freeway traffic can be simulated has increased steadily, shifting from a merely qualitative macroscopic perspective to a very detailed microscopic view, where the behavior of individual vehicles is emulated realistically. With the improvement of behavioral models, however, the computational complexity has also steadily increased, as more and more aspects of real-life traffic have to be considered by the simulation environment. Despite the constant increase in computing power of modern personal computers, microscopic simulation stays computationally expensive, limiting the maximum network size than can be simulated on a single-processor computer in reasonable time.*

Parallelization can distribute the computing load from a single computer system to a cluster of several computing nodes. To this end, the exisiting simulation framework had to be adapted to allow for a distributed approach. As the simulation is ultimately targeted to be executed in real-time, incorporating real traffic data, only a spatial partition of the simulation was considered, meaning the road network has to be partitioned into subnets of comparable complexity, to ensure a homogenous load balancing. The partition process must also ensure, that the division between subnets does only occur in regions, where no strong interaction between the separated road segments occurs (i.e. not in the direct vicinity of junctions).

In this paper, we describe a new microscopic reasoning voting strategy, and discuss in how far the increasing computational costs of these more complex behaviors lend themselves to a parallelized approach. We show the parallel architecture employed, the communication between computing units using MPIJava, and the benefits and pitfalls of adapting a single computer application to be used on a multi-node computing cluster.

1 INTRODUCTION

Microscopic traffic simulation has evolved from a purely academic research area to a valuable tool for the practical road designer. Over the years, the level of detail that can be obtained by simulation models has steadily increased. The computer-simulated vehicles are behaving more and more like their real-world counterparts, and many real-life phenomena can now be emulated. This enables the traffic engineer to cost-efficiently evaluate different road designs and compare the results for each variant. Furthermore, it is now possible to consider non-standard road layouts that cannot reliably be evaluated using the traditional design-table approach. For example, consider two succeeding freeway junctions. The smaller the distance between both junctions, the stronger the interactions between the on-ramping and off-ramping traffic becomes. Such cases are not covered by conventional pen-and-paper methods, but can be analysed by traffic simulation with comparable ease.

BABSIM, the "'federal highway simulator"', is a Java-based traffic simulation framework that has been developed at the Institute of Computational Engineering, in coorperation with the Institute for Traffic Engineering and the German Federal Highway Research Institute, since 2000. Specialized on freeway traffic, BABSIM is capable to emulate complex freeway interchanges or junctions, where vehicles often have to weave through multiple lanes to reach their designated on-ramps or off-ramps. To ensure an unobstructed traffic flow, the vehicles have to be situation-aware, and have to use anticipatory, long-time driving strategies instead of the short-sighted momentary decisions common in traditional traffic simulation. In the following chapter, we will discuss the multi-goal behaviour employed in the current version of BABSIM.

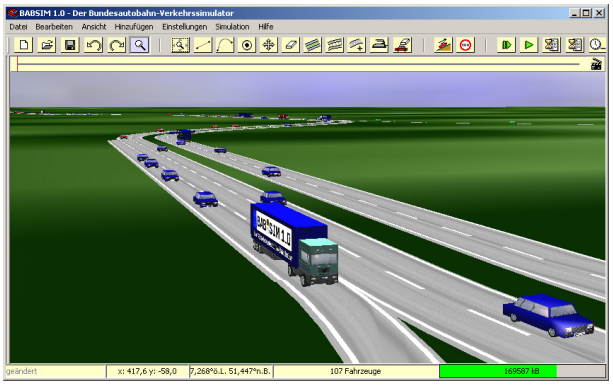

Figure 1: Screenshot from BABSIM

Improving the simulation detail results in more a realistic simulation, but this comes at a well-known prize: the computational effort increases significantly compared to traditional models. Numerous interactions between a vehicle, its neighbours, and even more distant vehicles have to be considered. Therefore, it is currently not feasible to perform a real-time simulation of a large-scale road network on a single computer. A rough estimate would be a maximum network size of 50 kilometers of freeway on a current 2,5 MHz desktop computer. To increase the road network, a parallelized approach has to be taken to distribute the computing load to multiple compute nodes. The problems of adjusting an existing single-processor application to be used on a multi-processor environment are being discussed.

2 MICROSCOPIC DRIVING BEHAVIORS

The basic elements of each microscopic traffic simulation are driver-vehicle units (DVUs), that move through a road network using driving behaviors. As the vehicles are constricted to move along their respective road elements (comparable to trains moving on rails) the driving behavior only has to determine the movement of the car in axial (acceleration or deceleration) and lateral direction (lane changes). BABSIM uses discrete time-steps, that are adjustable between usually 0.1 and 1.0 seconds. Alternate approaches, such as event-driven simulation methods, are not deemed feasible for complex simulations, as the high level of interaction between the vehicles would necessitate a high number of events being generated, thus counteracting all possible perfomance gains.

Using the constant time-step length Δt and the acceleration a determined by the driving behavior, the velocity v and position s of a vehicle for a given time $t_{n+\Delta t}$ can be calculated using the simple Euler integration:

$$v_{t+\Delta t} = v_t + a_{t+\Delta t} \cdot \Delta t \tag{1}$$

$$s_{t+\Delta t} = s_t + v_{t+\Delta t} \cdot \Delta t \tag{2}$$

Although Euler integration is known to be less accurate than higher-order integration methods like Verlet integration or Runge-Kutta methods, the error introduced is neglectable compared to the inaccuracies on the simulation abstraction level. The most challenging task is now to determine realistic values for $a_{t+\Delta t}$. Several well-known strategies have been developed over the years, e.g. the car following theory by Gazis, Herman and Rothery (1961, [4]), the psychophysical model by Wiedemann (1974, [7]) or the optimal velocity model by Bando et al. (1995, [1]). For a more detailed discussion of the models, refer to [6] or [3]. In the current implementation, BABSIM uses the Wiedemann model for car-following considerations (i.e. for axial movement).

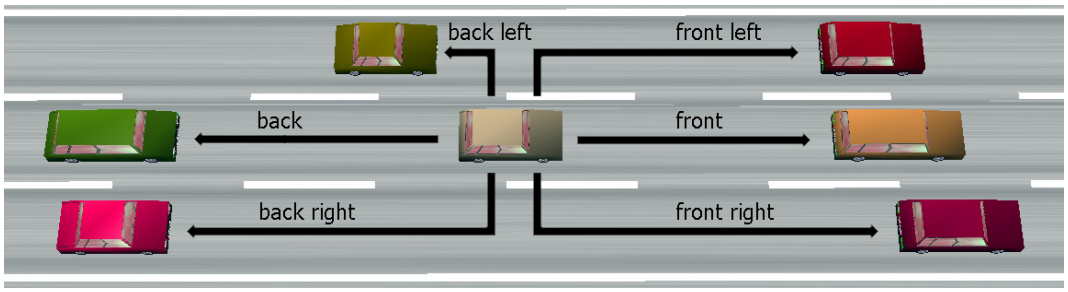

Figure 2: direct interaction partners of a vehicle

To perform lane changes on multi-lane roads, the vehicle must take every neighbor as a possible obstacle into consideration. The behavior is situation dependant: in between freeway junctions the vehicle will try to reach its desired velocity by chosing the lane with the highest benefit for its velocity. As soon as a vehicle enters the influence bounds of a multi-lane junction, the behavior has to check the current lane and, if necessary, initiate a lane change to the desired target lane. Furthermore, vehicles have to support each other to enable changing to densely populated lanes. It is obvious that with each additional situation to be emulated the behavioral model has to be expanded accordingly, thereby increasing the complexity of the behavioral algorithm. With increasing complexity, it becomes increasingly difficult to calibrate

and administrate the driving behavior. A new approach to define complex, expandable, modular driving behavior had to be found.

3 AN INTENTION-BASED BEHAVIOR

Our solution was to split the rather monolithic behavioral algorithm into smaller, modular chunks that can operate independently from each other. These chunks, we call them intentions, are each specialized to deal with one distinct problem domain, such as following the route, avoiding dangerous lane changes, or overtaking slower cars. Each intention is evaluated separatedly, and return its results over a well-defined interface to the managing driving behavior. The task of this behavior is to collect the desired values of each intention and decide which action to take. This is done in a "'democratic"' voting process: each intention can vote for or against an action, or abstain from voting (for further discussion of distributed reasoning, see [5]). Each vote is given numerically from the interval $[-1, 1]$, where positive values endorse an action and negative values veto it. The more the value deviates from 0, the more urgent the action becomes.

To find a "compromise" between these votings, the behavior uses an adjusted geometric mean. Let $b_{n,m}$ be the vote of the nth intention for action m, and a_m the resulting value for action m:

$$a_m = (\prod_i (b_{i,m} + 1)) - 1 \tag{3}$$

This equation ensures that an action is automatically canceled by a single full veto of any one intention (e.g. a warning to change to a lane already in use by another vehicle). Positive votings, on the other hand, merely raise the resulting value, while "no-oppinion" votings do not influence the result at all. After each possible action (i.e. accelerate, change left, change right or decrease security gap) has thusly been evaluated, the driving behavior acts accordingly and applies all possible changes. This process is repeated for each vehicle and each time-step.

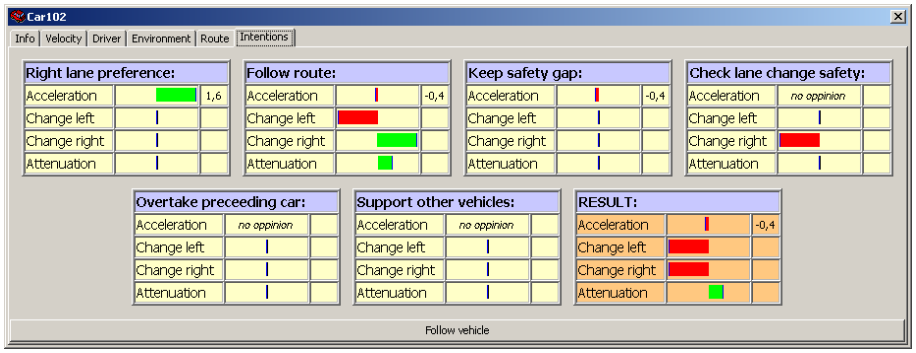

Figure 3: The current intentions of a vehicle during the simulation

During the simulation, the current intentions for each vehicle can be visualized as shown in figure 3. This significantly simplifies the calibration and debugging process, as it is now very easy to identify the intention that triggered a certain behavior. Another advantage is that the behavior can now be expanded by simply adding new intentions, ideally without touching the existing intentions. But, as has been mentioned before, splitting a single behavior into several

separate intentions has also one grave disadvantage: the overall computation costs increase, due to repeated calculation of values or general management effort. To counter this effect, a parallelization of the simulation can be considered.

4 DISTRIBUTED TRAFFIC SIMULATION

Parallelization of traffic simulation means distributing the overall computational effort from one computer to a group of several computing slaves (called compute nodes). The intended gain is either to minimize computation time, or to simulate large-scale road networks that would be too large to be computed on a single-processor system. There are several ways in which the parallelization may occur, e.g.:

- **calibration:** running several instances of the same simulation on multiple compute nodes with varying parameter sets to find the optimal parameter set
- **scenario comparison:** using each compute node to calculate a different scenario (e.g. different road designs) with the same parameters
- **large-scale simulation:** partitioning one large road network into several sub-nets, that are simulated on synchronized compute nodes

While the first two types of application are rather straightforward to implement (simply start the simulation program independently on each node and collate the results), the third method is the most challenging, as a way to synchronize the nodes has to be found. Synchronization means, that each seperate computation, while encompassing a unique spacial domain, occurs in a common time domain with the other nodes. After each node has completed calculating a fixed number of time-step (one or more), it informs all adjacent nodes (and the supervising headnode) that it finished its work, and pauses for synchronization. Once all nodes reported in in such fashion, they are notified by the headnode and can resume their work.

The connection points between the subnets in BABSIM are called *virtual sinks* and *virtual sources*. Each sink in a subnet is connected to a source in another subnet. Once a car reaches a virtual sink, it is eliminated on its current compute node and transferred to the compute node associated with the sink-source connection. There the vehicle ”rematerializes” at the virtual source and proceeds on the new subnet without taking notice of the transfer process. Figure 4 illustrates this process. As the number of vehicles transferred in this way is rather small (vehicles usually have a time-gap of at least 2 seconds, corresponding to 20 simulation time-steps), the communication overhead between the cluster nodes is small and manageable.

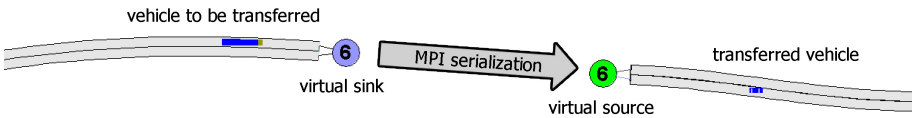

Figure 4: Transferring vehicles from one subnet (left) to another one (right)

We know that the simulation cannot proceed before the last (thus slowest) compute node has finished its calculation. This node is the bottleneck of the parallel simulation, and therefore large time span differences between the fastest and slowest compute node have to be avoided.

As we are assuming a homogenous cluster environment, where each compute node has equal or comparable hardware characteristics, the relative computation time depends only on subnet size and the number of vehicles to be simulated per subnet. An intelligent partition algorithm should therefore try to divide a given road network into subnets of comparable total length and traffic density, while at the same time creating subnet boundaries with as few connection points as possible to minimize communication overhead. Furthermore, the partition must be located in an area where no strong interactions between vehicles are to be expected (e.g. preferrably not in the vicinity of freeway junctions). Unfortunately, developing such an algorithm proved to be more time-consuming than expected, and therefore a more simple grid-based partition strategy, with the option to adjust the boundaries manually, was chosen for the current version of BABSIM.

5 IMPLEMENTATION DETAILS

Since 2005 the Institute of Computational Engineering employs a 52 node Linux-cluster running a LAM/MPI as communication protocol. The compute nodes are not directly visible for external access, but have to be addressed using the supervising headnode. The headnode initiates, manages and terminates all compute processes. As the whole BABSIM framework was developed in Java to ensure platform-independence, all communication on the cluster-level has to take place using the MPIJava package. MPIJava, developed as part of the HPJava project at Indiana University ([2]), is a set of JNI wrapper classes to access native MPI 1.1 packages. Once the MPI environment on the cluster has been initiated with the desired number of compute nodes, each node can directly communicate with other nodes using a very simple communication protocol. Therefore, it is comparatively easy for nodes to transfer vehicles from a virtual sink directly to the virtual source on the target node.

Not every user of BABSIM can afford to operate its own multi-cluster, and therefore a tool has been incorporated into BABSIM to directly send a road network to the headnode of the cluster. As the remove client is most probably not part of the MPI environment, a different means of communication has to be established between client and headnode, such as a CORBA or RMI connection. Too keep the communication as simple as possible, a straightforward HTTP socket connection has been chosen that can be realized using the basic Java packages.

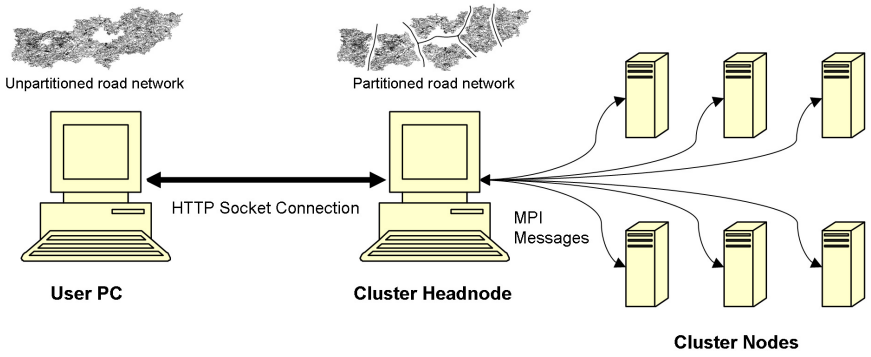

Figure 5: The client-server architecture employed

Once the user has constructed or loaded a road network, he can send it to the cluster using the socket connection and a very simple graphical user interface. The headnode then delegates

the network to one of the compute nodes to partition the network, and sends the resulting network back to the user for cross-checking. After the user has approved the chosen layout, he initiates the computing process. The headnode now sends each subnet to one available compute node. These begin their work as soon as the subnet transfer is finished, and synchronize with adjacent nodes, as described previously. During the simulation, each node collects the relevant simulation evaluations as specified by the user (such as travel times or traffic densities). After a preset total simulation time has expired, all compute nodes cease their work and return their results to the headnode, who collects them and forwards them to the client computer via socket connection. The simulation run is now finished, and the user can review the data generated by the simulation.

Figure 6 shows a road network that has been calculated on the cluster. As the simulation was proceeding, the current results were converted exemplary into a Google Earth input file (.kml format) and updated in real-time. The resulting images show the potential for

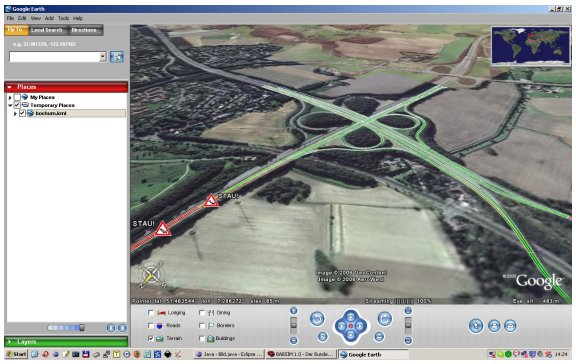

Figure 6: A real-time BABSIM simulation imported into Google Earth

6 CONCLUSIONS

Parallelization of existing applications is a challenging task. To justify the effort necessary to perform this task, the gains obtainable by a distributed approach have to be significant. In the first part of this paper, the basic requirements of a traffic simulation have been summarized. It has been shown that traditional behaviors are difficult to handle and to debug. An alternative strategy to separate an algorithm into smaller object-oriented parts and using a voting process to decide upon the future course of action has been presented. Parallelization is a means to prevent the additional computing effort from slowing the simulation. A strategy to partition the roads into several subnets with comparable numbers of vehicles has been illustrated. It was then shown how this strategy can be implemented on a PC-cluser using MPIJava as a communications tool.

For coming versions of BABSIM a tool has to be deviced that automates the partition algorithm. Furthermore, a more detailed communications protocol has to be specified, to enable a more selective access to simulation results on each compute node. And last but not least, we intend to create a user management system to define and manage user accounts, where each user has a limitable time allotment on the server, that is assigned according to licensing agreements. That way, we can ensure that accessing a remote compute server can become a powerful tool to perform complex simulations for future traffic design projects.

REFERENCES

[1] M. Bando, H. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama. Dynamical model of traffic congestion and numerical simulation. *Physical Review*, E51, 1995.

[2] B. Carpenter, V. Getov, G. Judd, T. Skjellum, and G. Fox. Mpj: Mpi-like message passing for java. *Concurrency: Practice and Experience*, Volume 12, Number 11, September 2000.

[3] Kai Erlemann. BABSIM - An object-oriented software framework for microscopic simulation of freeway traffic. In K. Karl Beucke, Firmenich, B. Donath, R. Fruchter, and K. Roddis, editors, *Xth International Conference on Computing in Civil and Building Engineering*, Weimar, 2004.

[4] D.C. Gazis, R. Herman, and R.W. Rothery. Non-linear follow-the-leader models of traffic flow. *Operations Research 9*, No. 4, 1961.

[5] J. Hancock. A distributed-reasoning voting architecture. In *Game Programming Gems 4*. Charles River Media, 2004.

[6] D. Helbing. *Vekehrsdynamik - Neue physikalische Modellierungskonzepte*. Springer Verlag, 1997.

[7] R. Wiedemann. *Simulation des Strassenverkehrsflusses*. PhD thesis, Universitt Karlsruhe, 1974.