Mobiliti: Scalable Transportation Simulation Using High-Performance Parallel Computing

Cy Chan, Bin Wang, John Bachan, and Jane Macfarlane
{cychan,wangbin,jdbachan}@lbl.gov, janemacfarlane@berkeley.edu
Lawrence Berkeley National Laboratory, Berkeley, CA, USA
University of California, Berkeley, CA, USA

***Abstract*— Transportation systems are becoming increasingly complex with the evolution of emerging technologies, including deeper connectivity and automation, which will require more advanced control mechanisms for efficient operation (in terms of energy, mobility, and productivity). Stakeholders, including government agencies, industry, and local populations, all have an interest in efficient outcomes, yet there are few tools for developing a holistic understanding of urban dynamics. Simulating large-scale, high-fidelity transportation systems can help, but remains a challenging task, due to the computational demand of processing massive numbers of events and the non-linear interactions between system components and traveling agents. In this paper, we introduce Mobiliti, a proof-of-concept, scalable transportation system simulator that implements parallel discrete event simulation on high-performance computers. We instantiated millions of nodes, links, and agents to simulate the movement of the population through the San Francisco Bay Area road network and provide estimates of the associated congestion, energy usage, and productivity loss. Our preliminary results show excellent scalability on multiple compute nodes for statically-routed agents, simulating 9.5 million trip legs over a road network with 1.1 million nodes and 2.2 million links, processing 2.4 billion events in less than 30 seconds using 1,024 cores on NERSC's Cori computer.**

***Index Terms*— large-scale transportation simulation, agent-based modeling, high-performance computing, parallel discrete event simulation**

I. INTRODUCTION

Transportation is a highly complex system of systems that is a core contributer to our national economy. It consumes enormous resources in terms of both energy usage and productivity loss due to increasing urban congestion. The introduction of new technologies associated with autonomous vehicles, for-hire and ride-share networks, and multi-modal systems are adding new choices but their future impacts are difficult to estimate. With a desire to understand the system more thoroughly and build energy and mobility efficient controls, researchers in government and industry are interested in developing new methods to model very large scale transportation networks. While many current transportation system modeling tools can handle pieces of the urban environment, applying these tools to a holistic, large-scale urban system, such as entire metropolitan area requires more computational power than traditional computing solutions can provide. This leaves city planners and transportation engineers with only a partial understanding of the metropolitan area as a whole. Parallelization efforts to further increase the speed and size of simulations can provide new capabilities that will be necessary for understanding optimization opportunities.

Most existing parallel transportation simulators have the following two properties: shared memory parallelism and a global synchronization mechanism. First, shared memory parallelism refers to a configuration where threads executing in parallel share the same memory address space and can directly read data another thread has written in memory. The advantage of shared memory is that all simulation components reside in the same address space, so they all have immediate access to each other's data. They can either simply read each other's data or they can establish a shared queue where messages can be pushed by the producer and read by the consumer. The disadvantage of using only shared memory (without distributed memory support) is that the simulation is limited to running on a single compute node, typically up to 2 to 4 processors and their associated memory on a motherboard.

Second, global synchronization is commonly used to ensure parallel components do not get too far out of sync from one another. In a parallel discrete event simulation (PDES), worker threads can process events in parallel up to an agreed upon simulation time, after which they tell a master thread they are done and wait. When the master thread has received notification from all workers, it signals them to start the next time step. This global synchronization ensures messages sent during a time step will be visible to other workers in the next time step. However, it introduces a serialization point on the master, which can become a bottleneck as simulators scale to large parallel systems, and it requires that workers wait for all other workers to complete a time step before they can make further progress.

We have developed Mobiliti, a proof-of-concept simulator that avoids both of these restrictions by enabling distributed memory parallelism with an asynchronous execution strategy. Mobiliti is built on top of the Devastator parallel discrete event simulation runtime. Moving to distributed memory

This report and the work described were sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Big Data Solutions for Mobility Program, an initiative of the Energy Efficient Mobility Systems (EEMS) Program. The following DOE Office of Energy Efficiency and Renewable Energy (EERE) managers played important roles in establishing the project concept, advancing implementation, and providing ongoing guidance: David Anderson.

parallelism enables the use of multiple nodes in a high-performance computer or cluster, thereby increasing both the total computational power (more processors) and the total memory available (more memory slots). When threads on the same node need to communicate, we can still use the light-weight communication of shared memory parallelism, but when threads on different nodes communicate, they must send messages over the network interconnect (e.g. Ethernet, Infiniband, Cray Aries, etc.). The disadvantage of supporting distributed memory parallelism is an increase in complexity and overhead for handling these inter-node communications.

The asynchronous execution strategy allows us to remove the potential serial bottleneck of all threads synchronizing with a single global coordinator thread. Instead, worker threads coordinate directly with one another in a distributed fashion when events are transmitted between agents owned by different threads. This asynchronous execution allows more progress to be made in parallel even if some threads are slow on a particular time step since not all threads have to wait for the slowest one. The disadvantage is that an additional control mechanism is needed to preserve causality in the simulation. Either null messages can be used (as in the Chandy-Misra-Bryant protocol[1], [2]), or speculative execution and roll back [3], which can be expensive if too many events are speculatively executed. We will describe how we mitigate the costs of distributed parallelism and asynchronous execution in this paper.

The rest of the paper is organized as follows: Section II discusses related work, Section III gives an overview of parallel discrete event simulation, Section IV describes the Devastator PDES runtime, Section V introduces our agent-based transportation system model, Section VI presents some proof-of-concept experimental results, and Section VII describes the simulator's performance.

II. RELATED WORK

Foundational work in parallel discrete event simulation (PDES) include the development of conservative [1], [2], [4], [5], [6] and optimistic [3] simulation protocols. Since then, a large number of simulators have been developed to apply these techniques to various domains. For example, in the area of computer hardware design, the Structural Simulation Toolkit [7], [8] and the ns-3 simulator [9] use conservative parallel simulation to model computer and network architectures. ZSim [10] and Graphite [11] utilize a version of the conservative protocol where events are allowed to be reordered within a time interval, trading off accuracy for speed of simulation. On the optimistic side, the ROSS simulator implements Jefferson's Time Warp algorithm to provide a common interface for various models. They have demonstrated scalability of the optimistic approach by scaling their simulator to run on nearly 2 million cores simultaneously [12], [13]. A comparison between conservative and optimistic approaches is given in [14].

There are many existing serial and parallel transportation system simulators that model various components of the system. These include software such as MATSim [15], BEAM [16], and POLARIS [17]. Current parallel simulators typically use conservative parallelization to divide the simulation time into intervals and split events in each time interval across multiple worker threads. The threads then synchronize with each other at the end of each interval to ensure all events in the previous interval are completed before processing any event in the next interval. Using a sufficiently short time interval, this method can actually guarantee that no events are processed out of order (i.e. identical to a serial simulation); however, this restriction is often relaxed to allow faster execution, trading accuracy for simulation speed. There has also been some previous work on using optimistic parallel discrete event simulation for transportation systems. Notably, the SCATTER-OPT project has studied vehicle evacuation scenarios using micro-scale simulation [18], [19]. Our study differs from this previous work in that the scale of simulation we are targeting is larger (i.e. millions of nodes and links in an urban-scale road network), and we are currently using a meso-scale simulation view of the system.

III. PARALLEL DISCRETE EVENT SIMULATION

A. Agent-based models and discrete event simulation

The transportation system can be simulated by defining agents in the system that interact with one another by sending discrete events, each tagged with a simulation time (or *time stamp*) at which the event occurs. For example, an assembly line may be simulated by defining each station as an agent, and events correspond to the movement of parts between stations. If an event signaling the arrival of a part arrives at time t_0, and a station takes Δt seconds to process it, it may send an outgoing event at time $t_0 + \Delta t$ to the downstream station. If the workstation can only process one part at a time, the incoming parts may be queued and processed in order of arrival, so the agent responsible for simulating the workstation must then manage the queue of incoming parts to produce output events in the correct order and at the correct simulation time. If all stations in the assembly line are defined in such a way, and the simulator processes events in order of increasing time stamp, then the simulation of parts moving through the assembly line will be correct. This is the basic idea used in the Mobiliti simulator, but with road network links as agents and vehicles as events.

B. Conservative vs. optimistic parallelization

In a typical serial simulation, a single global event priority queue containing all of the events in the system is utilized so that events are processed in strictly increasing time stamp order. However, in a parallel simulation, events are being processed across multiple worker threads simultaneously [20], [21], so the simulator must ensure that events that influence each other (e.g. through modification of an agent's state) are executed in increasing order of simulation time. However, events that do not influence each other may execute in parallel without affecting the outcome of the simulation. There are two main ways this parallelization is achieved: conservative and optimistic [22].

Typically, the components participating in the simulation will be distributed across multiple worker threads (threads of execution that may run on separate processor cores). In a common method of conservative simulation (window-based), a global time step ΔT will be chosen such that at each step i in the simulation, the worker threads can independently simulate events that occur within a window of time $[T_i, T_{i+1})$, where $T_{i+1} = T_i + \Delta T$. At the end of each time step, all worker threads tell the master thread that they have completed execution, then the master increments the global time and broadcasts a message to workers to let them process events in the next time step $[T_{i+1}, T_{i+2})$. Thus, since all workers must wait for all others to finish, each time step is limited by the slowest worker thread. The window may be set to the minimal time delta between causally linked events, resulting in a simulation with no out-of-order event execution. However, several conservative simulators relax this constraint to allow faster performance at the cost of processing some events out-of-order (e.g. see [10], [11]).

In the optimistic approach first described by Jefferson [3], there is no need for global synchronization or null messages. Instead, each worker thread optimistically executes events assuming there will be no external causality violation. However, if an external event is received that should have been executed earlier in simulation time, the worker thread rolls back its local mis-speculatively executed events, and then replays them in the correct order. If any of the rolled back events sent events of their own, cancellation events must be sent, potentially causing further roll back. The main advantage of the optimistic approach is that no time is spent waiting in global barriers for all workers to reach the end of the time step, and there is no serial bottleneck in the master thread to coordinate progress at the end of each time step. The main disadvantage of this approach is that extra time is taken rolling back events, so care must be taken to ensure the number of mis-speculatively executed events does not overwhelm the advantage of parallelization. This behavior can be tuned by adjusting the window of time ahead of *global virtual time* (see Section IV) events are allowed to be speculatively executed. We designed Mobiliti to partition the road network across workers in a way so that the number of events crossing between partitions is reduced, thus reducing the number of events that cause roll back. Section VII will show that the number of rolled back events is relatively stable as we scale out to higher degrees of parallelism.

IV. Devastator Simulator Design and Interface

Devastator is the software runtime that implements the optimistic parallel discrete event simulator (PDES) on which Mobiliti functions. It is a general discrete event platform that provides an API dealing in events abstractly, independent of the scientific domain simulated (e.g. transportation). The runtime design was motivated by the following two goals: first, we believe our approach can achieve a very high level of performance on HPC systems; and second, we feel that our approach has the potential to both increase functionality and decrease the programmability burden compared to existing PDES system APIs. These points will be elaborated in a future publication; this section only aims to provide a brief introduction to the API design and implementation.

Devastator is written in C++14. Our programming techniques heavily leverage function closures (lambdas) and eschew traditional class hierarchies typical in object oriented programming. This fits well with our parallel communication strategy built upon fire-and-forget remote procedure calls since it promotes lambdas to be the main currency of parallel communication. The CPUs in Devastator are busily engaged in queuing lambdas to the mailboxes of others while executing the lambdas as they arrive in their own.

As with most PDES systems, our algorithms are built over message passing between CPUs. While MPI is the de-facto standard for distributed message passing in HPC, we find it to be deficient in expressing our algorithms with respect to both performance and productivity. Notably, we find it lacking in the ability to efficiently implement Active Messages (e.g. fire-and-forget remote procedure calls) which comprise the single mechanism by which all of Devastator communicates. For this reason we have chosen GASNet [23] as the communication layer for passing messages between distributed processes, since it exposes active messages as a first-order primitive that is highly tuned for the networking stacks predominant in HPC.

Additionally, for passing messages between CPUs in the same process, we have implemented a custom thread-to-thread active message layer that we believe to be state-of-the-art due to the absence of both locks and atomic read-modify-write instructions, yielding a strong performance boost for x86 processors. Composing the process-to-process messaging of GASNet along with our intra-process thread-to-thread messaging allows us to expose a global model of parallelism where the threads may send active messages to each other seamlessly across thread and process boundaries.

On top of the global thread-to-thread messaging paradigm we built a fairly standard optimistic PDES engine. Each CPU core manages a set of owned logical processes (LPs) (also known as *actors* or *agents*). Each LP manages a time-sorted queue of events which are merely state-change functions which execute against the LP to achieve three outcomes:

1) Change the LP's application state.
2) Generate future events for this or other LPs.
3) Return a function closure to *unexecute* the state change performed by this event. This is called if the system detects it executed the event out of proper time order.

Our method for computing global virtual time (GVT) is completely asynchronous and concurrent with the processing of events. At all times in the simulation, there is a GVT reduction in-flight that continuously lower-bounds the minimum time stamp seen across all LPs and in-flight messages in the system. GVT is necessary for knowing how far back the system may have to rollback, thus it is the only means for reclaiming the memory dedicated internally to event *unexecute* functions. Once an executed event's timestamp is surpassed by GVT, the event is *committed* (cannot be rolled back) since it would be impossible for an unseen event to

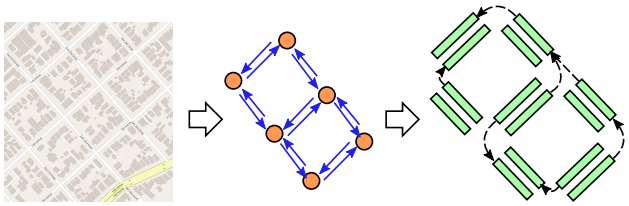


Fig. 1: Mobiliti System Model: OSM map (left) converted to road graph (nodes and links, center) converted to edge graph (right), where edges (green boxes) are the agents that send vehicle events (dashed arrows) to each other.

precede it, thus memory associated with the event and its rollback is reclaimed.

In summary, any application utilizing Devastator defines the behavior of its system agents by defining how events modify the agent's state and what (if any) future events are generated (via the *execute* function). Any modifications to the agent state must support being rolled back via the *unexecute* function, and the event supports being committed via the *commit* function.

V. TRANSPORTATION SYSTEM MODEL

A. Overall system design

In order to leverage the Devastator parallel discrete event simulator, we mapped the transportation system onto a set of agents and events. Our current implementation focuses on modeling the traffic flow and congestion dynamics on the road network. Figure 1 shows how a road network consisting of nodes and links is mapped to agents in the simulation. Each link is an agent in the simulation, and vehicles are events that are passed between the links. The link agent is responsible for determining how long it takes for each vehicle to pass through it, using information such as vehicle flow rate (currently implemented), downstream link blockage, and traffic signal timing (to be implemented). Using this information, the link agent can calculate the time each vehicle departs and send an event to the agent responsible for the next link on the vehicle's route with the correct arrival time stamp.

B. Link flow rate and congestion

Each link in the network has a freeflow speed and capacity associated with it. The freeflow speed is the speed at which a vehicle traverses the link given zero congestion on the link. The traversal time (under freeflow conditions) t_a is the link length divided by freeflow speed. The capacity c_a of the link is the number of vehicles per unit time the link has been designed to handle without significant congestion. In order to calculate the time $S_a(v_a)$ that a vehicle requires to traverse a link in congested conditions, we currently use the BPR formula [24]:

$$S_a(v_a) = t_a \left(1 + 0.15 \left(\frac{v_a}{c_a}\right)^4\right), \tag{1}$$

where v_a is the current vehicle flow rate. While the BPR formula enables very efficient estimation of congestion, it does not take into account spillback effects due to downstream link blockage. We are investigating the addition of a link storage capacity model to capture these spillback effects.

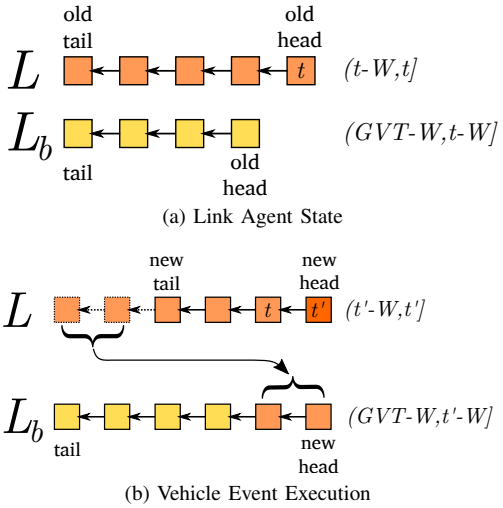


Fig. 2: Link agent state transition. Each link agent state consists of two linked lists of vehicle arrival times. When a vehicle arrival event is executed (at time t'), it is inserted onto the head of L and events older than $t' - W$ are migrated to L_b. The lists maintain the invariant that events are within the time ranges specified such that $|L|$ is the number of vehicles that arrived within time window W, and no event older than $GVT - W$ is kept in L_b.

In order to estimate the current vehicle flow rate v_a, the link agent keeps track of its most recent utilization. Our link implementation keeps track of the number of vehicles that have traversed the link in the last W seconds, where W is the link flow rate estimation window parameter. Given the number of vehicle arrivals in this window, we can estimate the current flow rate. To track this number, the link agent keeps track of a sorted linked list L of vehicles that have arrived within the window (see Figure 2). When a new vehicle arrives at simulation time t, it is pushed to the head of the linked list and all vehicles that arrived earlier than time $t - W$ are popped off the tail of the list. Thus, at any point in time, L contains exactly those vehicles that arrived within W seconds of the most recently arrived vehicle. The flow rate can then be estimated as the length of the linked list divided by the window size:

$$v_a = \frac{|L|}{W}, \tag{2}$$

C. Handling rollback

At some point in the simulation, a vehicle arrival event may need to be rolled back if another event subsequently arrives with an earlier simulation time stamp. The state of the link agent must then be reverted to before the vehicle arrival event was processed. Since the only state in the link agent is the linked list used to estimate vehicle flow rate, we only need to restore the state of the linked list L. There are many ways to do this, and in our current implementation we

chose to use a sorted "backup" list L_b that contains events that may need to be restored to L in the case of a roll back.

To support roll back, the original event execution is modified such that when a new vehicle arrives at time t, vehicles that arrived earlier than $t - W$ are moved from the tail of L to the head of L_b instead of dropped. The vehicle is also added to the head of L as before. If a vehicle arrival is rolled back, the vehicle is removed from the head of L, and vehicles are moved back from the head of L_b to the tail of L until L again contains all vehicle arrivals within W seconds of the (previous) head of L.

D. Handling commit

As described in Section IV, events can be committed when global virtual time (GVT) passes their timestamp. This means the event will never need to be rolled back, which implies that events on the backup list L_b with timestamps earlier than $GVT - W$ can be dropped from the backup list completely to return memory for other use. Since L_b is kept in sorted order, it suffices to pop elements off the tail of L_b until the tail event has a time after $GVT - W$. Since linked list insertions and deletions are cheap, event execution, rollback, and commit can all be processed very efficiently, as evidenced by our performance results.

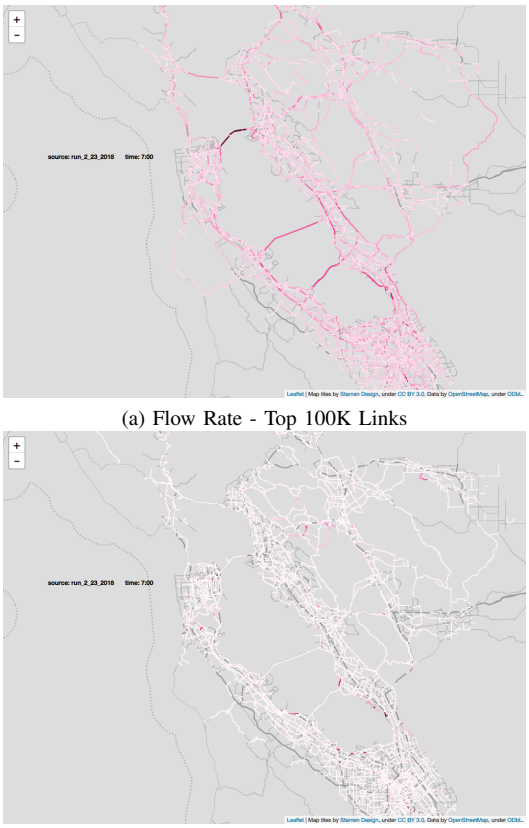
VI. Example Simulation Results

As our simulator is still in proof-of-concept phase, the results we show in this section are only meant to illustrate the capabilities of the simulator and give a rough sense for the performance of our approach. In a future publication, we will conduct further validation of the input and simulator and give more rigorous experimental results.

A. Example experimental setup

We are using a model of the San Francisco Bay Area exported from OpenStreetMap (OSM) [25]. It consists of 1,110,335 nodes and 2,185,984 links spanning from San Rafael, Concord, and Antioch to the north to San Jose to the south and Oakland, Hayward, and Fremont to the east (see Figure 3). For system demand, we used 463,938 agent plans generated using a modified Metropolitan Transportation Commission (MTC) Travel Model One [26]. Note that the term "agent" in this context refers to a simulated person utilizing the transportation network as opposed to the usage in "agent-based simulation". Each agent's plan consisted of multiple trip legs (origin/destination pairs), forming a total data set of 1,197,000 trip legs connecting 1,660,938 activities. Some trip legs in the data set were discarded since their origin and destination were the same (no movement). In order for the congestion model to generate more realistic congestion, we randomly synthesized more trip legs to increase the overall demand on the system.

Since the original trip leg data set containing 1,197,000 trip legs does not reflect the full volume of traffic in the Bay Area, we used it to synthesize random trip legs that represent more agents in the system. For each original trip leg, we can generate synthetic legs that are similar to it by randomly moving the origin and destination points and adding a random amount to the departure time (normal distribution, zero mean, one hour standard deviation). The origin and destination points are chosen by doing a random walk along the road graph for 400 steps. In the future, we may use information such as urban use data to generate more realistic origin and destination points instead of using a random walk. We tested running the simulator with eight times the number of original trips to see the impact of increased levels of congestion. The 8x replicated case corresponds to 3,711,504 agents and 9,548,128 trip legs.



(a) Flow Rate - Top 100K Links

(b) Delay Ratio - Top 100K Links

Fig. 3: Maps showing San Francisco Bay Area links with highest flow rates (a) and delay ratios (b)

B. Traffic flow rate and congestion delay

The simulator is capable of measuring various aspects of the transportation system throughout the simulation day. For each link in the network, we captured different metrics over every 15 minute time interval, including maximum flow rate observed and maximum delay ratio observed. Figure 3 illustrates a map of the San Francisco Bay Area with the links with the highest flow rate and delay ratios highlighted (deeper color means a higher value). Figure 4 shows the behavior of the top 1,000 links over the course of the simulation day, where the peak traffic flows during morning and afternoon rush hours can be clearly seen.

For each trip leg, we recorded the trip length and uncongested and congested durations. The uncongested durations were simply the sum of the free-speed traversal times over

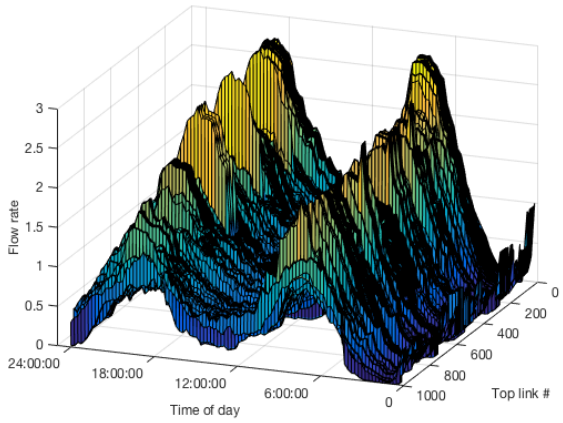


Fig. 4: Behavior over the course of the simulation delay for the top 1,000 links with the highest flow rates

the links in the path the vehicle takes. This allows us to compare the leg durations to compute the time the agent was delayed during that trip leg due to congestion. Figure 5 shows histograms of the uncongested versus congested delays along with a histogram of the delays experienced by all vehicles in the simulation (note the logarithmic Y-axis in these figures).

From the histograms in Figure 5, we observe that most trips in our example simulation are short and have minimal delay, but some legs are delayed a significant amount. In fact, while over 96 percent of trip legs are delayed by less than 10 minutes, almost 51,000 trip legs were delayed by over one hour, and almost 5,000 trip legs were delayed by over two hours due to congestion. The delayed legs are likely traversing the highly delayed links highlighted in Figure 3, which suggests an intervention through dynamic re-routing or dynamic signal timing could help alleviate the degree of congestion in this example scenario.

C. Fuel consumption model

Another example metric we collected is the amount of fuel consumed. Each vehicle can be assigned a specific powertrain model, leading to variable fuel consumption across vehicles. Instead of explicitly modeling the vehicle's aerodynamic drag, rolling resistance, and transmission efficiency, we model the vehicle's total resistance force as a function of velocity, based on the *coast down* testing approach [27]. Furthermore, the resistance force for each vehicle $F_r(v)$ can be approximated as a second-order function of velocity v:

$$F_r(v) = A + Bv + Cv^2. \tag{3}$$

For almost all current vehicles in the US market, the target coefficients can be found on the EPA website [28]. By conducting force analysis on a moving vehicle, the total traction force $F_t(v)$ can be expressed as:

$$F_t(v) = F_r(v) + mg\sin\alpha + ma \tag{4}$$

where m is the mass of the vehicle, g is the gravitational constant, α is the road inclination angle, and a is the instantaneous acceleration. For simplicity, we do not consider the instantaneous vehicle acceleration and the road inclination angle. To estimate each vehicle's fuel consumption rate, we developed data-driven vehicle energy models based on the dynamometer test datasets from Argonne National Laboratory [29], where the fuel consumption $c(v)$ can be modeled as a function of traction force and velocity:

$$c(v) = f(F_t(v), v) \tag{5}$$

We constructed fuel consumption maps to approximate this function for three typical vehicles: Ford Focus, Nissan Altima, and Ford F-150. Figure 6 shows the mapping constructed for the Ford Focus. Therefore, given the vehicle velocity, we are able to compute the traction force and fuel consumption rate using Equations 3 through 5.

Running our simulation with a 10% penetration of Ford Focuses, 10% Nissan Altimas, and 5% Ford F-150s, we collected the total daily fuel consumption with and without congestion, shown in the Fig. 7. We observe that the amount of extra fuel consumed due to congestion for the modeled vehicles (25% of the population) is 420,000 liters.

D. Productivity loss model due to congestion

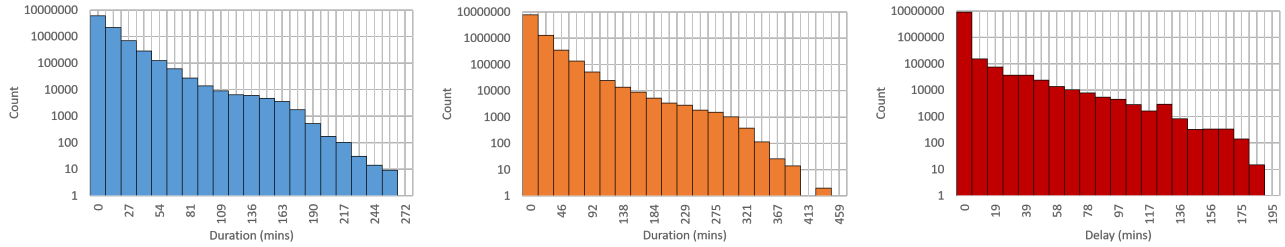
Following previous work conducted by the US Department of Transportation [30], we integrated productivity loss models due to congestion into Mobiliti. Specifically, the augmented trips in the simulation consist of five categories with assumed penetrations as shown in Figure 8. Based on the trip purpose, the calculation of the productivity loss is shown, where P_h is the local average hourly salary, P_{loaded} is the additional cost for business trips, P_{stck} is the stocking cost for trucks, η_{comm} is the cost ratio of commute trips, $\eta_{personal}$ is the cost ratio of personal trips, $P_{driver,}$ is the hourly cost for bus drivers, N_{pasg} is the estimated number of passengers in a bus.

The top 1,000 links with the highest productivity loss are shown in Figure 9. The simulation results indicate that the productivity loss on the top congested links reaches \$2,000 per 15 minute interval, and the total daily productivity loss is more than \$6 million. Specifically, the cost is \$2.17 million for business trips in passenger vehicles, \$0.67 million for business trips with medium and heavy trucks, \$1.69 million for daily commute trips, \$0.47 million for personal trips, and \$1.26 million for bus trips.

VII. Simulator performance

A. Routing and simulation

For these experiments, we pre-computed the routes for all of the trip legs using a modified A* search algorithm over the road network graph, where the A* heuristic is chosen to produce approximate paths rather than optimal for the sake of speed of evaluation. In future work, we will utilize a consistent, admissible heuristic so the paths returned are optimal. In either case, the routing is "embarassingly parallel", in the sense that each route can be computed independently of one another, making parallelization straightforward. Each route has 232 link traversals on average, and the longest trip legs have over 5,000 link traversals.



(a) Trip Leg Duration - Uncongested (b) Trip Leg Duration - Congested (c) Trip Leg Delay

Fig. 5: Histograms showing uncongested (a) versus congested (b) trip duration and the delay (c) experienced by each leg due to congestion. Note the logarithmic Y-axis scale.

Fig. 6: Representative fuel rate map of 2012 Ford Focus

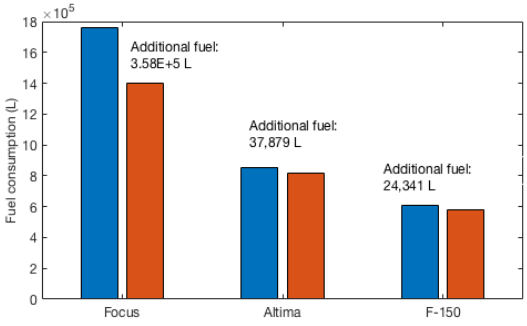


Fig. 7: Total daily fuel consumption with and without congestion

Trip Type	Price $P_{n,h}$ ($/h)	Rate Calculation	Penetration
Business-passenger vehicle	20	$P_{n,h} = P_h + P_{loaded}$, i.e. (20=15+5)	10%
Business-Medium&Heavy Truck	35	$P_{n,h} = P_h + P_{stck}$, i.e.(35=15+20)	5%
Commute-passenger vehicle	7.5	$P_{n,h} = P_h \cdot \eta_{comm}$, i.e.(7.5=15*50%)	60%
Personal	4.5	$P_{n,h} = P_h \cdot \eta_{persl}$, i.e. (4.5=15*30%)	20%
Bus	80	$P_{n,h} = P_{driver,h} + P_h \cdot \bar{N}_{pasg} = (P_h + P_{loaded}) + P_h \cdot \eta_{comm} \cdot \bar{N}_{pasg}$, i.e.((15+5)+8*50%*15)	5%

Fig. 8: Productivity loss computation for each agent type

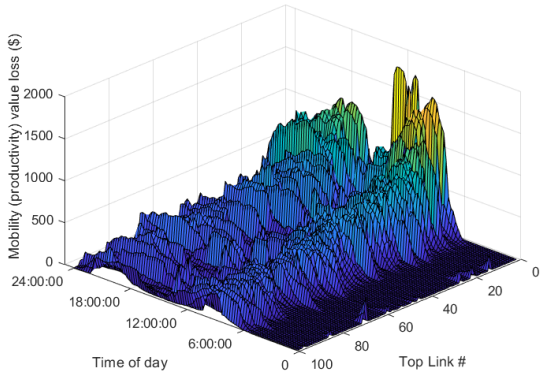


Fig. 9: Productivity loss for top 1,000 links

B. *Simulation scalability*

The simulation itself involves running the optimistic parallel discrete event simulation described in Section IV. To collect performance results, we ran on up to 1,024 cores on the Cori supercomputer at NERSC [31] using the base simulator configuration, which simulates the traversal of all vehicles through the system using the link congestion model (without energy and productivity loss calculation). The simulation runs with 8x trip leg replication (9,548,128 trip legs total) producing 2,382,465,156 committed events (vehicle-link traversals) system-wide. There are also some events that are mis-speculatively executed and rolled back, so the total number of events *executed* (committed or rolled back) is actually higher than 2.4 billion. Figure 10 shows the total number of events executed as the number of cores used is varied from 64 to 1,024. The total fluctuates from run to run due to the non-deterministic nature of the parallel algorithm, but we observed roll back to remain under 50 percent of the total, even at higher levels of parallelism.

As a result of the relatively flat aggregate event execution count as the number of cores is increased, the performance of the simulator scales very well. Figure 11 shows the time required to execute the simulation as the number of compute cores is varied from 64 to 1,024 (this corresponds to 1 to 16 compute nodes). We estimate this simulation would take about six hours to run serially on a single core, while our simulator can complete the simulation in well under a minute using 1,024 compute cores distributed across 16 nodes.

It is important to remark that the addition of more features or complexity to the model will likely impact performance

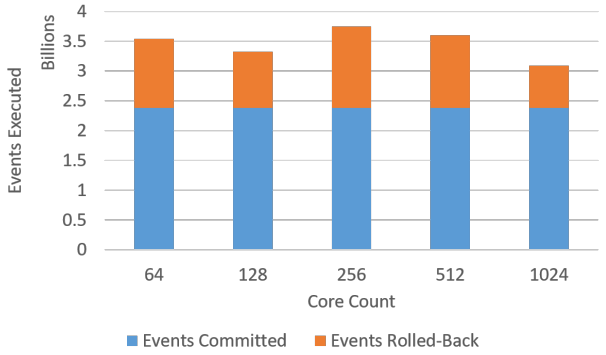


Fig. 10: Number of events committed and rolled back as number of cores is varied. The number of committed events is constant and the number of total events executed is relatively flat.

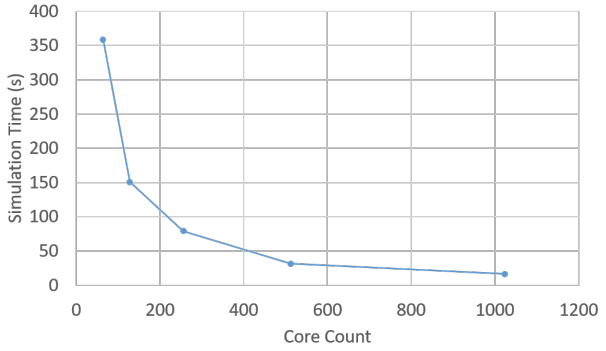


Fig. 11: Simulation execution time as number of cores is varied. Note this does not include route calculation time. Scalability is very good and achieves near linear speed-up.

– this is true of any computational model. Furthermore, changing the dynamics of how the components interact will impact how much rollback will occur in the simulator. For example, we are currently investigating the addition of a link storage capacity model to capture spillback effects [32], which will necessitate the addition of events propagating upstream as congestion occurs. In a future publication, we will discuss how this feature increases the accuracy of the model and analyze the impact on computational efficiency.

VIII. CONCLUSION

This paper describes our approach to large scale simulation of transportation systems using parallel discrete event simulation on high performance computing platforms. We have introduced the Mobiliti simulator to show proof-of-concept simulations of 9.5 million trip legs over a detailed San Francisco Bay Area road network. Our example analysis shows how the simulator may be used to estimate congestion, energy use, and productivity loss. In a future publication, we will present a more detailed analysis of our simulator including model validation and performance results. The capability to run simulations that process billions of events within minutes or seconds will enable mobility researchers in government and industry to better understand and predict future behavior of large-scale transportation systems.

REFERENCES

[1] K. M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Trans. Softw. Eng.*, vol. 5, no. 5, pp. 440–452, Sep. 1979.

[2] R. E. Bryant, "Simulation of packet communication architecture computer systems," Cambridge, MA, USA, Tech. Rep., 1977.

[3] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[4] B. Lubachevsky, A. Schwartz, and A. Weiss, "An analysis of rollback-based simulation," *ACM Trans. Model. Comput. Simul.*, 1991.

[5] P. M. Dickens *et al.*, "Analysis of bounded time warp and comparison with yawns," *ACM Trans. Model. Comput. Simul.*, 1996.

[6] D. M. Nicol and J. Liu, "Composite synchronization in parallel discrete-event simulation," *IEEE Trans. Parallel Distrib. Syst.*, 2002.

[7] A. Rodrigues *et al.*, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, March 2011.

[8] C. L. Janssen *et al.*, "A simulator for large-scale parallel architectures," *International Journal of Parallel and Distributed Systems*, 2010.

[9] J. Pelkey and G. Riley, "Distributed simulation with mpi in ns-3," in *Conference on Simulation Tools and Techniques*, 2011, pp. 410–414.

[10] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *International Symposium on Computer Architecture*, 2013.

[11] J. E. Miller *et al.*, "Graphite: A distributed parallel simulator for multicores," in *16th International Conference on High-Performance Computer Architecture*, 2010, pp. 1–12.

[12] D. W. Bauer Jr., C. D. Carothers, and A. Holder, "Scalable time warp on blue gene supercomputers," in *Workshop on Principles of Advanced and Distributed Simulation*, 2009, pp. 35–44.

[13] P. D. Barnes, Jr., C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp speed: Executing time warp on 1,966,080 cores," in *Conference on Principles of Advanced Discrete Simulation*, 2013, pp. 327–336.

[14] C. D. Carothers and K. S. Perumalla, "On deciding between conservative and optimistic approaches on massively parallel platforms," in *Winter Simulation Conference*, 2010, pp. 678–687.

[15] "MATSim.org." [Online]. Available: https://www.matsim.org/

[16] C. Sheppard *et al.*, "Modeling plug-in electric vehicle charging demand with beam, the framework for behavior energy autonomy mobility," Tech. Rep., 05/2017 2017.

[17] "Polaris." [Online]. Available: https://polaris.es.anl.gov/

[18] K. Perumalla, "A systems approach to scalable transportation network modeling," in *Winter Simulation Conference*, 2006.

[19] S. B. Yoginath and K. S. Perumalla, "Reversible discrete event formulation and optimistic parallel execution of vehicular traffic models," *International Journal of Simulation and Process Modelling*, 2009.

[20] L. F. Pollacia, "A survey of discrete event simulation and state-of-the-art discrete event languages," *SIGSIM Simul. Dig.*, Sep. 1989.

[21] F. J. Kaudel, "A literature survey on distributed discrete event simulation," *SIGSIM Simul. Dig.*, vol. 18, no. 2, pp. 11–21, Jun. 1987.

[22] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[23] "Gasnet." [Online]. Available: http://gasnet.lbl.gov

[24] Bureau of Public Roads: Transportation Research Board, "Special report 209: Highway capacity manual," Washington, DC, 1985.

[25] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org , 2017.

[26] G. Erhardt *et al.*, "MTCs travel model one: Applications of an activity-based model in its first year," in *5th Transportation Research Board Innovations in Travel Modeling Conference*, January 2012.

[27] I. Preda, D. Covaciu, and G. Ciolan, "Coast down test theoretical and experimental approach," Oct. 2010.

[28] US EPA, OAR, "Data on cars used for testing fuel economy," 2016.

[29] Argonne National Laboratory, "Downloadable dynamometer database." [Online]. Available: https://www.anl.gov/energy-systems/group/downloadable-dynamometer-database

[30] "Assessing the Full Costs of Congestion on Surface Transportation Systems and Reducing them through Pricing," Feb. 2009.

[31] NERSC, "Cori Configuration," http://www.nersc.gov/users/computational-systems/cori/configuration/, 2018, [Online; accessed 27-Apr-2018].

[32] V. Knoop *et al.*, "The influence of spillback modelling when assessing consequences of blockings in a road network," 2008.