Proceedings of the 2009 Winter Simulation Conference
M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, eds.

ACCELERATING TRAFFIC MICROSIMULATIONS: A PARALLEL DISCRETE-EVENT QUEUE-BASED APPROACH FOR SPEED AND SCALE

Sunil Thulasidasan
Stephan Eidenbenz

Computer, Computational and Statistical Sciences
Los Alamos National Laboratory
Los Alamos, NM 87545, U.S.A

ABSTRACT

We present *FastTrans* – a parallel, distributed-memory simulator for transportation networks that uses a queue-based event-driven approach to traffic microsimulation. Queue-based simulation models have been shown to be significantly faster than cellular-automata type approaches, sacrificing spatial granularity for speed, while preserving link and intersection dynamics with high fidelity. Significant advances over previous work include the size of the simulated network, support for dynamic responses to congestion and the absence of precomputed routes – all routing calculations are executed online. We present initial results from a scalability study using a real-world network from the North-East region of the United States comprising over 1.5 million network elements and over 25 million vehicular trips. Simulation of an entire day's worth of realistic vehicular itineraries involving approximately five billion simulated events executes in less than an hour of wall-clock time on a distributed computing cluster. Initial results suggest almost linear speed-ups with cluster size.

1 INTRODUCTION

Approaches to transportation simulation have been numerous and varied, spanning an interesting variety of simulation paradigms – from fluid-based aggregate models to detailed microsimulations, and from time-stepped approaches to discrete-event models. The paradigm employed is usually dependent on the problem under study: fluid models nicely describe macroscopic behavior of networks, while cellular-automata based microsimulations are more suitable for problems that require a higher level of spatial granularity – study of vehicular emissions, for instance. While the fundamental trade-offs are between speed and resolution, depending on the question under consideration, a simulation model can be both accurate and fast. Here, we present a queue-based approach to simulating vehicular dynamics starting with the premise that the questions of interest are vehicular dynamics at the intersection and link levels. A queue-based approach can be used to quickly answer time-critical questions like evacuation times and optimal exit routes from a city in an emergency situation. Further, such models can also be used to guide infrastructure planning activitities like the impact of building a new road or a relief-route, or conversely, the impact of disabling a route. Traditionally, traffic microsimulations of transportation networks have employed a time-stepped cellular-automata approach. A prominent example of this is TRANSIMS (Smith et al. 1995), developed at the Los Alamos National Laboratory, where vehicular dynamics are modeled at a high level of spatial granularity. This allows one to capture phenomena such as lane changing, and vehicular emissions, but comes at a high computational cost. Other simulators using the microscopic simulation paradigm include CORSIM (Prevedouros and Wang 1999), VISSIM (Concepts 2001) and PARAMICS (Cameron and Duncan 1996).

An alternative approach to traffic simulation, where road links are modeled as queues, was described in (Eissfeldt et al. 2006). A time-stepped, parallel implementation of this approach is described in (Cetin, Burri, and Nagel 2002). (Charypar, Axhausen, and Nagel 2006) observe that time-stepped computations are frequently unnecessary based on the fact that in a given road network, there are large numbers of links on which the traffic flow density is very low. Updating these links every time step are often "null ops" and wastes computational cycles. To overcome this inefficiency, the authors propose a discrete-event queue-based model for a sequential, single-processor environment.

A parallel discrete-event approach to microsimulations was described in (Perumalla 2006), though the modeling paradigm employed here is conceptually closer to the cellular-automata approach. Experimental results presented here on a 1000-node grid network indicate speed-ups of upto 1000 over realtime.

The FastTrans approach is to combine the discrete-event queue model with scalable parallelization. This allows us to simulate large-scale, real-world networks and realistic traffic scenarios involving tens of millions of vehicles in a fraction of real time. Also, since FastTrans simulates the behavior of each vehicle or traveler at the individual entity level, it retains some of the advantages of microsimulations. In addition, the congestion feature implemented in FastTrans, which updates the state of the routing graph on all simulation processes, allows one to observe the macroscopic behavior of the network.

The rest of the paper is organized as follows: in Section 2 we briefly describe the queue-based model employed in FastTrans. Section 3 describes the software architecture and distributed design of FastTrans; realistic activity modeling is described in Section 3.2; Section 3.3 gives an overview of the FastTrans routing module; Section 4 describes the computational set-up and experimental results on a large, real-world road network. Finally, Section 5 concludes with directions for future work. The main focus in this paper is the modeling method, software implementation and scaling; thorough validation and testing is reserved for future work.

2 EVENT-DRIVEN QUEUE-BASED MODEL

In the queue-model of road networks, each road link is modeled as a queue, whose properties are described by two main parameters: their physical capacity – i.e the number of bumper-to-bumper vehicles that can be accommodated on the link – and the flow rate of the link. The flow rate indicates the number of vehicles that can transit through the link and is calculated by the procedures established in the Highway Capacity Manual(Transportation Research Board 2000).

Further, each queue is attached to a network node; nodes represent a traffic intersection, or a point where the road link diverges (a freeway exit, for example). The scheduling policy for vehicle departures from a node is determined by the type of intersection that is being modeled. The initial implementation of FastTrans uses round-robin and First-Come-First-Serve schedulers though signalized and other types of schedulers can be easily incorporated into the design.

The parameters of flow rate and physical capacity also allow us to model congestion by dynamically adjusting the flow rate (as happens during a lane-closure, for instance) and also by blocking the link when its physical capacity has been reached. More details are provided in Section 3.4. Note that lanes per-se are not modeled in FastTrans; instead the effect of lane capacity is captured through flow-rate and physical capacity.

3 SOFTWARE ARCHITECTURE

FastTrans is written in C++, and is built on top of SimCore, a generic framework that provides application programming interfaces (APIs) for building discrete-event simulation models. SimCore provides generic constructs like *entities* and *services* that can be adapted to build objects in a simulation model. SimCore also provides message objects for communicating between simulation instances in a parallel environment. Previous examples of simulators developed using SimCore can be found in (Wauopotitsch et al. 2006) and (Galli et al. 2009). The former describes a packet-level network simulator for modeling large-scale communication networks; the latter describes ActivitySim, a discrete-event agent-based activity generator for infrastructure simulations.

For message passing and synchronization, we use the Prime Scalable Simulation Framework (PrimeSSF) (**?**), a parallel simulation engine that employs a conservative synchronization mechanism and supports both shared-memory and distributed-memory implementations. Since scalability is one of our prime considerations, FastTrans is implemented as a pure distributed-memory application. All message-passing is implemented using the MPI message passing interface (MPI 2008). Figure 1 illustrates the architectural layout of FastTrans.

Entities in the simulation are the fixed elements of the road network – road links and traffic intersections. All the properties of the network – capacity, flow rate etc. – are data members of the relevant entity class. The scheduler at a traffic intersection is implemented as a *service* on the traffic-node entity. The modular design of FastTrans allows the scheduling policy to be easily changed by simply replacing one scheduling service with another. The mobile elements of the simulation (vehicles) are represented using messages; vehicle objects (messages) are created and destroyed during the start and end of a trip, respectively. The main state variables associated with a vehicle are source, destination and the route vector. Route vectors are computed at the start of the trip by the FastTrans routing module; for this, we maintain a copy of the connectivity graph on each simulation process. The routing algorithm is described in more detail in Section 3.3. .

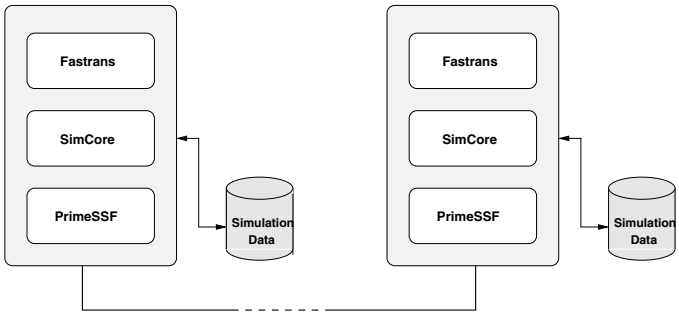

Figure 1: FastTrans software architecture

3.1 Distributed-memory Model of Queues

Since FastTrans uses a distributed-memory model, different entities of the road network are created in different memory spaces during simulation start-up. Each logical process (LP) in the simulation is an instance of a FastTrans executable running on a compute node.

Traffic intersections are distributed across LPs in a round-robin manner using a modulo function based on the numerical identifier of the node. This scheme allows fast lookups without the need o maintain expensive look-up tables that map nodes to LPs. Links (queues) are partitioned in a slightly different manner: note that each link has two intersection end-points – the source node, from where the link originates and the sink node, where the link terminates. FastTrans places each link on the same LP as its sink node based on the observation that more messages are exchanged between the sink node and the link (vehicle arrival, vehicle dequeue et cetera) than between the source node and the link. Placing the link and the sink node on the same process reduces message-passing overhead. A design schematic of the distributed architecture of FastTrans is shown in Figure 2.

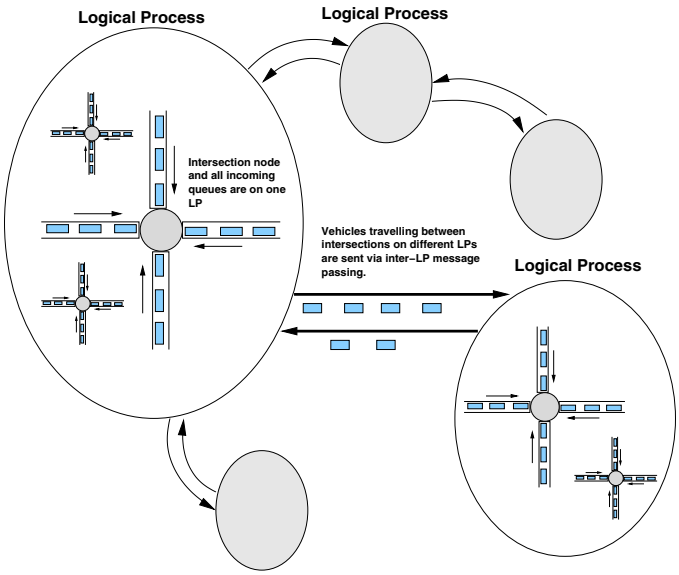

Figure 2: Distributed-memory model of FastTrans

3.2 Activity Modeling

For realistic vehicular trip generation, FastTrans leverages the detailed activity modeling capabilities of the Urban Population Mobility modeling tools, that was developed as part of the TRANSIMS project. A list of activities is generated for each member of a synthetic population, the set of activities being based on the demographics of that household. The activities are also sensitive to the network; activity locations reflect land-use and employment data based on census data. Alternatively, FastTrans can also be coupled to the agent-based activity generator, ActivitySim, mentioned in Section 3.

Each individual in the synthetic population receives an activity list with the following attributes: type (home, work, shopping et cetera), time, duration, mode and location. The activities are then filtered and pre-processed to generate a complete set of vehicular trips for the region being modeled. Trip density during various times of the day is illustrated in Figure 3, with peak times are observed around noon and 6 PM. Routes are not calculated in the pre-processing stage; all routing is done during the simulation and is described in more detail in the next section.

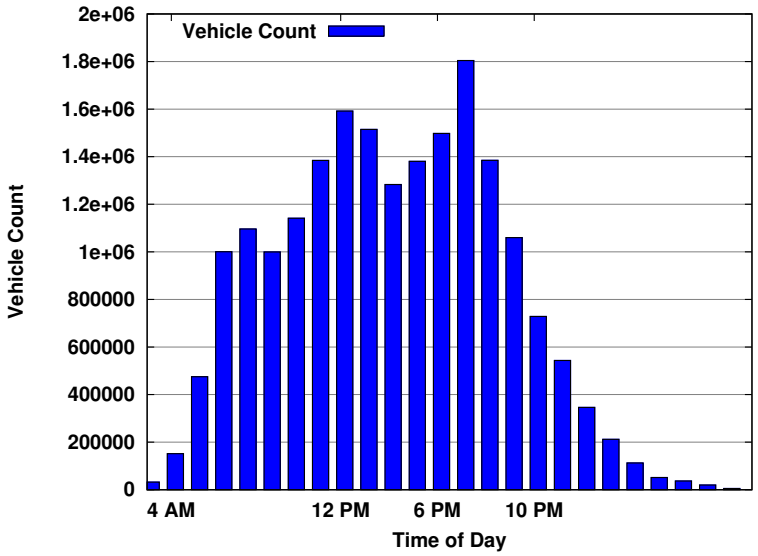

Figure 3: Vehicular count vs. time of simulated day

3.3 Routing

Calculating routes during the simulation allows us to observe dynamic responses to congestion (a side effect being significant reduction in input file-size and processing time). However, route computations on large graphs can easily become a bottleneck, leading to severe performance degradation; vigorous optimizations are required to make online route computation feasible.

The routing algorithm employed by FastTrans is a heuristic variant of the classic Dijkstra (Dijkstra 1959) shortest path algorithm, that uses geometric properties of the graph to speed-up shortest-path computations. A detailed performance analysis of the performance of the Dijkstra algorithm versus heuristics-based routing on real-world road-network graphs can be found in (**?**).

Routing experiments were conducted with different variations of the Dijkstra algorithm. Initially, FastTrans used a naive implementation of Dijkstra, where shortest-path trees, rooted at the source, are constructed for each routing query. This is obviously inefficient as we are only interested in a shortest-path to the destination node. While theoretically, the time complexity of path-computation for a source-destination pair is asymptotically the same as computing a shortest-path tree rooted at the source, we observe that in practice, there are often enormous differences in running times, especially for routing queries in real-world road networks, where source and destination nodes often lie in close physical proximity to each other. An optimized implementation of Dijkstra – where the search loop is terminated upon reaching the destination – improved running time significantly. Further optimizations include smart label-reset (where only nodes explored in a previous routing computation are reset), and the use of efficient data structures to represent the graph.

The current version of FastTrans uses $A*$ search (Sedgewick and Vitter 1986), a variant of Dijkstra that employs a heuristic cost-function to bias the direction of the search towards the destination node. $A*$ exploits the near-Euclidean properties of road network graphs; nodes that lie closer to the destination are more likely to be explored than nodes lying further away. This results in the search arriving at the destination node much quicker than Dijkstra, where the search tree is expanded in a circular manner centered around the source node. To bias the search towards the destination, we assign a cost to each fringe vertex as follows: given source s, and destination t, for each fringe vertex v, cost $C(v)$ is defined as:

$$C(v) = l(s,v) + D(v,t).$$

where $l(s,v)$ is the shortest path length from s to v as before, and $D(v,t)$ is the estimated cost from v to t based on the Euclidean distance between v and t. Since we are interested in the shortest path in terms of time rather than distance, $l(v,s)$ is the time-cost of the path from s to v based on link-speeds and flow-rates, and $D(v,t)$ is defined as:

$$D(v,t) = \frac{E(v,t)}{V_{max}}$$

where $E(v,t)$ is the Euclidean distance from v to t and V_{max} is the maximum allowable speed in the network. Note that the resulting paths using $A*$ may not always be optimal; the road graph is not strictly Euclidean, and the cost is expressed in time, not distance. Nevertheless, our experiments and those described in (**?**) have shown that the resulting paths are near-optimal.

Figures 4 and 5 illustrate the performance of $A*$ versus an optimized version of Dijkstra on the real-world road network that was used in the scaling studies in this paper. Figure 4 illustrates a typical-case scenario, with $A*$ expanding about 25 percent of the nodes (shown in blue) compared to Dijkstra (shown in green). Figure 5 is an example where $A*$ performs markedly well, expanding only about one percent of the nodes.

Figure 6 shows, on a logarithmic scale, the speed-up as well as the routing overhead for the three implementations (un-optimized Dijkstra, optimized Dijkstra and $A*$) from a code-profiling exercise of a sequential simulation run. These were executed on a 3 GHz Mac-Pro work-station for 20,000 itineraries that were randomly sampled from the study-set. About 63 percent of the execution time is currently spent inside the routing module of FastTrans (Figure 7), in a single-CPU run. In parallel scenarios this number is expected be lower, because of the additional overhead from message passing.

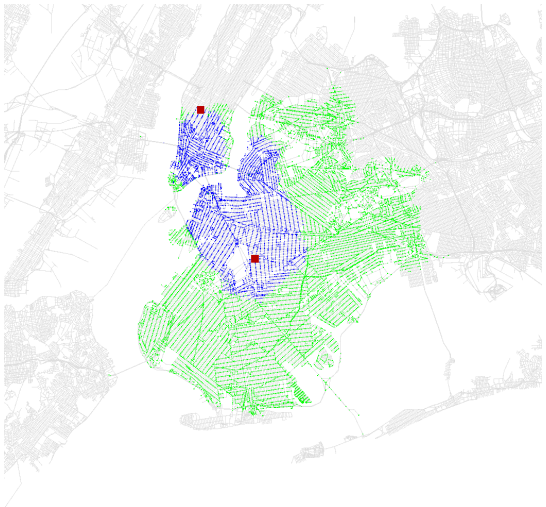

Figure 4: Nodes expanded in $A*$ (shown in blue) vs. Dijkstra (shown in green). The red squares are the source and destination nodes, with the source being at the center. Dijkstra expands the search tree in all directions from the source node, while $A*$ is more directed.

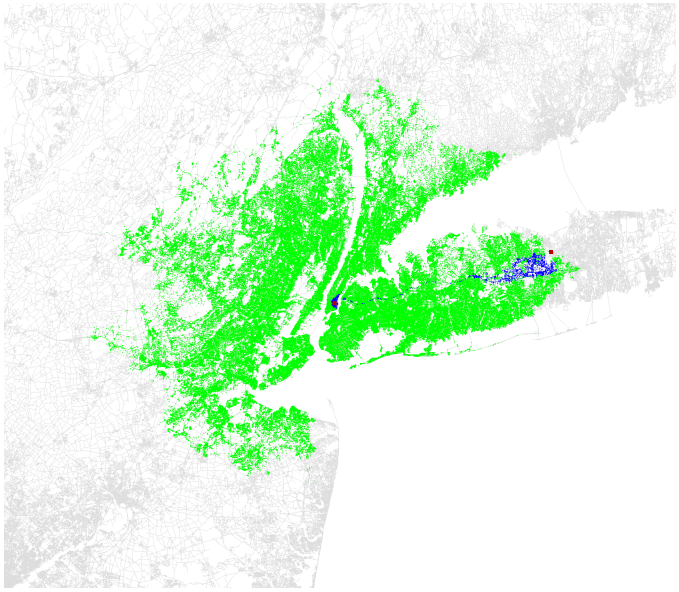

Figure 5: A routing query where $A*$ performs markedly better, expanding only about one percent of the nodes (blue) compared to Dijkstra (green)

3.4 Modeling Congestion: Locally and Globally

Congestion occurs at a link when the difference in the incoming rate of cars (rate at which cars are dequeued on to the link from the upstream node) and the outgoing rate (rate at which the downstream node removes cars from the link) causes a marked decrease in vehicular speed, compared to the free-flow speed for that link. This can be a result of a busy downstream intersection or due to a full or partial lane closure (during an accident for instance). The effect of lane closures can be modeled by adjusting the flow rate of the link, which involves increasing the delay between successive dequeuing events on that link. Also, once the occupancy on a link reaches maximum capacity, upstream nodes are blocked from dequeuing any further vehicles onto the link. For links and nodes on different LPs, this is achieved through message passing.

The effects of congestion are propagated backward to nodes and links that are further upstream, mimicking the behavior of traffic jams that spill backwards. Once vehicles start to leave the downstream congested links, this information is again propagated to the upstream links. To prevent upstream links from immediately sending vehicles onto a previously congested link, we model a "vehicular gap" that propagates backwards, similar to the mechanism described in (Charypar, Axhausen, and Nagel 2006).

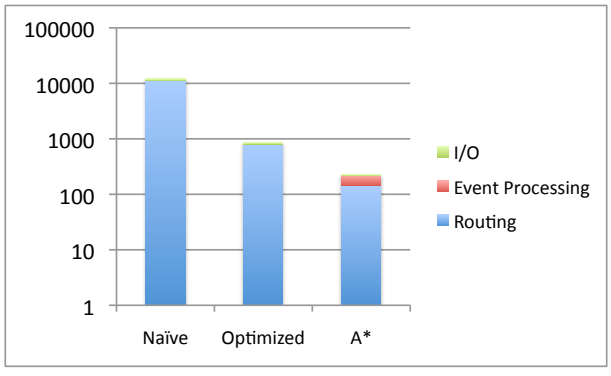

Figure 6: Execution time (in seconds) and routing overhead for unoptimized Dijkstra, optimized Dijkstra and $A*$

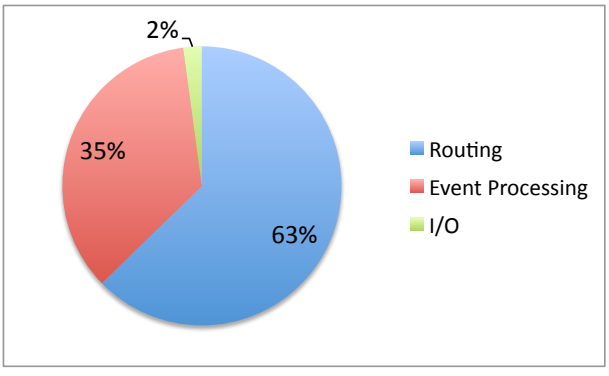

Figure 7: Routing-module overhead for FastTrans with $A*$ search in a single-CPU run

Further, we also model the global effects of congestion. Commuters often receive congestion notifications on busy freeways through radio updates or in-car satellite-based navigation systems. This effect is captured by having each LP distribute congestion information about links on that LP to other LPs with a pre-specified delay. Admittedly, not all travelers react to congestion; this is modeled by probabilistically determining if a routing query will use the updated link travel times or use the older link times.

4 EXPERIMENTAL RESULTS

4.1 Simulating a Real-World Road Network

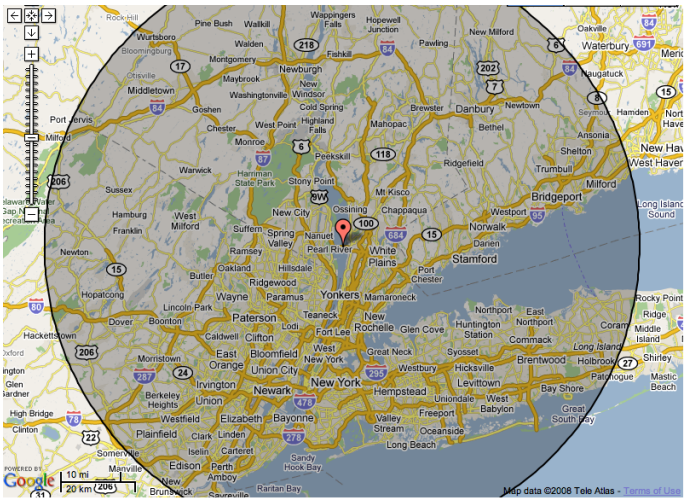

Figure 8: Road network from the North-East region of the US

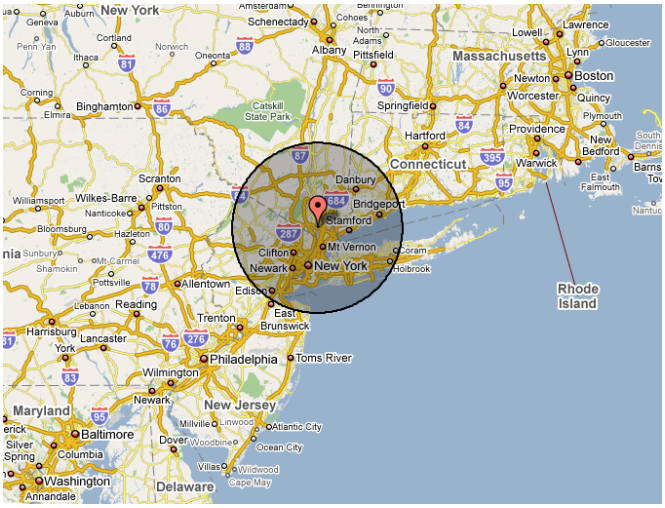

Figure 9: Study Area: Zoomed-out perspective

For the simulation and scaling studies of FastTrans, we use a real-world road network from the Northeast region of the United States, covering most of the urban regions of New York, and parts of New Jersey and Connecticut (Figures 8 and 9). The network graphs consists of approximately 500,000 nodes and 1.1 million links. The road network input is processed from NAVTEQ data (NAVTEQ 2000) and consists of detailed information such as link length, number of lanes, free-flow speed and geographical position information. Figure 10 illustrates the road-density of the network under consideration. In the experiments, we simulate an entire day's worth of itineraries from the activity modeler (see Section 3.2) that use private automobiles (mass transit is currently not implemented). This amounts to approximately 25 million vehicular trips.

4.2 Computational Setup

All experiments were carried out on Coyote, a high performance GNU/Linux cluster at the Los Alamos National Laboratory. Each cluster node consists of two 64-bit AMD Opteron 2.6GHz processors with 8GB memory, running 64-bit RedHat Fedora Core 3. Nodes communicate with each other via a high speed Infiniband interconnect. Due to resource usage limits, we were restricted to using a maximum of 512 CPUs for our scaling studies, starting with a minimum of 32 CPUs. Interprocess communication was carried out using the OpenMPI message passing interface (OpenMPI).

4.3 Simulation Results

Figures 11 and 12 illustrate the wall-clock execution times and speed-up over real time, respectively, for distributed memory runs of FastTrans on increasingly larger cluster sizes. We notice that the metrics follow the ideal curve (depicted in red) until about the 128-CPU mark. Beyond this, the performance gains diminish, even though performance continues to improve significantly upto the maximum cluster-size. We expect performance to continue to improve sub-linearly beyond the 512-node cluster-size.

The memory usage of the simulator per compute node is depicted in figure 14. Memory used per node remains more or less constant even with increasing cluster size since most of the memory usage on a node results from the size of the

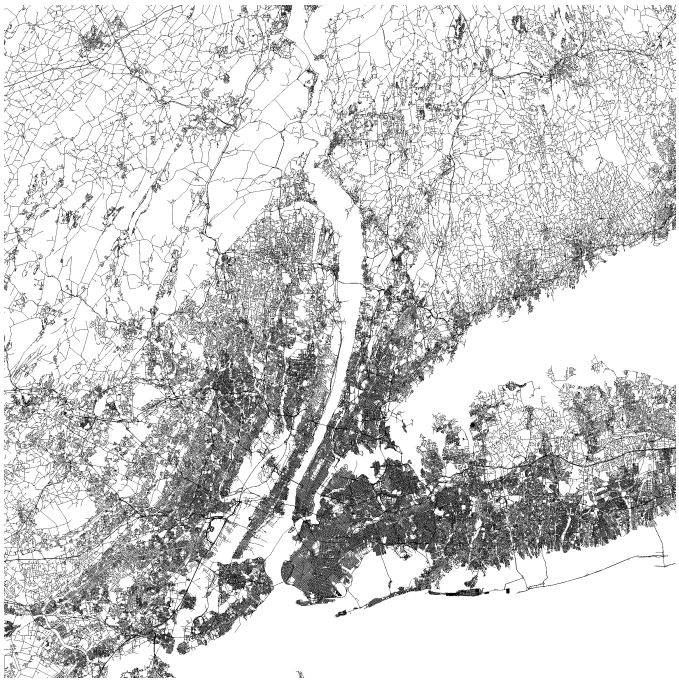

Figure 10: Road network graph of study region showing street density

routing graph, a copy of which is kept on each LP. Note that memory usage from events, on a per-node basis, reduces with increasing cluster-size. Figure 13 shows the event-processing throughput for various cluster sizes, and as expected, this is similar to the execution-time curve.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented initial results from a parallel discrete-event queue based approach to simulating vehicular networks with significant speed-ups over real time for large scale real-world simulations. Further, routing optimizations facilitate fast online route computations, allowing us to observe dynamic responses to congestion. While the queue-based approach does sacrifice spatial granularity, for capturing link dynamics and congestion, this method is a strong candidate. Experiments on large, real-world road networks indicate that this method can be successfully employed in quick turn-around studies like emergency evacuation and disaster modeling. Areas for future research include further validation studies using larger networks and activity sets, incorporating more sophisticated scheduling algorithms at traffic intersections, investigating the use of topological partitioning to minimize message passing overhead and augmenting the simulator with multi-modal (e.g. mass-transit) transportation capabilities.

ACKNOWLEDGEMENTS

The authors would like to thank Phil Romero, Shiva Kasivishwanathan and Deborah Kubicek of the Los Alamos National Laboratory.

REFERENCES

Cameron, G., and G. Duncan. 1996. Paramics: Parallel microscopic simulation of road traffic. In *Journal of SuperComputing*.

Cetin, N., A. Burri, and K. Nagel. 2002. Parallel queue model approach to traffic microsimulations.

Charypar, D., K. Axhausen, and K. Nagel. 2006. An event-driven queue-based microsimulation of traffic flow. In *Transport Systems Planning and Transport Telematics*.

Concepts, I. T. 2001. Vissim simulation tool. In *http://www.itc-world.com/VISSIMinfo.htm*.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1.

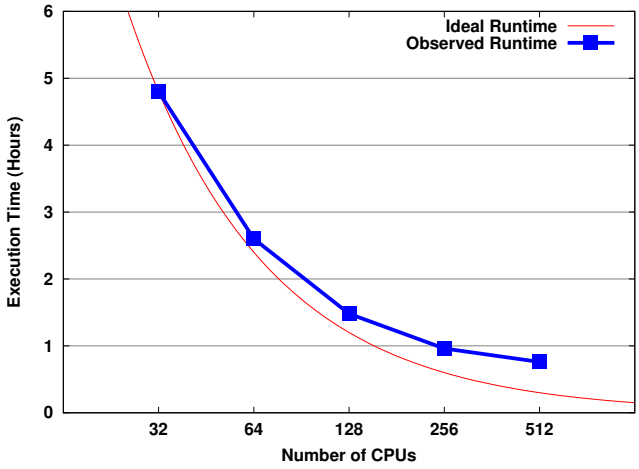

Figure 11: Wall-clock execution time for 25M car trips

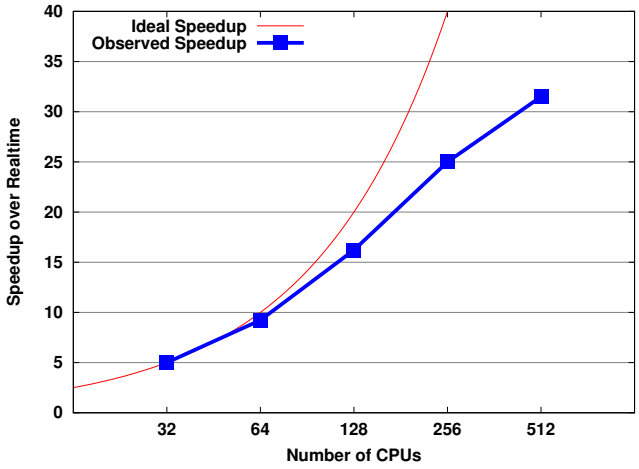

Figure 12: Corresponding speed-up ratio over real-time

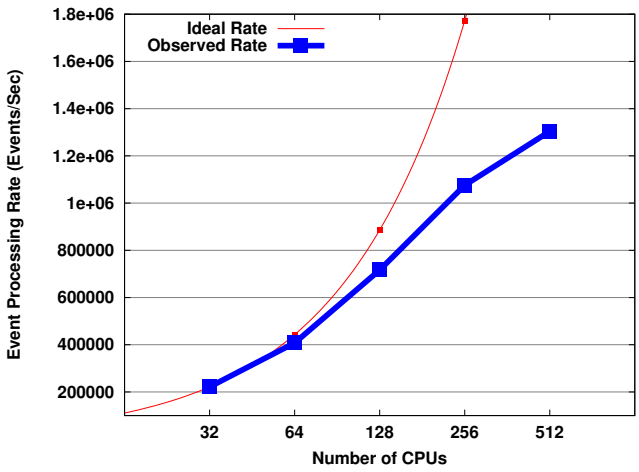

Figure 13: Number of simulation events processed per second

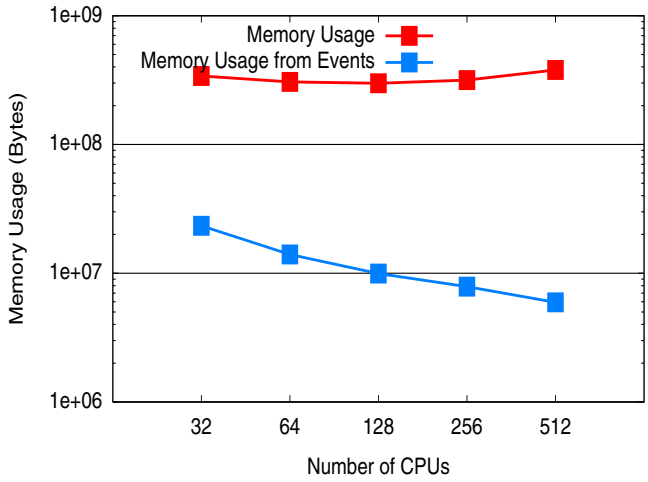

Figure 14: Total memory and event memory usage per node

Eissfeldt, N., D. Krajzewicz, K. Nagel, and P. Wagner. 2006. Simulating traffic flow with queues.

Galli, E., S. Eidenbenz, S. Mniszewski, C. Teuscher, and L. Cuellar. 2009. Activitysim: Large-scale agent-based activity generation for infrastructure simulation. In *Proceedings of the 2009 Spring Simulation Conference*.

MPI 2008. *The mpi message passing interface*. http://www.mcs.anl.gov/research/projects/mpi/.

NAVTEQ 2000. *Navteq*. http://www.navteq.com.

OpenMPI. *Openmpi: Open source high performance computing*. http://www.open-mpi.org.

Perumalla, K. 2006. A systems approach to scalable transportation network modeling. In *2006 Winter Simulation Conference*.

Prevedouros, P. D., and Y. Wang. 1999. Simulation of a large freeway/ arterial network with integration, tsis/corsim and watsim. In *Transportation Research Record 1678*, 197–207.

Sedgewick, R., and J. Vitter. 1986. Shortest paths in euclideann graphs. *Algorithmica* 1.

Smith, L., R. Beckman, D. Anson, K. Nagel, and M. E. Williams. 1995. Transims: Transportation analysis and simulation system. In *Proceedings of the Fifth National Conference on Transportation Planning Methods*. Seattle,Washington.

Transportation Research Board 2000. *Highway capacity manual*. Transportation Research Board.

Wauopotitsch, R., S. Eidenbenz, J. P. Smith, and L. Kroc. 2006. Multi-scale integrated information and telecommunications system(miits): First results from a large scale end-to-end network simulator. In *Proceedings of the 2006 Winter Simulation*

Conference, ed. L. R. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 2132–2139. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

AUTHOR BIOGRAPHIES

SUNIL THULASIDASAN is a Technical Staff Member in the Information Sciences group at the Los Alamos National Laboratory. He received his Masters Degree in Computer Science from the University of Southern California in 2001 and his Bachelor's Degree in Computer Science and Engineering from the University of Kerala, India in 1998. He has published in the areas of Internet congestion control, active queue management, high performance computing and wireless networking. He has been active in the area of discrete-event simulation software development for a number of years, contributing modules to the NS-2 network simulator, and is the author of the FastTrans transportation simulator described in this paper. He can be reached at <sunil@lanl.gov>.

STEPHAN EIDENBENZ is a Technical Staff Member in the Information Sciences group at the Los Alamos National Laboratory, where he leads the MIITS project that models large-scale infrastructure networks. He received his Ph.D. in Computer Science from the Swiss Federal Institute of Technology, Zurich in 2000, and obtained a Bachelor's degree in business administration from GSBA in Zurich in 1999. He has published in a large number of fields, including approximation algorithms, visibility problems in polygons, error-modeling in computational biology and network protocol design. He is also an active reviewer and ahs been a member of the ACM Mobihoc and Infocom program committees. He can be reached at <eidenben@lanl.gov>