Designing Systems for Large-Scale, Discrete-Event Simulations: Experiences with the FastTrans Parallel Microsimulator

Sunil Thulasidasan Shiva Kasiviswanathan Stephan Eidenbenz
Emanuele Galli Susan Mniszewski Phillip Romero

Los Alamos National Laboratory
Los Alamos, NM, USA
{sunil,kasivisw,eidenben,egalli,smm,prr}@lanl.gov

Abstract*—We describe the various aspects involved in building FastTrans, a scalable, parallel microsimulator for transportation networks that can simulate and route tens of millions of vehicles on real-world road networks in a fraction of real time. Vehicular trips are generated using agent-based simulations that provide realistic, daily activity schedules for a synthetic population of millions of intelligent agents. We use parallel discrete-event simulation techniques and distributed-memory algorithms to scale these simulations to over one thousand compute nodes. We present various optimizations for speeding up simulation execution times, including (i) a set of routing algorithms such as variations of Dijkstra's shortest path algorithm and heuristic-based A^* search, and (ii) a number of different partitioning schemes for load balancing, including geographic partitioning (that assigns simulation entities that are geographically close by to the same processor) and scattering (that assigns geographically close by entities to different processors). Our main findings include: (i) A^* significantly outperforms other routing algorithms while computing near-optimal paths; (ii) surprisingly, scattering outperforms more sophisticated partitioning schemes by achieving near-perfect load-balancing. With optimized routing and partitioning, FastTrans is able to simulate a full 24 hour work-day in New York – involving over one million road links and approximately 25 million vehicular trips – in less than one hour of wall-clock time on a 512-node cluster.

Keywords— Parallel Discrete-Event Simulation, Transportation Simulation, Load Balancing.

I. INTRODUCTION

How do we build systems that can realistically simulate, at a high level of detail, the traffic patterns resulting from the activities of tens of millions of people within a geographic region? What strategies do we employ such that we maximize the usage of the processor cycles available to us? And how do we scale these systems to hundreds, even thousands of processors on high performance computing clusters, so that they execute in a fraction of real time?

In this paper we present various aspects of designing, building, and optimizing FastTrans – a scalable, parallel microsimulator for transportation networks that can simulate and route tens of millions of vehicular trips on real-world road networks. Using parallel discrete-event simulation techniques [9] and distributed-memory algorithms, we are able to model transportation networks of large geographic regions – consisting of over a million road network elements and over 20 million vehicles – and scale these simulations to execute on large, high performance clusters upto 20 times faster than real time.

Large-scale simulations are an important tool in the emerging field of infrastructure modeling, where simulating the behavior of millions of entities and their interactions with various interdependent infrastructure networks (like transportation, communication, electric power) demand significant computational resources. At the Los Alamos National Laboratory, FastTrans is one of the key modules in a suite of simulators that have been built using a common parallel simulation framework for infrastructure modeling. The common framework allows for easy integration of the various modules; for instance, we have been able to integrate FastTrans (introduced in [20]) with ActivitySim, an agent-based simulator (introduced in [10]) that provides daily activity generation, scheduling and execution for a synthetic population of intelligent agents. This allows us to generate realistic activity schedules for millions of intelligent agents, route the tens of millions of vehicular trips generated from these activities, and observe how traffic conditions and the variations they cause in expected travel times impact activity scheduling and vice versa.

Organization. We present the key aspects of the road-network and activity modeling philosophy in Section II. We discuss the software architecture of FastTrans, ActivitySim, and the interaction between these modules in Section III. In Section IV, we describe the various design choices for routing, partitioning, and load balancing that were used to optimize the performance of our simulations. Section V contains scaling results for parallel runs. Conclusions are in Section VI.

II. TRAFFIC MODELING THROUGH REALISTIC ACTIVITY GENERATION

A. Modeling the Road Network

Approaches to transportation simulation have spanned a variety of simulation paradigms: from fluid-based aggregate models to detailed microsimulations and from time-stepped approaches to discrete-event models. Fluid models nicely describe the macroscopic behavior of networks, while microsimulations (where the behavior of each individual vehicle is simulated) are more suitable for problems that require a higher level of spatial granularity – study of vehicular emissions, for instance.

Traditionally, traffic microsimulations have employed a time-stepped cellular-automata approach [3], [6], [15], [16], [19]. A prominent example of this is TRANSIMS [19], developed at the Los Alamos National Laboratory which models vehicular dynamics such as lane changing and emissions, but at a high computational cost. In this paper, we present a queue-based, discrete-event approach to traffic microsimulation – which sacrifices some amount of spatial granularity for speed – 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 activities like the impact of building a new road or a relief-route, or conversely, the impact of disabling a route.

The queue-based traffic simulation model was first developed in [8]. A time-stepped parallel implementation of this approach was later developed by Cetin *et al.* [4]. Subsequently, Charypar *et al.* [5] observed that time-stepped computations are frequently unnecessary since in a given road network, there are a large number of links on which the traffic flow densities are very low. Updating these links every time step are often "null ops" and therefore, waste computational cycles. To overcome this inefficiency, a discrete-event queue based model is proposed in [5] for a sequential, single-processor environment.

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 model of FastTrans captures the non-localized effects of congestion, allowing us to observe the macroscopic nature of the network.

B. Queue Model of Road Networks

In the queue model, each road link is modeled as a queue, whose properties are described by two main parameters: (1) their physical capacity (i.e., the number of bumper-to-bumper vehicles that can be accommodated on the link) and (2) 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 [21]. Each queue is attached to a network node which represents 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.
Further, to model congestion, FastTrans builds upon the techniques introduced in the transportation simulation literature [3]–[5], [15]. The parameters of flow rate and physical capacity allow us to capture 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. In this case, upstream nodes are blocked from adding any further vehicles onto the link, mimicking the behavior of traffic jams that spill backwards. Once vehicles start to leave the downstream congested links, this information is propagated to the upstream links.

C. Activity Modeling

The ActivitySim agent-based simulation software (introduced in [10]) provides daily activity generation, scheduling and execution for a synthetic population of intelligent agents. Persons, locations, households, and zones comprise the *entities* used to model a geographical area. A *person* is characterized by age, gender, income, household, and position in the house (like principle income generator). Other state-variables of a person include current activity and location, demographics, preferences for activity locations, and an activity schedule. A *location* tracks persons as they participate in activities. A *household* is associated with a location, has aggregated income and members (i.e., a family). A *zone* is an aggregation of locations used when selecting where a given activity will take place.

The persons, locations, households, and zones are provided as input at runtime along with a specification of a set of activity types. A person's schedule consists of a sequence of activities (e.g., home, work, school) with each activity having a start time, activity location, and duration. Each person re-evaluates and modifies their activity schedule as required and plans new activities 24 hours in advance. The schedule can be generated through different methodologies: random, pre-calculated, or utility-driven. In the random method, a random activity type is added to the schedule; in the pre-calculated method, a daily schedule obtained from the National Transportation Survey [22] is used; in the utility-driven method, each agent generates a daily schedule based on personal characteristics, previously performed activities and current priorities. Each activity of an agent is associated with a utility function (which depends on activity duration), a priority function (which depends on time passed since that activity was last performed), and additional constraints (acceptable start and end times) that cannot be violated. A more thorough comparison of these activity generating schemes is presented in [10]. We will use utility-driven activity generation scheme for the rest of this paper.

III. Software Architecture

All simulation modules are built on top of SimCore, a generic framework written in C++ that provides application programming interfaces (APIs) for building distributed-memory, discrete-event simulation applications. 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.

For message passing and synchronization, we use the Prime Scalable Simulation Framework (PrimeSSF) [17], a parallel simulation engine that employs a conservative synchronization mechanism. PrimeSSF supports both shared-memory and distributed-memory implementations though, for scalability reasons, all the simulators described in this paper are pure distributed-memory applications. Message-passing is implemented using the MPI message passing interface [13].

A. *FastTrans Architecture*

FastTrans is written in C++ and built using the constructs in the SimCore library. In FastTrans, simulation entities are the fixed elements of the road network – road links and traffic intersections. All the properties of the network – capacity, flow rate, etc – are members of the relevant entity class. The scheduling logic at a traffic intersection is implemented as a *service* on the traffic-node entity. The modular design 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 IV-A). The input data to FastTrans is the road-network graph for the region being simulated – indicating connectivity, road length, speed limits, lane capacity, etc – and vehicular itineraries indicating source, destination and start time of a given trip. The trip inputs can be read from a file, or can be sent using messages when FastTrans is coupled with ActivitySim.

Since FastTrans uses a distributed-memory model, different entities of the road network are created in different memory spaces during simulation start-up. Each simulation process (also known as Logical Process, or LP) in the simulation is an instance of a FastTrans executable running on a compute node. A design schematic of the distributed architecture of FastTrans is shown in Figure 1. Traffic intersections are distributed across LPs according to the partitioning strategy employed – geographic, random, etc., which we describe in more detail in Section IV-B. Links (queues) are partitioned in a slightly different manner: each link is placed on the same LP as its terminating point[1] based on the observation that more messages are exchanged between the sink node and the link (vehicle arrival, vehicle dequeue and so on) than between the source node and the link. Placing the link and the sink node on the same process allows us to reduce message-passing overhead. A more detailed description of the architecture is presented in [20].

[1]Note that each link is attached to two intersection end-points – the source node, from where the link originates and the sink node, where the link terminates.

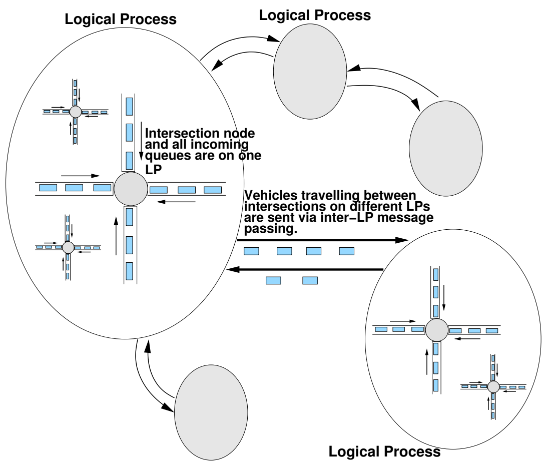
Logical Process
Logical Process
Intersection node and all incoming queues are on one LP
Vehicles travelling between intersections on different LPs are sent via inter–LP message passing.
Logical Process

Fig. 1. Distributed-memory model of FastTrans.

B. *ActivitySim Architecture*

ActivitySim, written in C++ and built on top of SimCore, implements agent-based models combining traditional agent technology from the artificial intelligence community and numerical methods for activity schedule calculations. The simulation entities in ActivitySim are individuals and locations, and generic agent classes are used to implement entities with intelligent behavior (people). The behavior of a person in ActivitySim is modeled using services that implement cognitive functionality such as “perceive”, “think”, and “act”. This allows people to change and adapt their activity schedules. ActivitySim can run on a single workstation as well as on high performance computing clusters. A more detailed description of the ActivitySim architecture is provided in [10], including a description of the *AgentCore* layer that provides the cognition functionality.

C. *Integrated Simulations*

Both FastTrans and ActivitySim can be compiled as separate applications and run independently of each other, each binary being statically linked against the SimCore, PrimeSSF, and MPI libraries. In this case, vehicle trips are pre-computed and fed into the FastTrans module, while for ActivitySim running in independent mode, activities that cause individuals to travel from one location to another are processed internally, without feedback regarding actual travel times that would have depended on traffic conditions in the road network.

A more realistic (and interesting) approach would be to combine the two simulators, and observe how traffic conditions and the variations they cause in expected travel times impact activity scheduling, and vice versa. The modular architecture of the simulators and the use of a common simulation framework allows us to integrate the two modules and achieve this feedback mechanism with relative ease.

In the coupled simulation scenario, events in ActivitySim that trigger a road trip (Figure 2) are sent to FastTrans via

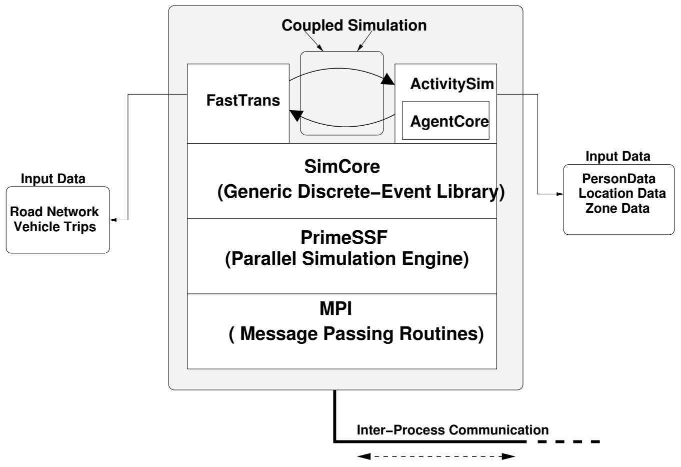

Fig. 2. Modular software architecture of the integrated simulator.

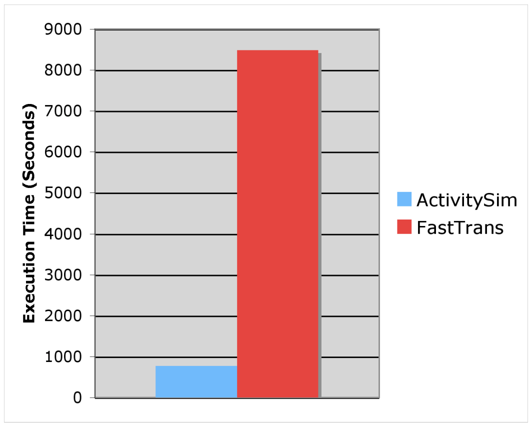

Fig. 3. Comparison of the execution times of the two modules for a 32-CPU parallel run of a medium sized US city

messages. The actual trip is simulated inside FastTrans, and upon arrival at the destination, an event is sent to ActivitySim. In this case, unexpected delays in arrival time due to road network congestion inside FastTrans can trigger recomputing of activity schedules – behavior that is not observed when the simulations are running independently. We note here that the entity partitioning scheme used inside the two modules are completely independent of each other; spatial mapping information is needed to map the activity locations inside ActivitySim to elements in the road network.

IV. Performance Tuning

Detailed microsimulations are frequently used in modeling critical infrastructures such as transportation and communication networks in a given region. Simulations involving natural and man-made disasters often require iterations through multiple scenarios – to determine the best evacuation strategy during an earthquake, for instance – implying that such simulations need to execute much faster than real-time to be useful. When combining the two simulation modules, ActivitySim and FastTrans, we observe that the execution time is overwhelmingly dominated by the FastTrans module (Figure 3). Consequently, we focus our optimization efforts on improving the performance of FastTrans.

A. Routing in FastTrans

Routing is the most important module of FastTrans – the algorithm and parameters used to route vehicles are critical to the validity of the simulation model. In addition, it also accounts for more than half of the total execution time. We use *dynamic routing* in FastTrans; calculating routes dynamically allows us to achieve fast responses to congestion with the additional benefit of reduction in the input data size[2]. However, route computations on large graphs can easily become a bottleneck, leading to severe performance degradation. We use several optimizations to make online route computations feasible in FastTrans where we require the shortest path between a pair of nodes. We also exploit the fact that the problem domain (road-network routing) allows us to use paths that are not necessarily optimal; this motivates investigation of very fast heuristic algorithms that obtain only near-optimal paths.

[2]If routes were pre-calculated, input file sizes would be significantly bigger.

Routing experiments were conducted with different variations of the standard Dijkstra's algorithm [7]. These algorithms were chosen due to recommendations made in Jacob *et al.* [12]. The algorithms studied were: (a) *Dijkstra*, where shortest-path trees, rooted at the source are constructed for each routing query, (b) *Optimized Dijkstra*, where the search loop is terminated upon finding the shortest path to the destination, (c) A^* search [11], a variant of Dijkstra that employs a heuristic cost-function to bias the direction of the search towards the destination. Further optimizations that were used in all these implementations include smart label reset (where only nodes explored in a previous routing computation are re-initialized) and use of efficient data structures.

A^* search was first proposed in AI literature [11], [18]. In road network graphs, A^* exploits the near-Euclidean property to expand the shortest path tree in the direction of the destination, whereas In Dijkstra, the search tree is expanded in a circular manner centered at the source node This results in the search arriving at the destination node much quicker for A^* than Dijkstra.

To bias the search towards the destination, we assign a cost to each intermediate vertex as follows: given source *s*, and destination *t*, for each intermediate 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*, and $D(v, t)$ is the estimated cost from *v* to *t* computed as a function of the Euclidean distance between *v* and *t*. Since we are interested in the shortest path in terms of time rather than distance, $l(s, v)$ is the time-cost of the path from s to v, and we define $D(v, t)$ as:

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

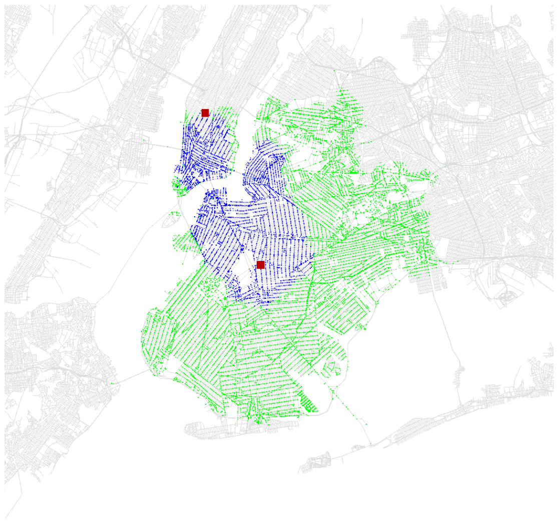
Fig. 4. Nodes expanded in A^* (shown in blue) vs. Dijkstra (shown in green) in a real road network. The red squares are the source and destination nodes, with the source being at the center. Dijkstra expands the search tree in all directions from the source node, while A^* is more directed.

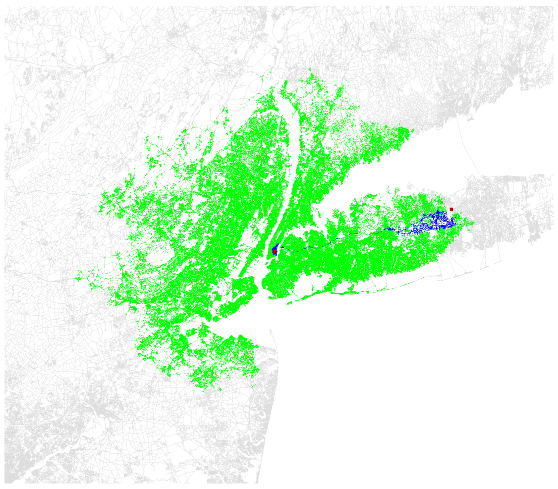
Fig. 5. A routing query where A^* performs markedly better, expanding only about one percent of the nodes (blue) compared to Dijkstra (green). On average, in our computations A^* expands 82% lesser nodes than Dijkstra.

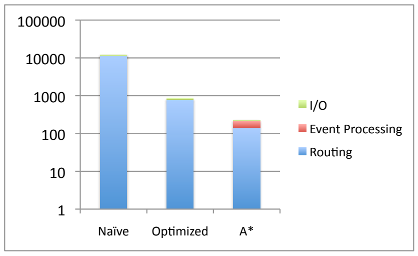

Fig. 6. Logarithmic chart showing various overheads for Dijkstra, optimized Dijkstra, and A^* in a serial simulation run.

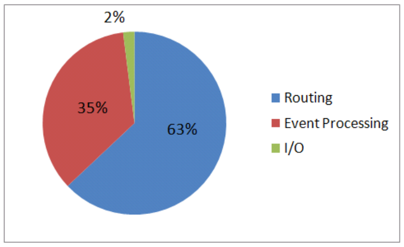

Fig. 7. Execution profile of FastTrans with A^* in a serial simulation on a 3 GHz Mac-Pro work-station.

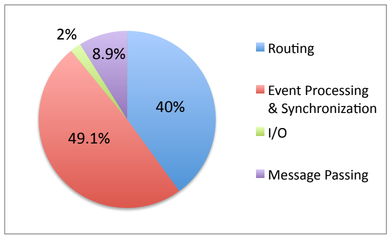

Fig. 8. Execution profile of FastTrans with A^* in a parallel simulation on a 32-CPU Linux infiniband I/O cluster.

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^* are not provably optimal; however, since the road graph is almost Euclidean, our experiments showed that the paths computed by A^* are on average only 0.02% longer than the paths computed by Dijkstra.

Figures 4 and 5 illustrate the performance of A^* versus an optimized version of Dijkstra on a real-world road network (from the northeast region of the United States). Notice that A^* explores a much smaller fraction of nodes than (even) the optimized Dijkstra. Figure 6 shows, on a logarithmic scale, the speed-up as well as the routing overhead for the three implementations (Dijkstra, optimized Dijkstra, and A^*) from a code-profiling exercise of a serial simulation run. The serial run was executed on a 3 GHz Mac-Pro work-station for $20,000$ itineraries that were randomly sampled from the study-set. Clearly, A^* gives much better performance-accuracy trade-off than other schemes. Figures 7 and 8 show the execution profiles for FastTrans with A^* in serial and parallel settings. In the serial setting (Figure 7), about 63 percent of the execution time is spent inside the routing module of FastTrans. In the parallel setting, where each simulation process keeps a copy of the entire routing graph, the fraction of time spent in routing decreases (Figure 8) due to the additional overhead from message passing and synchronization.

B. Partitioning and Load Balancing

In (standard) synchronized parallel discrete-event simulations, the difference between the simulation clock of any two logical processes cannot be greater than the look-ahead time value specified for the simulation[3]. Thus, a simulation process with a relatively high computational load, whose simulation clock progresses at a slower (wall-clock) rate than the other processes, slows down the entire system. Essentially, the speed of execution of the simulation is determined by the slowest process.

The computational load on a simulation process is determined by the partitioning scheme used to assign simulation objects to compute nodes. In this section, we explore different strategies for partitioning the simulation work load on a high-performance cluster. An ideal partitioning algorithm achieves

[3]The look-ahead time value specifies the minimum amount of simulated time it takes for messages to travel between two simulation processes.

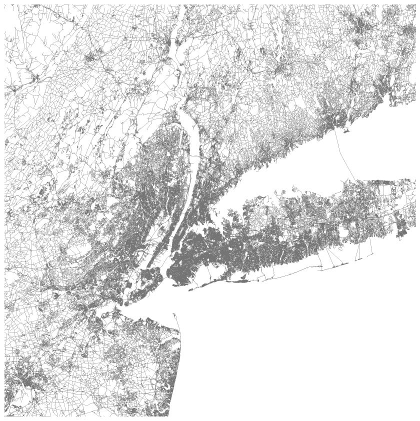
Fig. 9. Figure illustrating road-network density in the New York region.

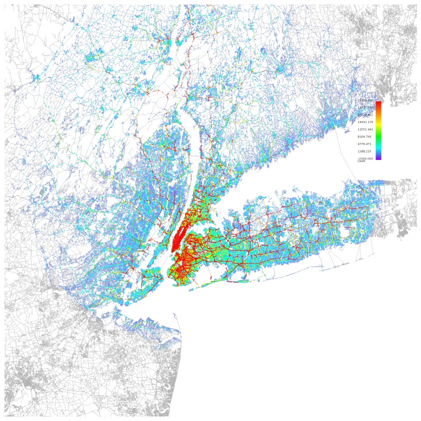
Fig. 10. Figure illustrating distribution of event load in the New York region.

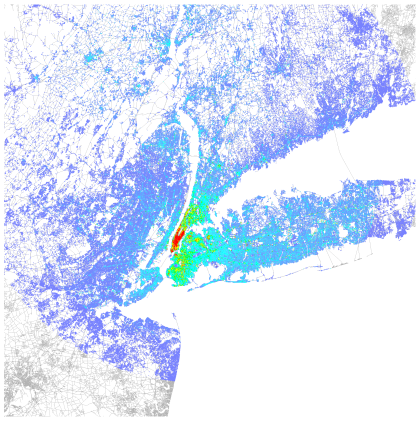
Fig. 11. Figure illustrating distribution of routing load in the New York region.

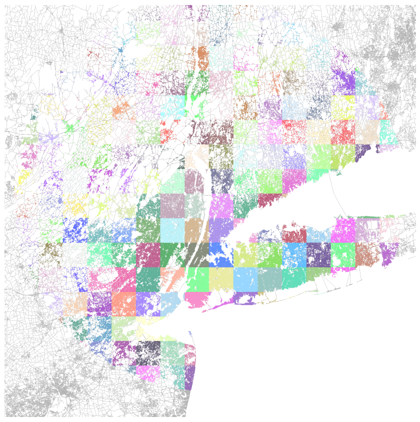
Fig. 12. Figure illustrating assignment of entities under geographic partitioning scheme. All entities with the same color are assigned to the same processor.

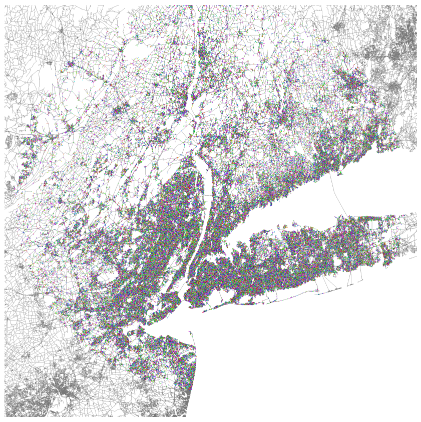
Fig. 13. Figure illustrating assignment of entities under scatter partitioning scheme. All entities with the same color are assigned to the same processor.

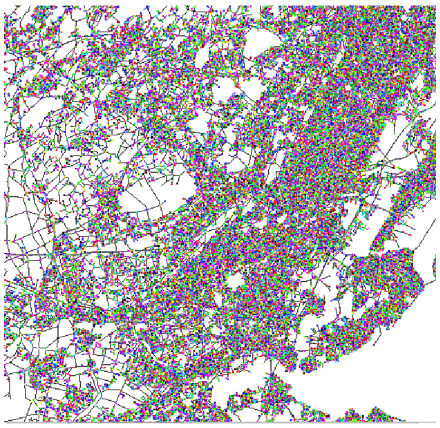
Fig. 14. A zoomed in version of the Figure 13. The scatter scheme assigns geographically close-by entities to different processors.

balanced load, while keeping messaging overhead to a minimum.

1) Geographic Partitioning: In FastTrans, the bulk of inter-process messaging occurs when vehicles travel between locations in the road network that have been assigned to different processors. Since vehicular trajectories are, by nature, spatially constrained, a partitioning algorithm that minimizes messaging overhead would assign road-network locations that are geographically close to the same processor. A simple geographic partitioning scheme divides the simulated geographic region into a uniform rectangular grid, and assigns all road-network entities (road intersections in our case) belonging to a grid cell to the same processor.

2) Geographic Partitioning with Balanced Entity Distribution: In real-world road networks, spatial distribution of network elements is far from uniform; node density is much higher within urban downtowns, and decreases significantly as one moves away from the core regions (Figure 9). Thus, a pure geographic partitioning scheme would result in unequal entity distribution. To overcome this, we use a non-uniform rectangular partitioning of the spatial region such that each grid cell now contains an approximately equal numbers of entities. Again, all entities in the same grid cell are assigned to the same processor.

3) Geographic Partitioning with Balanced Event Load Distribution: From our earlier profiling exercise (Figure 8), we observe that event-processing accounts for approximately 50% of execution time in parallel scenarios. Furthermore, the number of events generated by the simulation entities also varies with location. The event load distribution across the road network has a pronounced spatial characteristic – busy roads and intersections often tend to be geographically clustered. Figure 10 depicts a *a heat map* of the road network in the New York region, with areas in red being the busiest points in the network, and areas in blue being regions with low traffic volume. While it is generally not possible to accurately determine the event load associated with a given entity without actually running the entire simulation, we could assign an *event-weight* to an entity by simulating a small sample of the vehicular traffic. Once event-weights are assigned to all entities, we once again do a geographic partitioning assigning entities to processors while balancing event load distribution across processors.

4) Geographic Partitioning with Balanced Routing Load Distribution: This is similar to the previous scheme, with entityt-weight being the number of routing computations (rather than event computations) performed at an entity. Recall that routing also accounts for a significant fraction of execution time. Since, for a given trip, the entire route from source to destination is calculated at the starting point, locations that serve as trip originators (residential locations, business districts, etc) will generate more routing computations than transit locations with high traffic volume (busy freeways). Figure 11 illustrates the spatial distribution of the routing load in the road network.

5) Geographic Partitioning with Balanced Weighted-Sum of Loads: Event processing and routing both account for significant chunks of execution time; however, on a per-computation basis, a single routing computation can be up to two orders of magnitude slower than processing a single event. By assigning appropriate weights to each of these computational tasks, we can assign a weighted sum to an entity that approximately represents it's total computational cost. Then, while partitioning geographically, we balance this computational weight across processors. However, choosing the appropriate weights for routing and event processing is a complex task; routing calculations themselves can display enormous variations in running time depending on source and destination. Further total routing overhead itself can change with the size of the graph. Because of the complexities involved, we have not yet completely explored this scheme.

6) Scatter Partitioning: In contrast to the previous schemes, this partitioning scheme assigns entities that lie close to each other to *different* processors; that is, nearby entities are *scattered* across the cluster. In the road network data, successive nodes (intersections) are often numbered consecutively. If we define an entity-to-processor assignment as $P = m \bmod N$ (P is the processor to which entity m is assigned and N is the number of processors), then nearby entities will be assigned to different processors. This scheme is motivated by the spatial nature of load distribution in the network; entities that lie close to each other have similar load characteristics (Figures 10 and 11), and thus, by assigning nearby entities to different processors, we spread the computational load across the cluster. The obvious disadvantage to this scheme is the significant number of interprocess messages; now, for every node-hop that a vehicle makes, an interprocess message will be generated. However, as we shall see in the following section, this scheme performs surprisingly well.

V. Experimental Results

We carried out our experiments on two high performance clusters Coyote [1] and Lobo [2] available at the Los Alamos National Laboratory. Coyote, running 64-bit Fedora Core 3, has 1290 compute nodes, with each node consisting of two 64-bit AMD Opteron 2.6 GHz processors and 8GB memory. Lobo has a total of 272 compute nodes with each node containing 16 cores and 32 GB memory, for a total of 4352 cores. Both these clusters use a Voltaire Infiniband high-speed interconnect, and the Panasas parallel file system which provides a theoretical transfer rate of 20 GB/sec. For our scaling studies, we ran our experiments starting from a minimum of 32 processors to a maximum of 1024 processors. Interprocess communication was carried out using the OpenMPI message passing interface [14].

A. Partitioning and Load Balancing

We first present experimental results for the various partitioning schemes for FastTrans described in Section IV-B. The partitioning schemes were tested on the road network in the New York region, consisting of approximately half a million intersections, 1.1 million road links, and over 25 million vehicular trips. Together, these result in about four billion simulation events. All partitioning experiments described in this section were conducted on the Coyote cluster for different processor configurations, ranging from 32 to 512 processors.

Figure 15 illustrates the basic performance of the various partitioning schemes in terms of execution speed. Pure geographic partitioning performs the worst, while scatter partitioning is the fastest, outperforming geographic partitioning by about an order of magnitude. Interestingly, performance in terms of message-passing overhead is almost the reverse of execution time – the number of inter-process messages being passed in scatter partitioning (highest overhead) is an order of magnitude higher than geographic (lowest overhead). This simply re-iterates what we observed during the code profiling exercise (Figure 8), namely, that message-passing does not take up a significant chunk of execution time.

Execution times for partitioning based on routing-load and event-load are comparable to scatter, especially in the larger processor configurations (256 and 512). At these sizes, the geographical areas assigned to each grid cell become rapidly smaller, resulting in a more balanced load. Ultimately, the fairness of load distribution is the most important criterion for performance, as is made amply clear in Figures 17 through 21. These figures illustrate the computational load on each processor in the 256 processor set-up, with load being defined as a weighted[4] sum of event and routing load. Each bar represents the load on one CPU over the entire simulation. The load profile in the best-performing partitioning scheme (scatter) is more or less flat, while that in the poorly performing schemes are highly uneven. The Min/Max load ratio depicted in these figures is a useful metric for fairness comparison, since the speed of execution in a synchronized simulation is determined by the slowest process; it follows that a low Min/Max ratio will severely degrade performance, with ideal ratio being 1. Figure 22 compares this ratio for the various partitioning schemes; note how these correlate with speed of execution (Figure 15).

[4]For a fixed network and simulation scenario, we can determine, through profiling, the ratio of the relative cost of event processing to routing. The ratio in this scenario is about $1 : 94$.

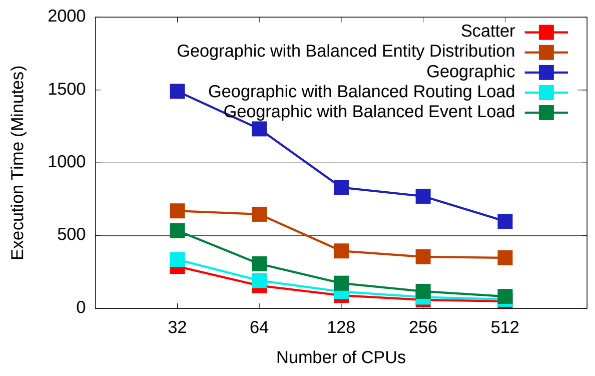

Fig. 15. Comparison of execution times of FastTrans (as a function of #CPUs) under different partitioning schemes.

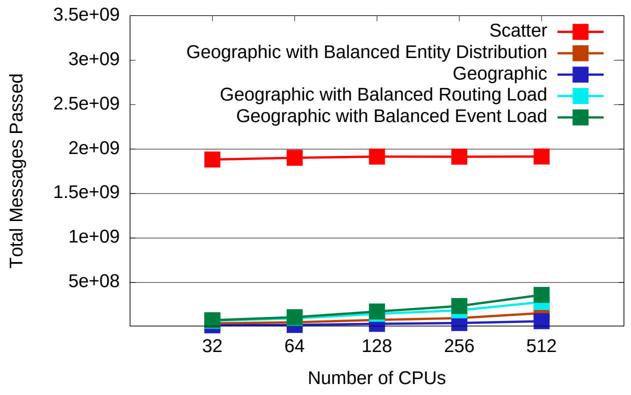

Fig. 16. Comparison of the number of messages passed in FastTrans (as a function of #CPUs) under different partitioning schemes.

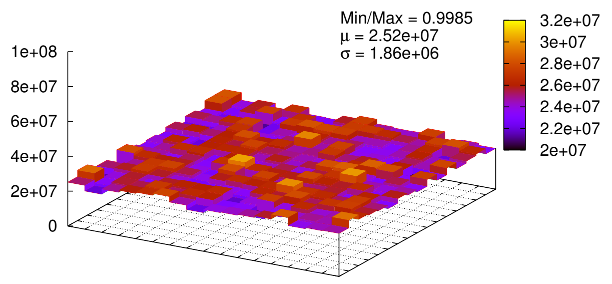

Fig. 17. Computational load distribution of FastTrans in a 256 CPU run under scatter partitioning. Each bar represents the load on one CPU.

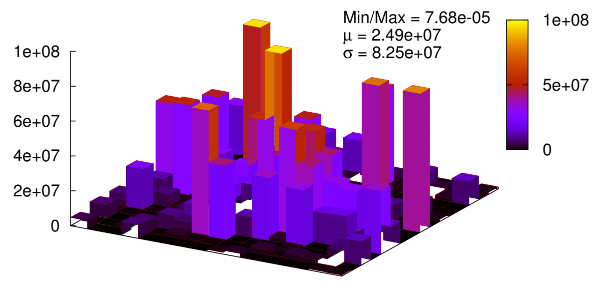

Fig. 18. Computational load distribution in FastTrans in a 256 CPU run under pure geographic partitioning.

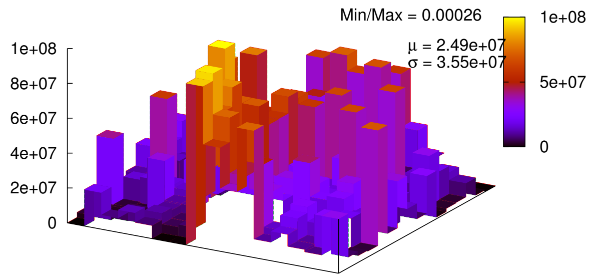

Fig. 19. Computational load distribution in FastTrans in a 256 CPU run under geographic partitioning with balanced entities.

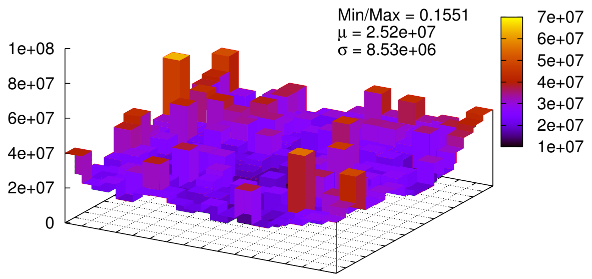

Fig. 20. Computational load distribution in FastTrans in a 256 CPU run under geographic partitioning with balanced event load (estimated).

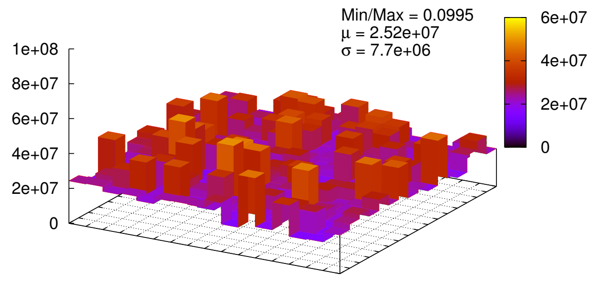

Fig. 21. Computational load distribution in FastTrans in a 256 CPU run under geographic partitioning with balanced routing load.

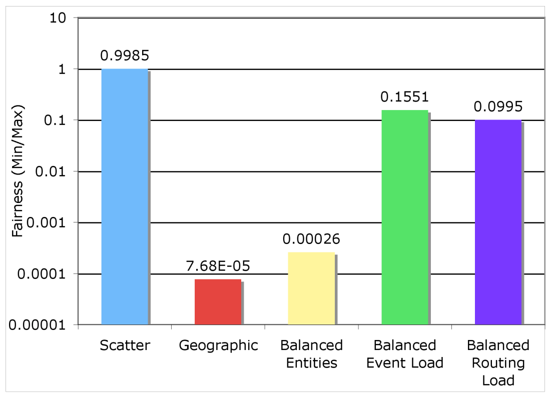

Fig. 22. Comparison of fairness of computation load in FastTrans under different partitioning schemes.

To conclude this section, we note that the geographic partitioning schemes based on routing-load and event-load perform comparably to scatter in terms of execution time, and also have very low messaging overheads. Thus, as noted earlier, for a specific scenario it may be possible to tune these schemes to outperform scatter partitioning. The tuning parameters, however, would be highly dependent on the specific scenario to be simulated; the event and routing load profile may differ significantly in a simulation of a disaster scenario from the simulation of a normal day. The load patterns would still exhibit spatial clustering (e.g., there are usually only a few main evacuation routes from a city) and scatter partitioning would continue to achieve a balanced load profile. Generally, for problem domains that exhibit spatial clustering of load, scatter partitioning is a simple and highly effective approach.

B. Computational Scaling Results

In the experiments in this section, we test the scaling performance of FastTrans, ActivitySim, and the integrated simulation on different processor configurations. For the FastTrans experiments, in addition to the New York region, we also simulated the smaller region of Twin Cities in Minnesota. This allows us to observe the scaling behavior of FastTrans as a function of the size of the road (routing) graph. The Twin Cities road network consists of approximately $300,000$ road links and $150,000$ intersections; the New York graph consists of half a million intersections and about 1.1 million road links. All the experiments in this section use the scatter partitioning scheme, and simulate an entire day's worth of vehicular trips – approximately six million for Twin Cities and 25 million for New York. All experiments described in this section were conducted on the Lobo cluster for different processor configurations, ranging from 32 to 1024 processors.

Figure 23 shows the scaling performance in terms of execution time for FastTrans for the two scenarios. As expected, the size of the New York road graph implies that routing calculations dominate the New York simulation and consequently, execution time falls much more rapidly (as we add more processors) for the New York scenario compared to the Twin Cities scenario. The performance levels off at about 512 CPUs for New York indicating that this may be close to the ideal number of processors for a scenario of comparable size. We expect further improvements with bigger cluster sizes (≥ 512) for larger graphs.

Memory usage per processor, depicted in Figure 24 is fairly constant for mid-size cluster runs (≤ 256), and increases for larger cluster sizes, even though one would expect decreasing memory burden per processor as more processors are used. The reason for this behavior is that memory usage on a compute node is dominated by the size of the routing data structures. Since each processor keeps a copy of the routing graph (as explained in Section IV-A), memory usage does not decrease. At larger cluster sizes, the size of the inter-process message buffers[5] causes the memory usage per node to increase, though still very much within the memory capacity of a compute node.

For both cities, the messaging overhead for FastTrans does not vary with the number of simulation processes. This behavior is entirely due to scatter partitioning (see also Figure 16); since nearby entities are assigned to different processors, an inter-process message is usually generated for each node (intersection) that a vehicle traverses. Thus, the number of inter-process messages in scatter partitioning essentially depends only on the number of trips and the average path-length (node traversals) – both of which depend only on the routing and modeling aspects of the simulation. Consequently, messaging overhead does not vary with cluster size.

Figures 26 through 28 show the scaling behavior performance of the integrated simulation in the Twin Cities scenario, compared to the performance of the modules when run separately. The performance of the integrated simulation follows the performance of the dominant module; thus, the execution time of the integrated simulation is similar to FastTrans, while memory usage resembles that of ActivitySim. Once again, as a result of scatter partitioning, messaging overhead is relatively constant for all three simulators irrespective of the cluster size. The absolute number of messages generated in the integrated simulation is less than that of FastTrans, since the number of vehicular trips that are created dynamically from within ActivitySim is less than the stand-alone version of FastTrans. All simulations for both cities run significantly faster than real-time even on 32 processors. The high realtime speedups allow us to simulate multiple scenarios and provide timely feedback and analysis in real-world situations.

VI. Conclusions and Future Work

In this paper, we presented the use of large-scale, parallel discrete-event simulation systems for realistic, microsimulation-based modeling of large urban areas. These included modules for realistic activity generation at the individual level, and a discrete-event queue-based parallel transportation simulator. The modular software architecture employed in building these systems allowed us to easily combine the two modules into a high-fidelity, activity-driven simulation of the road network.

Optimizations in the routing module through heuristics-based routing allow us to perform simulations with significant speed-ups over real time. Further, we discovered that the optimal way to partition the computational workload in our case was to exploit the spatial nature of the road network in a counter-intuitive way – namely to *scatter* the entities, that is, to explicitly assign nearby entities to *different* processors in the cluster. A possible direction for further investigation here is to observe if scatter partitioning or similar approaches perform as effectively in other types of networks that exhibit spatial clustering of load.

Experiments on HPC clusters illustrate the scalable nature of the simulation paradigms and software engineering principles employed in this paper. We are currently performing scaling studies on even larger areas of the continental US. Further,

[5]Each process maintains 2N communication buffers, where N is the number of processes in the simulation

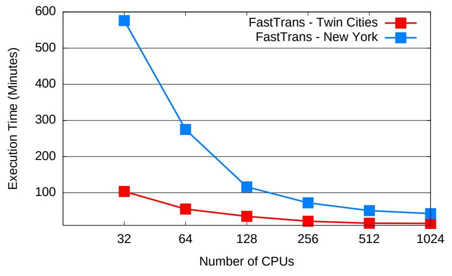

Fig. 23. Execution time of FastTrans as a function of #CPUs.

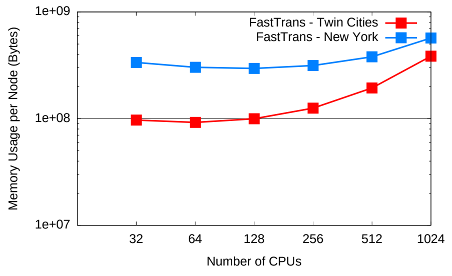

Fig. 24. Memory usage per node in FastTrans as a function of #CPUs.

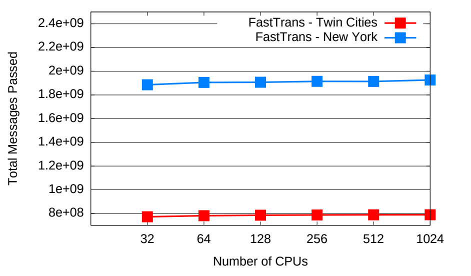

Fig. 25. Messages passed in FastTrans as a function of #CPUs.

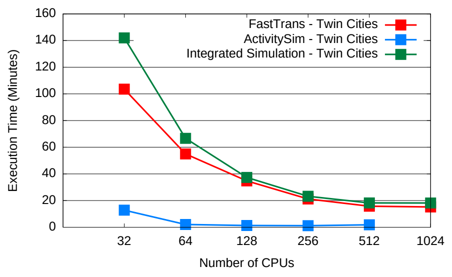

Fig. 26. Comparison of execution times of FastTrans, ActivitySim, and the integrated simulation as a function of #CPUs.

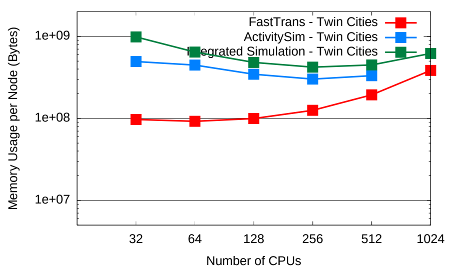

Fig. 27. Comparison of the memory usage per node in FastTrans, ActivitySim, and the integrated simulation as a function of #CPUs.

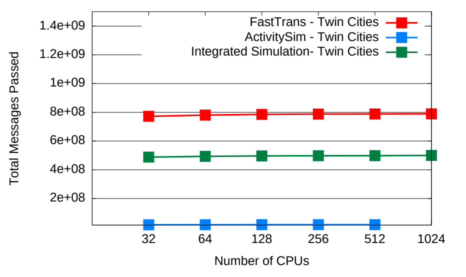

Fig. 28. Comparison of the number of messages passed in FastTrans, ActivitySim, and the integrated simulation as a function of #CPUs.

we are also exploring the use of hybrid architectures – computational platforms that use conventional and cell-based processors – in the area of discrete-event microsimulations.

Acknowledgements

This paper is a Los Alamos Unlimited Release LA-UR-09-03622.

References

[1] http://computing.lanl.gov/article/505.
[2] http://computing.lanl.gov/article/570.
[3] Cameron, G., and Duncan, G. Paramics: Parallel microscopic simulation of road traffic. In *Journal of SuperComputing* (1996), vol. 10, Springer, pp. 25–53.
[4] Cetin, N., Burri, A., and Nagel, K. Parallel queue model approach to traffic microsimulations. In *Proceedings of Swiss Transportation Research Conference* (2002).
[5] Charypar, D., Axhausen, K., and Nagel, K. An event-driven queue-based microsimulation of traffic flow. In *Transport Systems Planning and Transport Telematics* (2006).
[6] Concepts, I. T. VISSIM Simulation Tool. In *http://www.itc-world.com/VISSIMinfo.htm* (2001).
[7] Dijkstra, E. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959).
[8] Eissfeldt, N., Krajzewicz, D., Nagel, K., and Wagner, P. Simulating Traffic Flow With Queues.
[9] Fujimoto, R. M. Parallel discrete event simulation. *Commun. ACM 33*, 10 (1990), 30–53.
[10] Galli, E., Eidenbenz, S., Mniszewski, S., Teuscher, C., and Cuellar, L. Activitysim: Large-scale agent-based activity generation for infrastructure simulation. In *Proceedings of the 2009 Spring Simulation Conference* (2009).
[11] Hart, P., Nilsson, N., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.
[12] Jacob, R., Marathe, M., and Nagel, K. A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics 4*, 1999 (1999).
[13] MPI. *The MPI Message Passing Interface*. http://www.mcs.anl.gov/research/projects/mpi/.
[14] OpenMPI. *OpenMPI: Open Source High Performance Computing*. http://www.open-mpi.org.
[15] Perumalla, K. A systems approach to scalable transportation network modeling. In *2006 Winter Simulation Conference* (2006), pp. 1500–1507.
[16] Prevedouros, P., and Wang, Y. Simulation of large freeway and arterial network with CORSIM, INTEGRATION, and WATSim. *Transportation Research Record: Journal of the Transportation Research Board 1678*, 1 (1999), 197–207.
[17] PRIME. *Parallel Real-time Immersive network Modeling Environment*. Available at http://prime.mines.edu/.
[18] Sedgewick, R., and Vitter, J. Shortest paths in Euclidean graphs. *Algorithmica 1*, 1 (1986), 31–48.
[19] Smith, L., Beckman, R., Anson, D., Nagel, K., and Williams, M. E. Transims: Transportation analysis and simulation system. In *Proceedings of the Fifth National Conference on Transportation Planning Methods* (Seattle,Washington, 1995).
[20] Thulasidasan, S., and Eidenbenz, S. Accelerating traffic microsimulations: A parallel discrete-event queue-based approach for speed and scale. In *Proceedings of the 2009 Winter Simulation Conference* (2009).
[21] Transportation Research Board. *Highway Capacity Manual*, 2000.
[22] US Department of Transportation (DOT 2003). Bureau of Transporataion Statistics, NHTS 2001, Highlights Report, BTS03-05, 2003.