Reversible discrete event formulation and optimistic parallel execution of vehicular traffic models

Srikanth B. Yoginath and Kalyan S. Perumalla*

Oak Ridge National Laboratory,
P.O. Box 2008, MS-6085, Oak Ridge, TN 37831, USA
E-mail: perumallaks@ornl.gov
E-mail: yoginathsb@ornl.gov
*Corresponding author

Abstract: Vehicular traffic simulations are useful in applications such as emergency planning and traffic management, for rapid response and resilience. Here, a parallel traffic simulation approach is presented that reduces the time for simulating emergency vehicular traffic scenarios. We use a reverse computation-based optimistic execution approach to parallel execution of microscopic, vehicular-level models of traffic. The unique aspects of this effort are

- exploration of optimistic simulation of vehicular traffic
- addressing the related reverse computation challenges
- achieving absolute, as opposed to self-relative, speedup.

The design, development and performance study of the parallel simulation system is presented, demonstrating excellent sequential and parallel performance. A speed up of nearly 20 on 32 processors is observed on a vehicular network of 65,000 intersections and 13 million vehicles.

Keywords: parallel simulation; discrete event; reverse computation; vehicular simulation.

Reference to this paper should be made as follows: Yoginath, S.B. and Perumalla, K.S. (2009) 'Reversible discrete event formulation and optimistic parallel execution of vehicular traffic models', *Int. J. Simulation and Process Modelling*, Vol.

Biographical notes: Srikanth B. Yoginath is a research staff member in the Computational Sciences and Engineering Division at ORNL. His expertise is in the Design and Development of Parallel Computing Systems. He has previously developed Parallel Data-Analysis Software Systems and has performed research in Grid Computing and Discrete-Event Computer Network Simulation Models. He earned his Master's Degree in Computer Science from Illinois Institute of Technology, Chicago, and is currently pursuing his PhD in Computational Sciences and Engineering, from Georgia Institute of Technology, Atlanta.

Kalyan S. Perumalla is a senior research staff member in the Computational Sciences Division at the Oak Ridge National Laboratory (ORNL), and holds an Adjunct Professor appointment at the Georgia Institute of Technology. His areas of interest include parallel and distributed systems, modelling and simulation, and parallel combinatorial optimisation. He has co-authored a book and has published over 70 papers in peer-reviewed conferences and journals. Four of his co-authored papers received the best paper awards, in 1999, 2002, 2005 and 2008. Several of his research prototype tools have been disseminated to research institutions worldwide. He earned his PhD in Computer Science from Georgia Tech (1999).

1 Introduction

In applications such as emergency or evacuation planning (Franzese and Han, 2001; Perumalla and Bhaduri, 2006), a higher speed of simulation for traffic models translates into faster determination of critical metrics such as expected evacuation time. While sequential simulators exist for traffic simulation, scalable parallel simulations are few. Among the existing parallel traffic simulators, parallelism is realised either by functional parallelism (Fisher, 2000) (parallelising the steps such as trip planning, configuration generation, partitioning, etc.), or by synchronous parallel execution (time-stepped models) (Laboratory, 2001; Meister et al., 2006; Innovative Transportation Concepts, 2001; Cameron and Duncan, 1996), or both. Our focus is to combine the speed of discrete event models with parallel execution of the actual simulation runtime. Additionally, despite speed concerns, our efforts are in staying at higher fidelity with entity-level models, as opposed to resorting to aggregate techniques such as fluid or network flow models. The higher fidelity vehicular-level models enable the simulation of more complex flow patterns and accommodating potentially rich routing and behavioural mechanisms.

With advances in parallel discrete event simulation modelling and the availability of multi-processor hardware, high-speed simulation of high fidelity vehicular traffic models is beginning to become possible now. Based on this premise, a parallel vehicular traffic simulation model called SCATTER-OPT has been developed, standing for an optimistic-parallel version of the SCATTER simulation system. This simulator is capable of using either conservative or optimistic synchronisation when executed on parallel platforms. Importantly, the parallel execution speedup achieved by this simulator is over and above some of the best performance achievable sequentially by vehicular traffic simulators today. The performance gains reported here, in that vein, are absolute and not simply self-relative. While self-relative speedups have been reported in the vehicular simulation literature, absolute speedup is relatively much more challenging to achieve, in view of the highly optimised execution possible in sequential simulation without the overheads arising out of model partitioning and parallel synchronisation issues.

Optimistic parallel simulation and reverse computation-based rollback are not new. However, they have not been explored for vehicular traffic simulation before. Also, purely discrete event execution has not been applied to vehicular traffic. Reversible formulation of vehicular traffic operation is another contribution of our effort. Our use of discrete event techniques from scratch helps us achieve the high sequential speed that is competitive with aggregate models in optimised sequential vehicular traffic simulators. Optimistic execution of the reversible formulation enables us to scale to much larger networks with good speedup on platforms with 64 processors.

In order to perform a controlled evaluation of the performance gains, both sequential and parallel, we define and use a benchmark grid-like vehicular network that is customisable to realise a wide range of network scenarios. The runtime performance is measured on one processor against the performance of an existing state-of-the-art simulator. Parallel performance is evaluated on larger network sizes. Among the challenges in applying reverse-computation and optimistic execution to traffic simulation are:

- developing an effective domain partitioning method
- dealing with road congestion effects that need to be modelled accurately in order to obtain correct travel time results.

These aspects distinguish our effort from earlier traffic simulations such as Carothers et al. (1999), Garrett et al. (2003), Tang et al. (2006), Cameron and Duncan (1996), Fellendorf et al. (1996), Franzese and Han (2001), Gartner and Stamatiadis (1998) and Yang et al. (2000).

In the rest of the paper, the design, development and performance of SCATTER-OPT is described. This is started in Section 2 with a presentation of the design overview of the transportation model, and a documentation of some of the abstractions of the road network system. This is followed by a discussion on our parallelisation approach, with discussion of the methods for realising vehicular traffic system characteristics such as congestion and routing. In Section 3, the reverse computing algorithms are discussed that are used for realising optimistic synchronisation. Section 4 is dedicated to a performance study of sequential execution, in which sequential simulation runtime performance is evaluated against that of a de facto standard emergency management system called the Oak Ridge Emergency Management System (OREMS) (Bhaduri et al., 2006; Franzese and Han, 2001). This is followed by a parallel performance study of SCATTER-OPT in Section 5, evaluating both optimistic and conservative synchronisation performance for parallel execution. Section 6 summarises our current work and outlinesfuture directions.

2 SCATTER-OPT system

Our simulation model is implemented in a simulator called SCATTER-OPT, which includes a discrete event model, and a parallel execution framework for vehicular networks. Parallel discrete event simulation of vehicular network operation involves the development of a discrete-event model of the system coupled with a careful partitioning of the model on to multiple processors for optimum run-time performance. In our simulation model, the road network is modelled as a graph, in which road segments connect intersections. Each road segment is modelled with a few physical attributes (number of lanes and length of road segment), kinetic specifications (speed limit) and controllers (traffic lights). The traffic lights are synchronised in their operation and have a fixed period of GREEN time (when the vehicles are allowed to enter the road-segment) and a fixed period of RED time (when the vehicles are not allowed to enter). Traffic lights control the entry of vehicles for every road segment of the simulated road network.

An input file is used to specify parameters such as the GREEN time period and the RED time period, along with an initial-offset (i.e., the wait time to the first GREEN period at simulation start time). Each road-segment in the simulated transportation network contains this information.

An intersection, a point at which the road segments connect to each other, is considered as an indivisible, independent processing unit for parallel computing purposes. Every intersection is capable of generating new vehicle instances that are distinguished from each other by their unique identifiers. Each generated vehicle has its physical attributes (e.g., length of vehicle) and kinetic attributes (e.g., travel velocity and acceleration), in addition to its source and destination intersection information for its current trip. Every intersection in the road-network is capable of routing vehicles to their next hop toward their individual destinations.

A *VehicleEvent* (*VE*) signifies the arrival of a vehicle from one intersection to another. Events of this type are processed by each intersection to generate additional *VEs* that act as arrival events on this intersection's

neighbouring intersections. The dynamically chosen neighbouring intersection is the next hop node toward the destination of the vehicle contained in a *VE*.

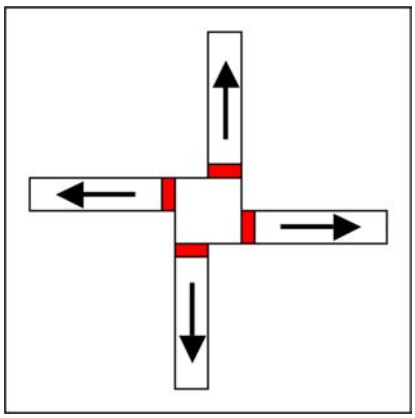
Information is provided in an input file about the road network layout (that encompasses intersection co-ordinates, connecting road-segment characteristics), traffic light information, source and destination intersections, traffic generation rate etc. This information is used to setup and initialise the simulation process.

2.1 Parallel decomposition

A challenge in any parallel discrete event model involves the issue of how to split the discrete event road-network model among the parallel processors. An efficient partitioning of the application process across different processors during parallel computation fetches better performance. Efficient partitioning involves recognition of modules of an application process that could run concurrently, with minimal, infrequent interaction between each other. In particular, zero lookahead interaction is to be avoided for parallel simulation efficiency.

We make an intersection (node) in the input road-network graph as a logical process. This would require grouping the road-segments connecting to each other via an intersection. There is a choice on how the segments are mapped relative to their intersection(s). Either incoming road-segments or out-going road-segments can be included as part of an intersection logical process. The former approach, namely, incoming segments mapped to their destination intersections, was used in the original SCATTER system (Perumalla, 2006). We use the alternative approach in SCATTER-OPT, namely, outgoing segments mapped to same logical process as their source intersection, as this somewhat simplifies the data structures for reversibility. Thus, in SCATTER-OPT we consider the node of the road-network graph along with its outgoing road-segments as a logical process, as shown in Figure 1. Each intersection in the transportation model is a collection of outgoing segments from that intersection plus the actual intersection space itself.

Figure 1 Intersection comprising four out-going road-segments with traffic lights (see online version for colours)



2.2 System implementation

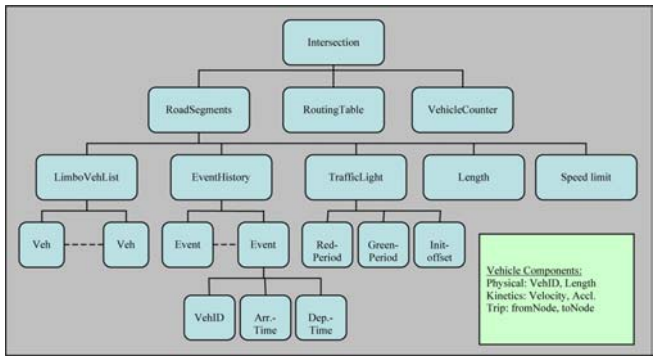
2.2.1 Simulation engine

We build our simulator on top of the publicly available μsik simulation library (Perumalla, 2004, 2005) to realise both conservative and reverse-computing based optimistic parallel discrete event simulation model. The μsik library is a general-purpose parallel/distributed simulation kernel, which provides programming interfaces to develop models that could run on one or more machines. Each processor can host multiple simulation logical processes. Logical processes are autonomous in the sense they hold and manage their own events and can be optimistic or conservative in their event processing. Efficient communication and virtual time synchronisation are also provided across processors in shared and/or distributed memory platforms.

2.2.2 Intersection data-structure

The organisation of data structures within each road intersection is shown in Figure 2.

Figure 2 Intersection data-structure (see online version for colours)



Each intersection contains one or more road segments, a routing table and vehicle counts and other statistics. The major part of the intersection is the list of its outgoing road segments. Each road segment contains an event history, a 'limbo' list, a traffic light specification and other details. Each limbo list element contains that state of a vehicle that has reached this intersection but yet to be routed or accommodated into their exit time lines. Event history is a list of events of departed vehicles in increasing time order.

2.2.3 Traffic generation

The SCATTER-OPT input file contains the information of the rate at which vehicles travelling to a particular destination to be generated. Note that a rate, rather than individual vehicle identity, is specified in the file. This is for convenience only; each vehicle is individually represented and simulated autonomously (e.g., a single vehicle could get stalled at a traffic light), and no aggregation is performed. Note also that the rate is on a per

flow-basis, the rate can be different across different flows. Any desired traffic pattern can thus be generated in general.

A *CreateEvent* (*CE*) type is used for the purpose of creating a vehicle at each intersection based on the rate specified in the input file. A*CE*, when processed (as shown in Figure 3), creates a *VE* that is rescheduled on the same intersection, and also creates another *CE* that is scheduled to a random time increment d*t* in future, to continue chaining generation of vehicles at specified input rate.

Figure 3 *CreateEvent* processing block

CreateEvent:
- Create VE at this intersection
- Schedule it to the same intersection at (t + lookahead)
- Create and schedule CE at (t +dt) to this intersection

2.2.4 Routing

After generation of a vehicle, each intersection ensures that a 'vehicle-following' scheme is correctly followed for every vehicle. A vehicle-following scheme ensures vehicles follow First-In-First-Out (FIFO) scheme while following their individual speeds and accelerations to the farthest possible distance. Routing of each vehicle toward its destination is based on Dijkstra's shortest path algorithm. The Boost's Graph Library (BGL) is used for the purpose of computing shortest paths. The road-network of the input file is converted into a BGL graph. For every node the shortest path to every other node in the network is determined, from which a vector of parents for each node is obtained. Each intersection uses this vector recursively to determine the next hop intersection of the vehicle toward its destination. Based on the next hop, the intersection determines out-going road-segment on which the incoming vehicle is routed.

Whenever a vehicle enters the intersection, the departure time (t_d) is calculated using transit time (t_t) and the vehicle's wait time at the traffic signal (t_w): $t_d = t_t + t_w$.

Transit time (t_t)

Transit time is calculated by solving the quadratic equation from the Newton's law of motion for time *t*. It takes the velocity and acceleration of the given vehicle into consideration, $S = ut + (1/2at^2)$, where, *S* is the distance travelled (road-segment length), *u* is the velocity of the vehicle; *a* is the acceleration of the vehicle, and *t* is the time to traverse the road-segment length. This approximation assumes that acceleration remains fixed during its travel, which is a reasonable assumption for estimation of overall time for an evacuation-type of traffic flows. It is worth noting that the assumption is only regarding acceleration: which remains constant during the travel of the vehicle on a link (i.e., braking times at intermediate locations are ignored). While the acceleration is assumed to be constant, the velocity, on the other hand, is indeed varied over its travel time, until the vehicle reaches the speed limit of that link, at which time the velocity is capped to the speed limit.

The value thus computed for departure time is retained only if the calculated value is greater than the departure time of the vehicle ahead of this vehicle. The departure time of the vehicle ahead of this could be obtained by looking into the *EventHistory* list held by the road-segment connecting to the vehicle's next-hop intersection.

This transit time has to be reconciled with the vehicle-following constraints, namely, slow the vehicle down if one or more vehicles ahead of this vehicle prevent this vehicle from reaching at the computed transit time. A sum of departure time of the vehicle ahead (t_{dva}) and time needed to cover a distance of vehicle length ($v$len) at a speed specified by speed limit (s), is used as the departure time to ensure the vehicle following $t_d = t_{dva} + (v\text{len}/s)$.

Wait time at the traffic signal (t_w)

As mentioned earlier every road-segment has a GREEN time period during which it allows the flow of traffic through it and RED time period during which no vehicle is allowed to enter the road-segment. At any given simulation time, every intersection is capable of knowing the GREEN time period or RED time period of any road-segments in the road-network and hence the wait period on traffic signal for a particular vehicle is calculated determining when it enters any road-segment.

The departure time for every vehicle is then calculated as: $t_d = t_t + t_w$.

Each road-segment lane maintains a list named *EventHistory* that keeps track of the *arrival_time*, *departure_time* and *vehicle_id* of every vehicle that entered and left the intersection. The processing of preceding steps is performed as part of the processing for a VehicleEvent, as shown in Figure 4.

Figure 4 *VehicleEvent* processing block

VehicleEvent:
If(enough space to hold veh in road-segment)
{
- compute transit time (t_t)
- compute wait time (t_w)
- calculate departure time (t_d)
- send vehicle to next hop intersection at t_d
- record the event in EventHistory
}

2.2.5 Congestion

As mentioned earlier every road-segment keeps track of incoming and out-going vehicle events in an event list (*EventList)*. This is referred by the intersection to determine the occupancy of any of its road-segments at a given time. On vehicle arrival, the intersection looks into the occupancy of the road-segment that connects to the vehicle's next-hop intersection. The vehicle is scheduled to depart the intersection only if enough space in the road-segment is ascertained to exist at that point of time. On the other hand if the associated road-segment is completely occupied, then a congestion behaviour in the road-network is emulated. To realise this, the vehicle is held in the current intersection

as long as enough space is available in the corresponding out-going road-segment. The additional time delay (time spent in waiting for availability of enough space) that the vehicle experiences in the intersection before being scheduled for departure directly corresponds to the congestion in the road network. In the following paragraphs we discuss different approaches that we have implemented to realise this behaviour.

Simple rescheduling

In this approach, on detecting non-availability of space in the road-segment the arriving vehicle is re-scheduled to the same road-segment at the next GREEN time period (when this road-segment allows vehicle to pass through it). With this simple technique the vehicle is (logically) retained in the intersection as long as there is enough space in the outgoing road segment to accommodate it, also ensuring that the vehicle leaves the intersection in the order of its arrival.

This technique works well for minor congestion in the transportation network. However, if the arrival rate of the vehicles is far greater than the departure rate, then the simulation crawls toward its end. This is due to the need for regular polling required for rescheduling the departures; as a result of this polling, every vehicle suffers multiple reschedules on a single intersection, this in-turn gives rise to equivalent number of events.

Rescheduling with initial time adjustment

Needless polling and rescheduling of *VEs* degrades the performance in the previous strategy. If it were possible to ascertain the number of vehicles 'in limbo' that are ahead of any new arriving vehicle, we can calculate the time taken for that many vehicles to depart and schedule the vehicle arrival at that time in future. To this end, we maintain a variable named *nlimbo* that keeps track of the number of limbo vehicles in an intersection that have logically moved on beyond the road-segment. This variable is used to calculate the new departure time as $t_{new\_d} = nlimbo \times (v\text{len}/s) + t_d$, where t_{new_d} is the new departure time calculated; $v$len is the average vehicle length; s, is the speed limit on the road-segment and t_d is the departure time calculated based on *transit time* and *wait time*.

However, the *nlimbo* variable could not be used for rescheduling the already rescheduled events, since with this variable the order of arrival of vehicles is not taken into consideration. Hence, the time calculated using the *nlimbo* variable could be used only to schedule the newly arrived vehicles from peer intersections and not to reschedule the vehicles withheld previously due to congestion. While we could get better speed-up pushing the initial re-scheduling time of the arriving vehicle much farther in future, the later schedules of the same vehicle would be of constant time; hence this suffers from the same problem of the former model. Better run-time performance could be achieved for smaller periods of congestion, but for longer periods of congestion the run-time performance would be similar to the previous model.

Using a 'limbo list'

The best performance is obtained by keeping track of more information, namely, by maintaining a list (*limbo-list*) instead of a single *nlimbo* variable. Using this scheme, we were able obtain a faster simulation. The FIFO *limbo-list* preserves the order of arrival and obviates rescheduling congested vehicle events on the same intersection. Whenever a new vehicle arrives at the intersection, it is put into the *limbo-list*. The intersection ensures that enough space in the outgoing road segment is available before removing the vehicle from the *limbo-list*. If enough space is available, the intersection schedules the vehicle at the end of the *limbo-list* to ensure the FIFO order in traffic flow.

If enough space to hold the vehicle is not available on the outgoing road segment at that point of time, the vehicle is retained in the *limbo-list*. The removal of the vehicle from the *limbo-list* should be dependent on the availability of space on the out-going road-segment. To ensure this, a *SelfUpdateEvent* (*SE*) is scheduled to a future time, when enough space to hold the vehicle in the outgoing road segment lane is available. This future time, t_x, can be approximated to the departure time of the second vehicle at the farthest end of the corresponding outgoing road segment lane, which ensures spatial availability to hold at least one vehicle in the road-segment lane.

Thus, by maintaining the additional *limbo-list* and with *SE*, we were able to emulate congestion in the road-network and as expected, better runtime performance was also observed. Figure 5 gives the altered algorithm to process *VE*, to accommodate modelling the congestion behaviour in the road-network. Absence of the first step, i.e., the insertion of vehicle into *limbo-list*, is the only change in the algorithm used to process *SE* arrival.

Figure 5 Altered *VehicleEvent* processing algorithm to accommodate congestion behaviour

```
VehicleEvent:
- Insert the incoming vehicle in limbo-list
While (limbo-list not empty)
{
  if (space for veh in road-segment){
    - get the first veh from  limbo-list
    - compute transit time(t_t)
    - compute wait time (t_w).
    - calculate departure time (t_d)
    - send VE to next hop intersection at t_d
    - record the event in EventHistory
   } else{
    - create a SelfUpdateEvent (SE)
    - schedule SE at t_x
    - break from loop
  }
}
```

2.2.6 Memory management

As described earlier, each road-segment maintains a list to keep track of the events arrived and departed from that

road-segment, we refer to this list as the *EventHistory*. As the simulation progresses with time, the length of the *EventHistory* grows. This not only increases the memory requirement for the simulation run but also degrades the performance, since the list is used for calculating "number of vehicles in road-segment" that results in a search with $O(n)$ complexity. Hence, constant cleanup of the *EventHistory* is needed to overcome the memory and performance inefficiencies. To ensure safe cleanup, only the elements with vehicle arrival time less than the safe global time (Lower Bound on Time Stamp (LBTS)) are purged.

Using a periodic timer

If we were to use a timer that would schedule an event periodically to clear the *EventHistory*, it would solve the performance as well as memory problem. But, events for clearing the *EventHistory* will be generated even when the event history is not big enough. Also an inactive intersection (to which vehicles have not arrived yet) ends up needlessly performing this clearing operation. Hence, we generate and process reclaimable events while using a timer; and this in-turn affects the performance of the simulation.

Keeping track of length of the EventHistory

Another strategy is to keep track of the length of *EventHistory* to initiate the cleanup process. This strategy works fine but poses a problem when the vehicle arrival rate is very high. In this case, each arrival event looks into the length of *EventHistory* and schedules a cleanup event. If a check on previously initiated clean-up process is not made before initiating a new process, redundant events are created and processed. This would go on until the point where the very first arrival actually completes the cleanup process and updates the length of the *EventHistory*.

This problem of redundant event generation is overcome by making the scheduling of the clean-up process mutually exclusive, i.e., if an event has already scheduled a cleanup process, no other event would schedule one, until the initiated one completes. By doing so, we reduce many redundant events thus enhancing the runtime performance of the simulation model. The cleanup process of *EventHistory* in SCATTER-OPT is realised through an event type called *FlushEvent*.

3 Synchronisation and reversibility considerations

In what follows, an overriding consideration behind the modelling approaches and consequential data structures is to enable perfectly reversible computation of state changes. SCATTER-OPT is currently tested to work in both conservative and optimistic synchronisation modes. Reverse computation technique is used to realise rollback in optimistic synchronisation.

3.1 Reverse computation

Optimistic federates differ from their conservative counterparts in that they do not discard events after processing them. Instead they keep the events around, and also maintain copies of simulation states before modifying them as part of event processing. Since optimistic federates do not rely on lookahead, they execute their events without blocking for safety. Thus a federate will have to rollback its computation if/when it later receives events whose timestamp is less than its current simulation time. There are two main parts to such rollback

- undo local computation by restoring the state prior to erroneous event processing
- undo all events erroneously sent to other federates.

While the parallel simulation library performs these rollbacks at the library level, reversal code to restore the application data structure state, in the event of rollback, needs to be written and provided by the application.

3.2 Reverse event handlers

To recap from previous section, four events are used to realise the discrete event traffic model. They are: *CreateEvent* (*CE*), *VehicleEvent* (*VE*), *SelfUpdateEvent* (*SE*) and *FlushEvent* (*FE*). The *FEs* are utilised for optimisation purposes in a safe manner (reclaiming memory that strictly belongs to past that cannot be rolled back) and hence do not impact the correctness of simulation, hence the reversal of *FE* can be ignored. The *CE* is used to generate the traffic at the specified rate. The only state variable they alter is the *VehicleID* counter, which is incremented as vehicles are generated. Rollback of a *CE* involves decrementing this counter. In the experimental road network that we have considered, source intersections do not have any incoming events from any other intersections, thus eliminating the possibility of causality error occurrences. Hence, in this experimental setup *CE* reversal code is never utilised.

The *VE* and the *SE* events are responsible for routing the vehicles from one intersection to another (as shown Figure 6). Doing so, they alter the states of *EventHistory* and *limbo-list* data-structures at the application level. Hence, during rollback, reversal of the data structures to their previous states is necessary for correct rollback. Further, as discussed earlier, both *VE* and *SE* processing use the same algorithm; the only difference being that the former inserts the arriving vehicle into the *limbo-list*.

In the following, the reversal procedures for *VE* and *SE* processing are similar, unless otherwise noted. Each VE_i may generate one or many VE_{ij}, (where $j = 1, 2, 3, \ldots$) and, may or may not create an *SE*, based on the congestion in the network. Hence, for reversal we should first find out if *VEs* were generated and if so, how many were generated.

Figure 6 Algorithm for application level *VehicleEvent* and *SelfUpdateEvent* reversal

```
VehicleEvent (VE) /SelfUpdateEvent(SE)
If(VEs generated)
{
 loop(Number of generated VEs){
  - remove event-record from EventHistory
  - insert Veh into limbo-list (from front)
 }
 if (this event is VE){
  - remove last Veh. from limbo-list
 }
}
```

We know that when an intersection generates an out-going *VE*, it records that event in the *EventHistory* list. Hence, after finding the number of *VEs* generated, we need to remove the record for each event generated in the *EventHistory* list. Further, the vehicle object in that *VE* should be extracted and pushed back into *limbo-list* from the end through which it was removed. While, the *SE* reversal procedure completes here, the *VE* reversal goes a step further and removes only the last vehicle that was inserted in the *limbo-list* to complete its reversal procedure. This takes back the intersection road segment to the correct state prior to the processing of this *VE*.

After arriving at the reversal algorithm, implementation was found to be a challenge with the reverse execution interface. The reverse computation algorithm in the previous sub-section needs to identify the events that generated specific events and extract objects from the generated events. Modifications to the library had to be made to expose the event's causal list data structure, so that the application could iterate through the copy of events sent, to find the necessary events.

4 Sequential performance study

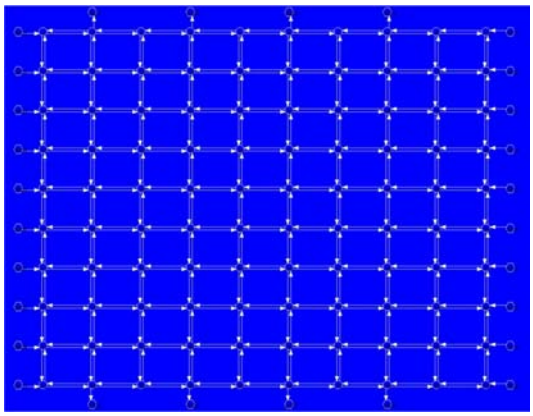
4.1 Benchmark network

In this section, we refer to road networks to be grids of size $N \times N$. The experimental setup contains $N \times N$ intersections, with $2 \times N$ sources injecting traffic from either sides (right and left), to eight destinations equally distributed on the top and bottom ends of the grid. Hence, our $N \times N$ grid scenario consists of $(N \times N) + (2 \times N) + 8$ intersections. Figure 7 shows a 10×10 road network grid, containing ten sources on left and right, and four destinations at top and bottom, in addition to 100 interior intersections.

Also, the traffic lights are specified with an 8 s cycle time. This cycle time is equally shared between GREEN and RED time periods. The intersection transit time is set to 1 s.

Road networks of dimension 10×10, 12×12, 14×14 and 16×16 were used for comparison. The intersections are connected to their neighbours through single-lane road-segments of length 1600 m (1 mile). The lengths of the road-segments are same throughout the test network, except for the road segments connecting the sources and sinks, which are of length 10 m. For each road network, sources generate traffic at a rate r vehicles/hour/destination, where, $r = 400$, 500, 600, 800 and 900. Larger grid sizes of the network were not considered for study since OREMS input format prevents it from executing beyond a 16×16-sized grid.

Figure 7 10×10 road-network grid (128 intersections) snapshot of OREMS graphical interface (see online version for colours)



4.2 OREMS and SCATTER input specifications

SCATTER-OPT is tested sequentially for getting an idea of its raw sequential speed. To make sure it is close to the best sequential performance available today, it is compared with OREMS (Evacuation Modelling System). OREMS is an aggregate model, very fast in execution, and is used in emergency operations.

OREMS input file specification

In OREMS, the information describing a traffic network is fed to OREMS simulation engine as a data-file with a specific format. The information is in the form of various record types, each record type consists of 80 columns. Information must be entered in the correct columns to properly describe the network. The code that we developed creates an input file considering record types 00–06, 11, 35, 36, 170, 175, 176, 195 and 210. The record types 01–06 identify the run, the time period data, output specifications and other run control data, while record type 00 corresponds to record comments. Record type 11 defines the surface street links of a network. The record-types 35 and 36 define signal control at every node in the network. Record type 170 is used as a delimiter record and one such record is required for each time period. A time period in ORMES is an interval during which the network conditions do not change. OREMS provides the capability to specify how the time-dependent data items change in course of a simulation run by allowing the user to partition the simulation time into series of time periods of varying lengths. In the OREMS input file that we generate, we consider a single time period. Record types 175 and 176 provide the origin-destination

data for traffic assignment. Record 195 provides the positional information (longitude and latitude) of every intersection in the road network. Record type 210 is required to mark the end of the input specifications for a time period and the final record in the input stream must be a card of type 210. Further, the ascending order of the record types must be ensured in the input data-file of OREMS.

SCATTER input file specification

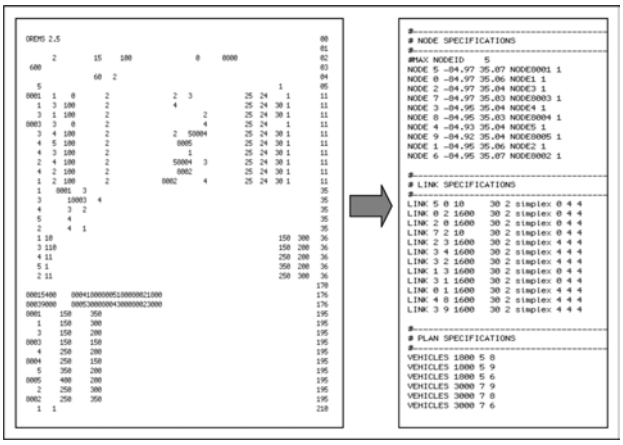
The SCATTER input file uses three types of records; they differ from each other based on their starting word. The records could start either with NODE, LINK or VEHICLES. The NODE record specifies the physical attributes like the positional information of each intersection in the road-network. The LINK record specifies the information of each road-segment in the road-network along with the traffic light associated with it. The VEHICLES record specifies the traffic assignment information in the road-network.

OREMS and SCATTER input file generators

OREMS software provides a Graphical User Interface (GUI) to build a desired vehicular traffic network. The GUI converts the user input in to a file. The usage of this GUI to build large transportation networks is unwieldy and time consuming. Hence, software to generate OREMS input file for Grid based test road networks was developed. Input file for initialising SCATTER is generated by parsing the input file generated for OREMS. Figure 8 shows the snap shots of the input files of OREMS and SCATTER.

Origin volumes and destination capacities need to be provided in OREMS for trip generation and traffic assignment. Node numbers of form 8xxx represent entry and exit nodes connected to an internal node. SCATTER does not distinguish between normal intersections with source and sink intersections; hence any intersection could be a source or sink. However, to keep the experimental scenario uniform in both the systems, some nodes are used only as sources, only as sinks or as normal intersections in SCATTER.

Figure 8 Conversion of OREMS input file into a SCATTER input files



4.2.1 *Hardware*

The sequential performance comparison of SCATTER-OPT with OREMS was carried out on an Intel® Core™2 Duo CPU T7700 at 2.4 GHz, with 2GB of memory running Microsoft Windows XP Professional SP2.

4.3 *Performance*

Simulation runtime is plotted in seconds to evacuate traffic generated at rates as specified in the experimental setup against the number of vehicles evacuated, for both OREMS and SCATTER-OPT.

As seen from Figures 9 and 10, in both 10×10 and 16×16 networks, the discrete event-based SCATTER-OPT model runtime is smaller than that of OREMS at lower input traffic rate, but SCATTER-OPT' runtime slowly increases beyond OREMS' as the input traffic rate increases. Similar pattern is observed for grid sizes 12×12 and 14×14.

However, the increase in simulation time of SCATTER-OPT on larger networks is negligible when compared to runtimes of equivalent high-fidelity (vehicle-level) simulators. In separate experiments, we benchmarked the same networks with MITSIM and TRANSIMS and obtained runtimes that were at least one order of magnitude higher (i.e., simulations were $10\times$ slower than OREMS and SCATTER-OPT).

Figure 9 Simulation runtime of OREMS and SCATTER-OPT against number of vehicles evacuated plot, for 10×10 grid (see online version for colours)

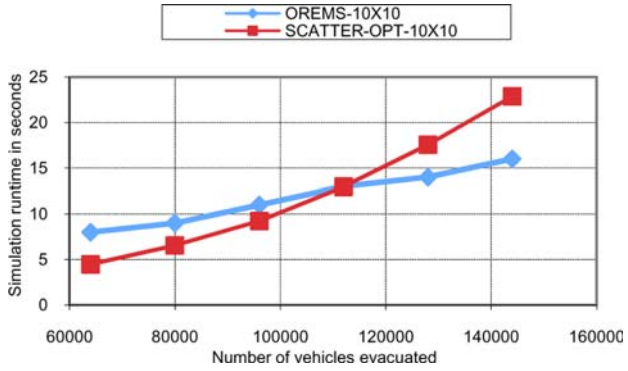
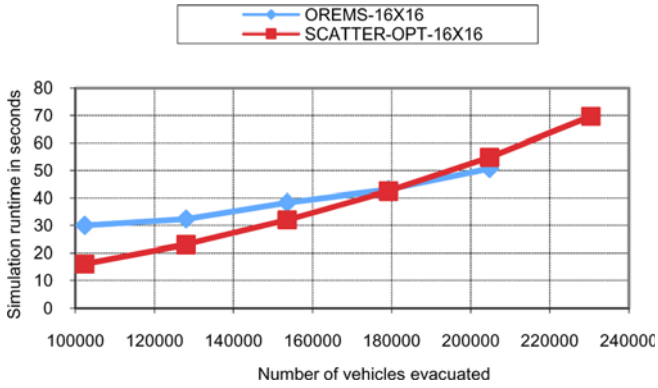


Figure 10 Simulation runtime of OREMS and SCATTER-OPT against number of vehicles evacuated plot, for 16×16 grid (see online version for colours)

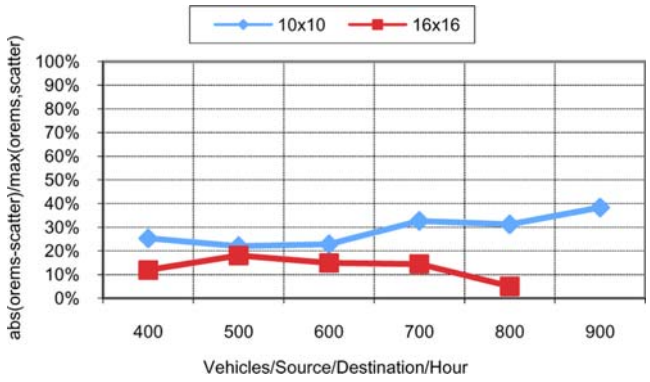


4.4 Validation

For the validation purpose we compare the results of the evacuation scenarios (10 × 10 grid and 16 × 16 grid layouts) from SCATTER-OPT with those of OREMS. Validation is performed by comparing the evacuation times predicted by the two tools and verified the closeness of the predicted times for different network sizes of the benchmark network.

The ratio of difference in the evacuation times (of OREMS and SCATTER) to the maximum predicted evacuation time can be considered as a measure of closeness in evacuation time predicted by either tool. The percentage of the difference observed is plotted against the rate of vehicle generation in terms of vehicles generated per source per destination per hour, as shown in Figure 11. On an average, a discrepancy of 30% for 10 × 10 network and 15% for the 16 × 16 road-network is observed from the plot. A reduction in the difference of predicted evacuation times with increase in the road-network size is also observed. For practical purposes, such a difference is well within margin of uncertainty, and hence, both the simulations can be considered to equivalent, and hence, SCATTER-OPT-based prediction is considered to give similar evacuation time as predicted by OREMS.

Figure 11 Percent discrepancy in evacuation time prediction (see online version for colours)



5 Parallel performance study

We now turn to a detailed performance study of the parallel execution of SCATTER-OPT. First, the runtime for simulating relatively smaller networks is evaluated. Sequential runtime of OREMS is compared to that of one- and two-processor runs of SCATTER-OPT. Next, scalability of SCATTER-OPT is measured on significantly larger network sizes. Ultimately, the performance of traditional conservatively synchronised execution is seen to be significantly improved when optimistic execution is enabled in conjunction with reversibility of our traffic model.

5.1 Absolute speedup of 2-processor runs

To demonstrate the absolute speedup of SCATTER-OPT, we present the simulation runtime comparison of OREMS and SCATTER-OPT (with one and two processors) on the same 16 × 16 road network scenario considered for sequential runs in Figure 12. The distribution of the intersections across two federates was done as shown in Figure 13.

Figure 12 Simulation runtime of OREMS and SCATTER-OPT (using one and two processors) against number of vehicles evacuated, for 16 × 16grid (see online version for colours)

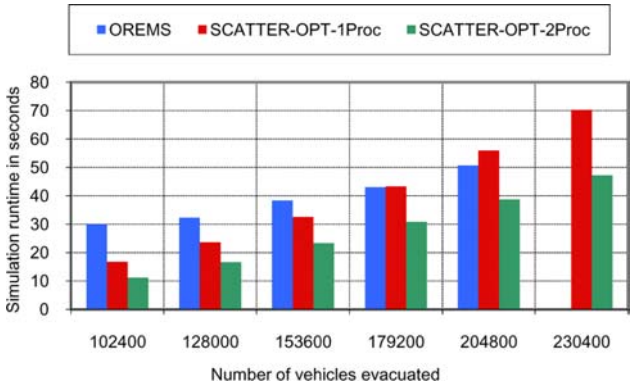
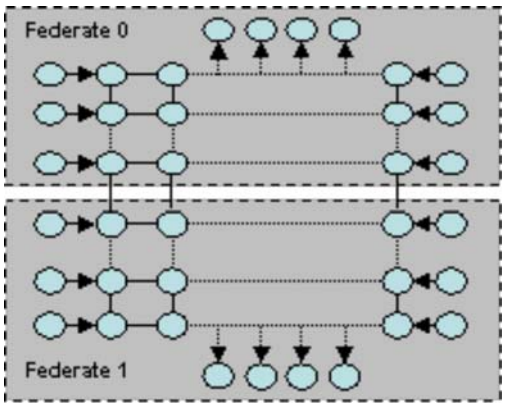


Figure 13 Distribution of intersections across two federates (see online version for colours)



The serial and parallel SCATTER-OPT runs were carried out again on the same hardware, but running Mac OS X 10.4.11 operating system. Note that the runtimes of single processor runs carried out on Mac OS X in Figure 12 closely correspond to the one taken on Windows XP in Figure 10. The reduction in the simulation runtime with the increase in the number of processors, demonstrate the absolute gain in speedup of SCATTER-OPT.

5.2 Larger road-network scenario

Previously, we considered a constant time of 1 s as the time required for a vehicle to cross any intersection (the space that connects road-segments). This is used as the minimum lookahead TP in conservative synchronisation and this time period is constant across all intersections. In general, the transit time to cross the intersection space could be variable, potentially smaller than 1 s. In conservative mode, if the lookahead is large, it is clearly favourable for the runtime

performance since the simulation can take long strides. On the other hand, smaller values of lookahead could make the simulation run slower. The simulation performance would be worsened for a broad lookahead range, since the minima of these lookahead values is taken into consideration while calculating the global virtual time. With the range of input network scenarios, the simulator is bound to be used with a range of lookahead values. Hence, the study of the performance of the simulation model with decrease in lookahead becomes significant for both conservative and optimistic synchronisation based models. We evaluate the performance on two such representative extremes of lookahead values, namely, 1 s and 0.1 s, respectively.

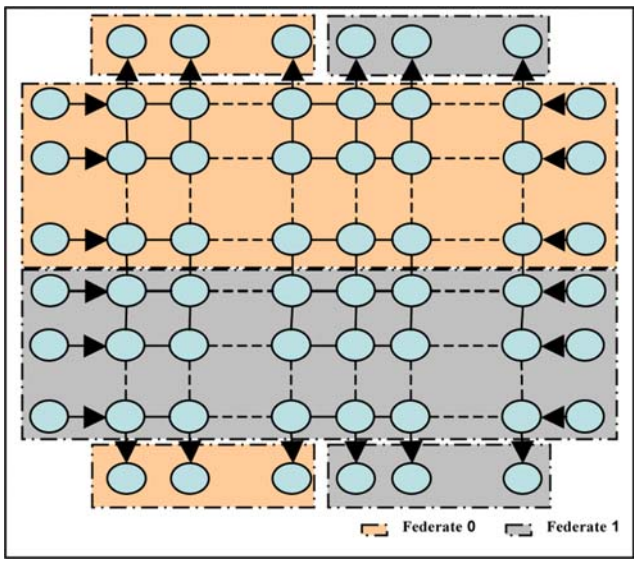
5.3 Experimental setup

The experimental setup contains $N \times N$ intersections, with $2 \times N$ sources injecting traffic from either sides (right and left), to $2 \times N$ destinations equally distributed on the top and bottom ends of the grid. Hence, our $N \times N$ grid scenario consists of $(N \times N) + (2 \times N) + (2 \times N)$ intersections. For the optimistic and conservative performance comparison purposes, we have considered a 64×64 network grid, with 128 sources generating traffic at 400 vehicles/hour/destination rate, toward 128 destinations.

5.3.1 Partitioning the road network

The input network grid is divided into blocks of rows, and intersections (including the sources), falling in each block run on one federate, the destination intersections are equally divided among all federates. For example: a 64×64 road-network grid, when divided among two federates, each federate gets 64×32 intersections and 64 sources (32 left and 32 right) and 64 (32 top and 32 bottom) sinks. Figure 14, shows the distribution of intersections among two federates.

Figure 14 Distribution of intersections across processors or federates (see online version for colours)



5.3.2 Hardware

The parallel runs to study the conservative and optimistic runtime performances were carried out on Our Institutional Clusters (OIC). The OIC cluster consists of a unique bladed architecture from *Ciara Technologies* called *VXRACK*. The *VXRACK* contains 80 usable nodes. Each node has *Dual Intel® 3.4GHz Xeon EM64T* processors, *4GB* of memory and dual *Gigabit Ethernet* interconnects. All nodes run *Red Hat Linux Enterprise WS v4* operating system.

5.4 Parallel performance for 100% evacuation

Here, we discuss the performance of the model that simulates the evacuation of all (100%) of vehicles in the network. In a 64×64 grid case, the simulation consists of around 6.5 million vehicles generated from 128 sources, through 4096 intersections toward 128 destinations. Figures 15 and 16 presents observed simulation runtimes (in hours) for parallel runs across number of processors used. The two curves seen in these figures pertain to the runtimes of the parallel runs using conservative and optimistic synchronisation. The effect of the values for intersection crossing time is evaluated, giving two estimated lookahead times, namely, 1 s and 0.1 s.

Figure 15 64×64 grid, Simulation runtime against number of processors, with lookahead 1 (see online version for colours)

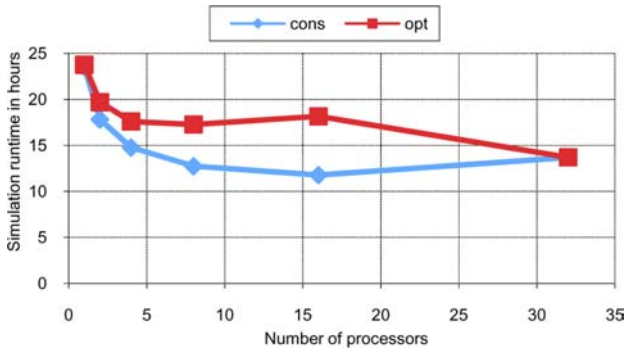
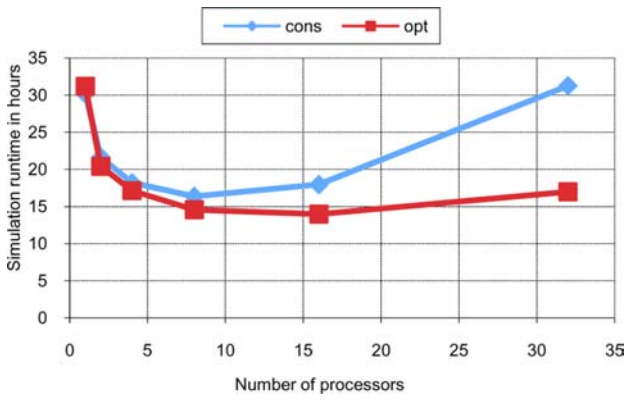


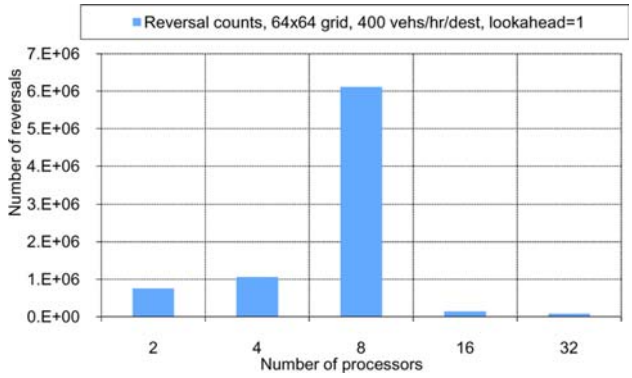
Figure 16 64×64 grid, Simulation runtime against number of processors, with lookahead 0.1 (see online version for colours)



With a lookahead of 1 s, a higher runtime in the model with optimistic synchronisation is seen and the conservative

mode performs better than optimistic. The degradation in the performance of the model using optimistic synchronisation is attributed to the number of reversals incurred due to rollback. The performance gain expected due to optimistic synchronisation is lost due to the higher number (of order 10^6) of reversals. However, with 32 processors the runtime of the optimistic performance significantly improves due to very small reversal counts. The bar graph in Figure 17 shows the reversal counts recorded for simulation runs with varying number of processors. Note that the reversal counts plotted here correspond to *VE* and *SE* reversals only. The *FE* reversal-count is not considered, since no code is invoked for its reversal.

Figure 17 64 × 64 grid, Number of reversals against number of processors, when the lookahead is 1 (see online version for colours)



As we reduce the lookahead to 0.1 s, the simulation runtime for conservative mode starts to rapidly increase with increase in number of processors. This deteriorated performance can be attributed to the frequent synchronisation requirement. On the other hand, with optimistic synchronisation, the simulation runtime decreases with increase in the number of processors.

From Figure 16, we see that with a lookahead of 0.1, the 16 processors simulation runtime for optimistic mode is around 14 h; the corresponding conservative mode value is around 18 h that increased from its lowest of 16.5 h with 8 processors. Hence, using optimistic mode a drop of around 2.5 h (15%) is achieved using 16 processors over the best simulation conservative mode runtime, when the lookahead is 0.1. Similar observation with a lookahead of 0.1 can also be made in 128 × 128 grid scenario that models the evacuation of around 6.5 million vehicles from 256 sources toward 256 destinations through 16,384 intersections, as shown in Figure 18. Thus, in modelling vehicular traffic network, where the lookahead is not fixed, optimistic synchronisation (reverse-computing) provides a better promise for timely simulation results.

The reversal counts exhibit an interesting trend that does not fully correlate with the overall speedup and performance. The lightly-loaded scenario represented by the 64 × 64 grid size, reflected in Figures 17 and 19, show the effects that a relatively lightly loaded scenario induces on optimistic execution. The lower load translates to larger discrepancies of the optimistic time horizon across processors (because some processors are more or less heavily loaded than the others). More reversals occur towards the tail end of the simulation than the beginning or middle of simulation, because of the smaller amount of event load on processors that do not have as many final destination nodes. As will be seen in later results, the rest of the scenarios exhibit cleaner phenomena, and map easily to the expected trends, namely, optimistic execution performing better than conservative execution under low lookahead conditions, on large vs. smaller network sizes, and the simulation time depending upon the fraction of evacuated traffic.

Figure 18 128 × 128 grid, Simulation runtime against number of processors, with lookahead 0.1 (see online version for colours)

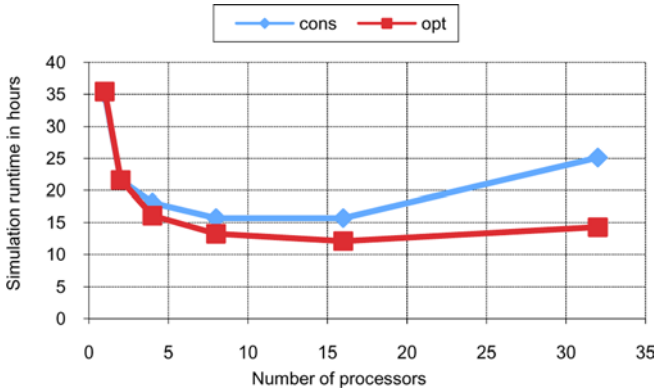


Figure 19 128 × 128 grid, Simulation runtime against number of processors, with lookahead 1 (see online version for colours)

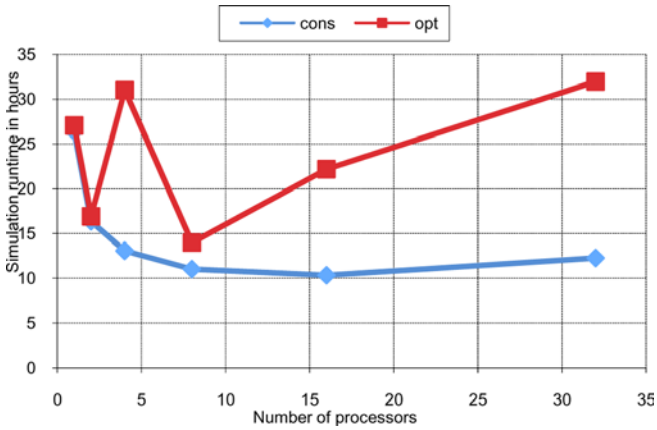
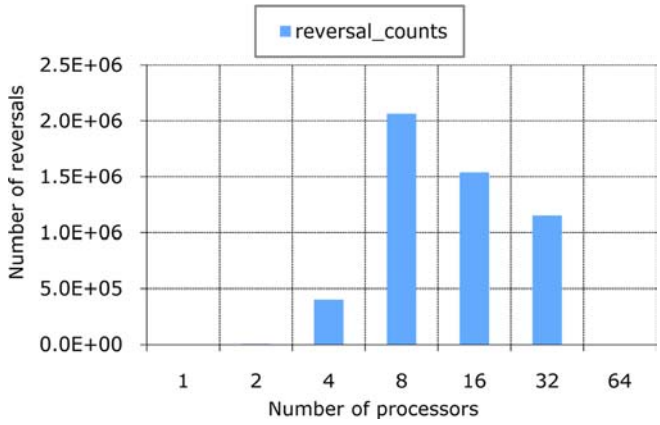


Figure 20 shows the observed variation of reversals with the number of processors. The simulation results in both optimistic and conservative execution have been verified to be correct and evacuation time results are repeatable, for both sequential and parallel runs (evacuation time shows negligible variation among runs with varying number of processors, varying flush event periods, and so on).

Figure 20 128 × 128 grid, Number of reversals against number of processors, when the lookahead is 1 (see online version for colours)



5.5 *Parallel performance for 75% evacuation*

In the previous section, we demonstrated the absolute-speedup of SCATTER-OPT by comparing its single processor run with the macroscopic model of OREMS. We also compared the conservative and optimistic parallel performance by varying a lookahead values in the 64 × 64 grid and 128 × 128 grid evacuation scenarios that simulated the evacuation of 6.5 million vehicles. The 64 × 64 grid scenario modelled the evacuation of over 6.5 million from 128 sources through 4096 intersections toward 128 destinations, while the 128 × 128 scenario modelled the evacuation of same number of vehicles but from 256 sources to 256 destinations through 16,384 intersections. However, in both these scenarios the minimum simulation time required to model the evacuation scenario regardless of number of processors used was 10 h to 15 h. Such runtime is still significantly high.

It was found that evacuation of most of the vehicles in the road-network happens in around 1/3rd of simulation runtime needed for complete 100% evacuation. When the number of events processed per synchronisation step (the so-called Lower Bound on (incoming) Time Stamp (LBTS) computation), is plotted as the simulation proceeds, an interesting phenomenon is seen. In Figures 21 and 22, we plot number of events per LBTS against the simulation-time for 64 × 64 and 128 × 128 grids, these values correspond to a single-processor run with lookahead 1. In Figure 22, the same data is shown differently, with the number of events per LBTS plotted with increasing simulation time.

Figure 23 provides similar plots for 256 × 256 grid scenario where in 13 million vehicles generated from 512 sources make their through 65 thousand intersections to their corresponding destination, one among the total of 512 destinations in the grid. Due to memory constraints we were not able to run this scenario with less than 8 processors and hence have used the output data from 8 processor conservative run to plot the graph. In this graph to calculate the number of events generated per LBTS, we summate the number of events generated by all the 8 federates and divide by LBTS time period.

Figure 21 Plot of number of events computed per lbts with respect to simulation time in hours from a evacuation scenario of 64 × 64 grid on a single processor (see online version for colours)

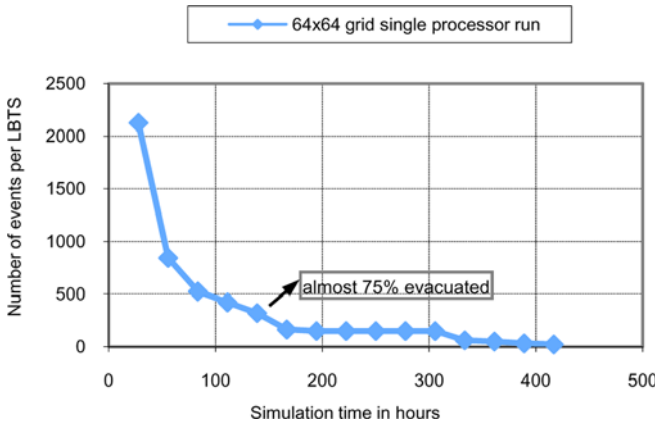


Figure 22 Plot of number of events computed per LBTS with respect to simulation time in hours from a evacuation scenario of 128 × 128 grid on a single processor (see online version for colours)

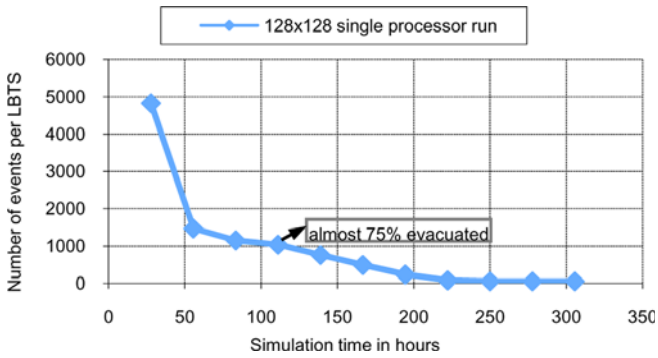
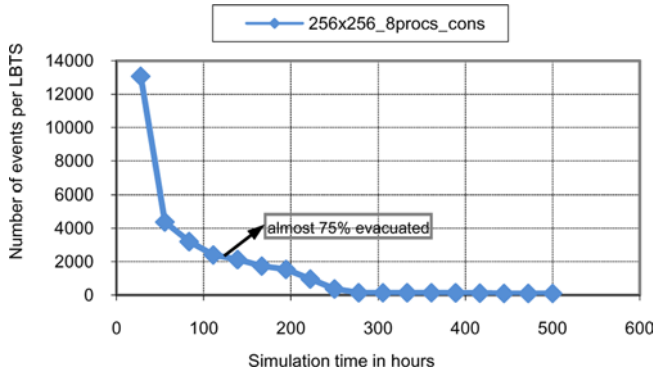


Figure 23 Plot of number of events computed per LBTS with respect to simulation time in hours from the evacuation scenario of 256 × 256 grid conservative run on eight processors (see online version for colours)



It is evident that a large number of events are processed during the initial stages of the simulation and the processing drops off drastically during the later stages leaving a long tail toward the end of the simulation, as seen from Figures 21–23. For the performance of SCATTER-OPT discussed earlier, the simulation was run till all the generated vehicles reach their corresponding destination. Hence, if we were to simulate for the evacuation of 75%

of the generated vehicles a better performance in the simulation runs can be expected since the simulation does not crawl through the long tail toward its end. With the logical processes having relatively large number of events to process even at the end of simulation, more computation is carried out and less time is lost in synchronisation; hence a better parallel performance than previously observed is expected in a 75% evacuation model.

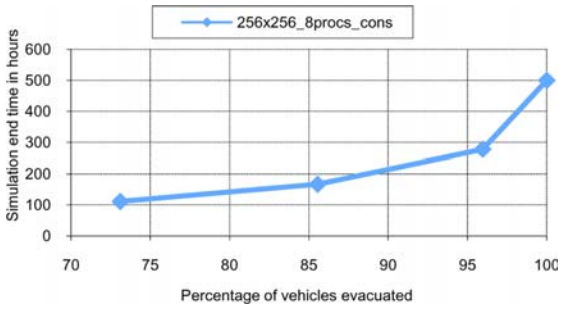
We ran 256 × 256 grid scenario with lookahead 1, on 8 processors using conservative synchronisation for different simulation end times. Figure 24 gives an idea on how the reduction in the simulation end time affects the simulation-runtime. In Figure 24(a), we see that as the percentage of number of vehicles evacuated decreases, the simulation end time drops drastically, as we reduce the percentage of evacuation from 100% to 75%, the simulation end time reduces from 500 h to 111 h. This reduction in end-time directly corresponds to the reduction in simulation run-time as seen from Figure 24(b).

Figure 25 presents the reduction in the simulation time in 64 × 64, 128 × 128 and 256 × 256 grid scenarios as the percentage of vehicles evacuated is reduced from 100% to 75%.

Scenarios with 75% evacuation of the generated traffic in 64 × 64, 128 × 128 and 256 × 256 grids are studied here for performance comparisons while using conservative and optimistic synchronisation methods with varying lookahead values. The rest of the plots are shown with two lookahead value extremes: 1.0 and 0.1. For each lookahead value, performance is logged for grid sizes of 64 × 64, 128 × 128 and 256 × 256.

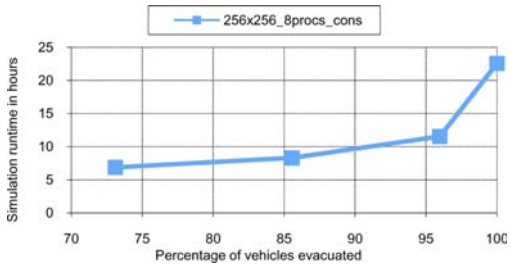
Overall, it is seen that with the larger lookahead, as expected, conservative synchronisation works fine. On the other hand, with the smaller lookahead, optimistic synchronisation becomes competitive with conservative synchronisation. At the largest network size, 256 × 256, and the largest processor count of 64, the gain in speedup is the largest. The power of reverse computation and optimistic execution become evident only on the largest configurations reported here, namely, on 64 processor and 256 × 256 network size. The plots are given next, in Figures 26(a)–31(b).

Figure 24 From 256 × 256 grid scenario, lookahead 1, 8 processor conservative run: (a) percentage of vehicles evacuated over simulation end-time and (b) percentage of vehicles evacuate over simulation runtime (see online version for colours)



(a)

Figure 24 From 256 × 256 grid scenario, lookahead 1, 8 processor conservative run: (a) percentage of vehicles evacuated over simulation end-time and (b) percentage of vehicles evacuate over simulation runtime (see online version for colours) (continued)



(b)

Figure 25 Simulation time vs. percentage of vehicles evacuate in 64 × 64, 128 × 128 and 256 × 256 grid scenarios with the traffic generation rate of 400, 100, 50 vehicles/hour/destination, respectively (see online version for colours)

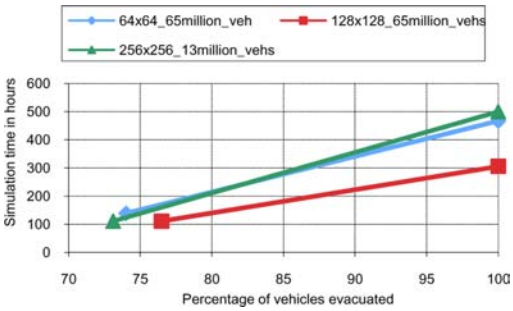
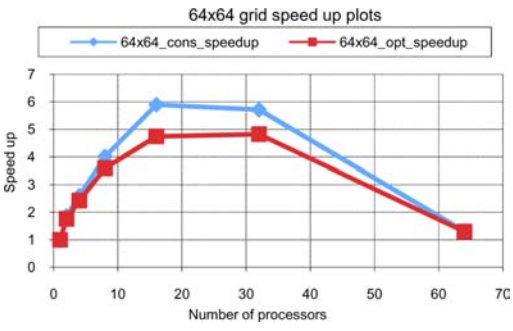
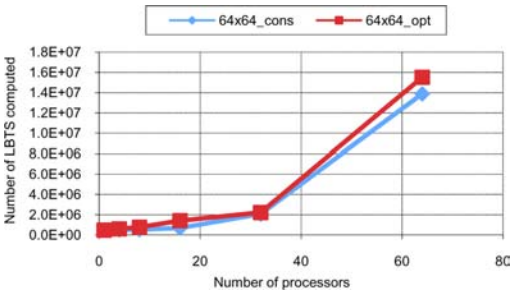


Figure 26 75% evacuation scenario of 64 × 64 grid, with lookahead = 1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



(a)

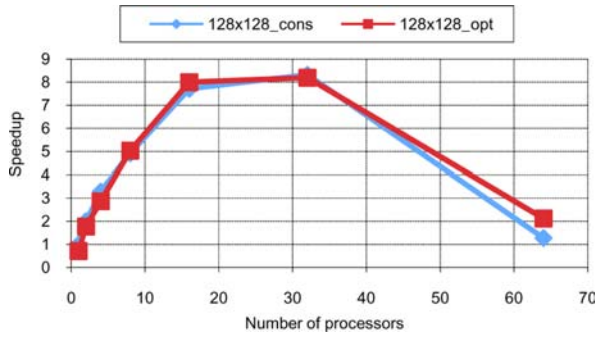


(b)

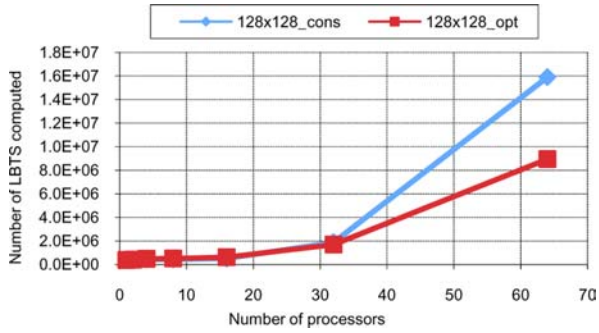
With lookahead = 1.

64 × 64 grid.

Figure 27 75% evacuation scenario of 128 × 128 grid, with lookahead = 1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



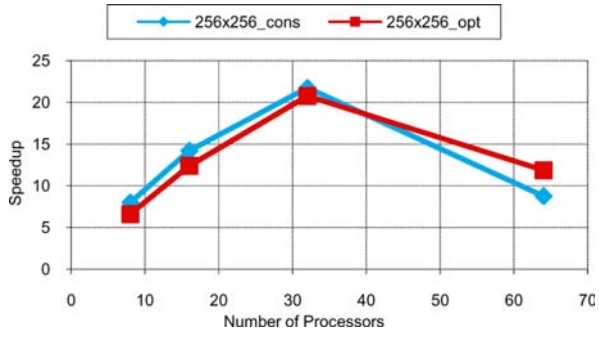
(a)



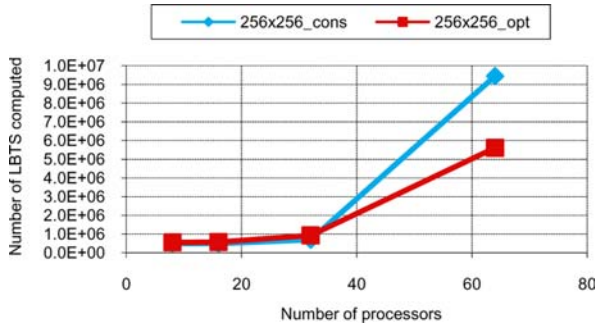
(b)

128 × 128 grid.

Figure 28 75% evacuation scenario of 256 × 256 grid, with lookahead = 1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



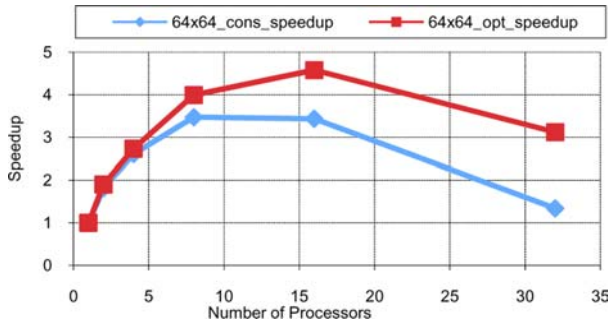
(a)



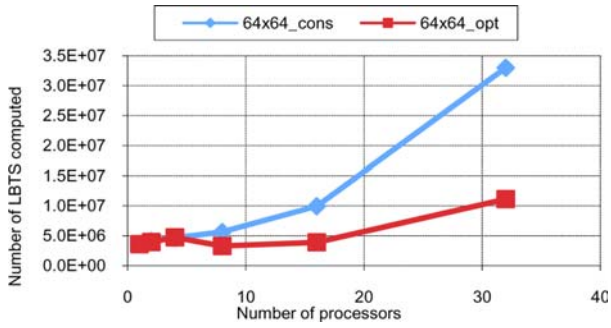
(b)

256 × 256 grid.

Figure 29 75% evacuation scenario of 64 × 64 grid, with lookahead = 0.1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



(a)

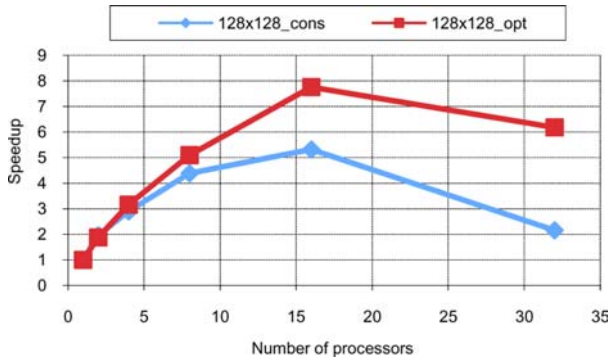


(b)

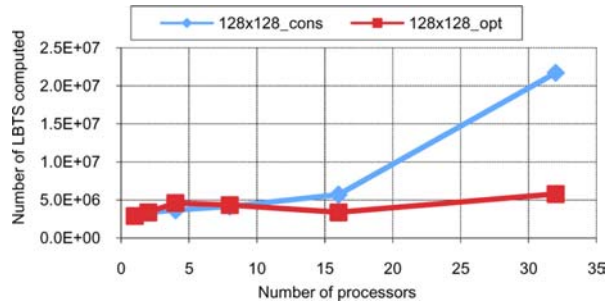
With lookahead 0.1.

64 × 64 grid.

Figure 30 75% evacuation scenario of 128 × 128 grid, with lookahead = 0.1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



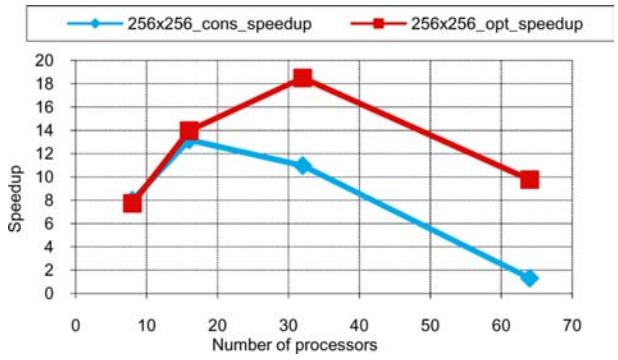
(a)



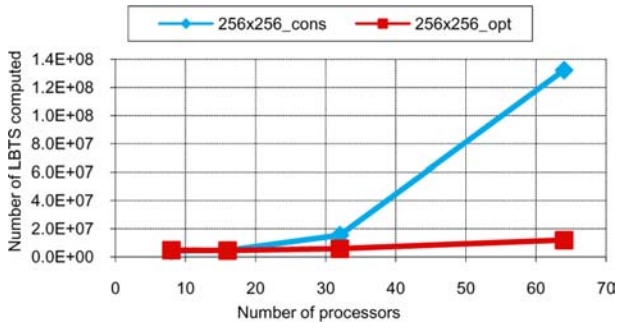
(b)

128 × 128 grid.

Figure 31 75% evacuation scenario of 256 × 256 grid, with lookahead = 0.1: (a) speedup plot and (b) number of events/LBTS across number of processors used (see online version for colours)



(a)



(b)

256 × 256 grid.

In Figure 31, for the largest network size and large number of processors, optimistic synchronisation shows its best performance, namely, a speedup of nearly 20 on 32 processors.

6 Summary and conclusion

In this paper, we discussed the design, development and performance of a parallel discrete event vehicular traffic simulation model. A discrete event formulation forms the basis of the sequential model, while being reversible for use in parallel execution. We ensured its sequential performance is comparable to that of one of the best available transportation model (OREMS). This sequential speed is achieved based on SCATTER's discrete event nature. To this model, we incorporated the relatively less explored method of reverse-computing based optimistic synchronisation for parallel discrete event models. It was implemented in a library called μsik, which permits both conservative as well as optimistic event processing modes. We compared the parallel performance of models using optimistic and conservative synchronisation techniques fixing the input traffic generation rate and varying lookahead values. To our knowledge, this is the first attempt at applying optimistic simulation techniques to parallel vehicular network simulation. The perfectly reversible formulation of the model is also novel that enables reverse computation. Additionally, the performance improvement is challenging due to the requirement of low parallel computation overhead needed to compare favourably with an extant, fast sequential simulator (OREMS). In that vein, the absolute speedup (i.e., speedup compared to OREMS), rather than self-relative speedup (speedup compared to SCATTER-OPT on 1-processor) is an additional strength. The performance, as expected, is observed to improve with increasing network size and decreased amount of lookahead. In one of the best performance runs, a speedup of nearly 20 is observed on a 256 × 256 node network with several million vehicles transferred, representing nearly a 100% improvement over conservative synchronisation.

6.1 Future work

While current implementation is limited to a single lane per direction per road segment, support for multiple lanes needs to be added. Performance on generalised networks remains to be evaluated, although we expect the challenges to only lie in optimising intersection-to-processor mapping for performance; the software is capable of delivering correct results with any arbitrary assignment. Performance of rollback using reverse computation could be compared to that of state saving. Also, in high lookahead conditions, we are investigating the effect of limiting optimism in order to reduce the amount of rollbacks.

Acknowledgements

Constructive comments by external reviewers and ORNL internal reviewers have helped improve the presentation. This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. Accordingly, the US Government retains and the publisher, by accepting the paper for publication, acknowledges that the US Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US Government purposes.

References

Bhaduri, B., Liu, C. and Franzese, O. (2006) 'Oak ridge evacuation modelling system (OREMS): a PC-based computer tool for emergency evacuation planning', *Symposium on GIS for Transportation.*

Cameron, G.D.B. and Duncan, G.I.D. (1996) 'PARAMICS, parallel microscopic simulation of road traffic', *Journal of Supercomputing*, Vol. 10, pp.25–53.

Carothers, C., Perumalla, K.S. and Fujimoto, R.M. (1999) 'Efficient optimistic parallel simulations using reverse computation', *ACM Transactions on Modelling and Computer Simulation*, Vol. 9, pp.224–253.

Fellendorf, M., Schwerdtfeger, T. and Stauss, H-J. (1996) DYNEMO, a mesoscopic traffic flow model to analyze ATT measures', *Transportation Planning Methods.*

Fisher, K.M. (2000) 'Transims is coming!', *Public Roads*, Vol. 63, pp.49–51.

Franzese, O. and Han, L. (2001) 'A methodology for the assessment of traffic management strategies for large-scale emergency evacuations', *11th Annual Meeting of ITS America*, Miami, FL, USA.

Garrett, Y., Christopher, D.C. and Shivkumar, K. (2003) 'Large-scale TCP models using optimistic parallel simulation', *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation*, *IEEE Computer Society*, San Diego, California, pp.153–162.

Gartner, N.H. and Stamatiadis, C. (1998) 'Integration of dynamic traffic assignment with real-time traffic adaptive control system', *Transportation Research Record*, pp.150–156.

Innovative Transportation Concepts, I. (2001) *VISSIM Simulation Tool*, URL: http://www.itc-world.com/VISSIMinfo.htm

Laboratory, L.A.N. (2001) *TRANSIMS*, URL: http://transims.tsasa.lanl.gov/

Meister, K., Balmer, M., Axhausen, K.W. and Nagel, K. (2006) 'A comprehensive scheduler for a large-scale multi-agent transportation simulation', *International Conference on Travel Behaviour Research*, Kyoto, Japan.

Perumalla, K.S. (2004) *µsik – Software Package Homepage*, URL: http://www.cc.gatech.edu/fac/kalyan/musik.htm

Perumalla, K.S. (2005) 'µsik – a micro-kernel for parallel/distributed simulation systems', *Workshop on Principles of Advanced and Distributed Simulation*, Monterey, CA, USA, pp.59–68.

Perumalla, K.S. (2006) 'A systems approach to scalable transportation network modelling', *Winter Simulation Conference* IEEE, Computer Society, Monterey, CA,.

Perumalla, K.S. and Bhaduri, B. (2006) 'On accounting for the interplay of kinetic and non-kinetic aspects in population mobility models', *European Modelling and Simulation Symposium*, Spain.

Tang, Y., Perumalla, K.S., Fujimoto, R.M., Karimabadi, H., Driscoll, J. and Omelchenko, Y. (2006) 'Optimistic simulations of physical systems using reverse computation', *SIMULATION: Transactions of the Society for Modelling and Simulation International*, Vol. 82, pp.61–73.

Yang, Q., Koutsopoulos, H.N. and Ben-Akiva, M.E. (2000) 'Simulation laboratory for evaluating dynamic traffic management systems', *Transportation Research Record*, pp.122–130.