Large-scale dynamic transportation network simulation: A space-time-event parallel computing approach

Article *in* Transportation Research Part C Emerging Technologies · February 2017

DOI: 10.1016/j.trc.2016.12.003

Xuesong Zhou
Arizona State University
125 PUBLICATIONS **1,434** CITATIONS

Large-scale dynamic transportation network simulation: a space-time-event parallel computing approach

Yunchao Qu

State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing, 100044, China

School of Sustainable Engineering and the Built Environment, Arizona State University, Tempe, AZ 85287, USA

Email: quyunchao0613@gmail.com

Xuesong Zhou* (Corresponding Author)

Associate Professor

School of Sustainable Engineering and the Built Environment, Arizona State University, Tempe, AZ 85287, USA

Email: xzhou74@asu.edu

To be published in Transportation Research Part C: Emerging Technologies

This is a preprint version.

Abstract: This paper describes a computationally efficient parallel-computing framework for mesoscopic transportation simulation on large-scale networks. By introducing an overall data structure for mesoscopic dynamic transportation simulation, we discuss a set of implementation issues for enabling flexible parallel computing on a multi-core shared memory architecture. First, we embed an event-based simulation logic to implement a simplified kinematic wave model and reduce simulation overhead. Second, we present a space-time-event computing framework to decompose simulation steps to reduce communication overhead in parallel execution and an OpenMP-based space-time-processor implementation method that is used to automate task partition tasks. According to the spatial and temporal attributes, various types of simulation events are mapped to independent logical processes that can concurrently execute their procedures while maintaining good load balance. We propose a synchronous space-parallel simulation strategy to dynamically assign the logical processes to different threads. The proposed method is then applied to simulate large-scale, real-world networks to examine the computational efficiency under different numbers of CPU threads. Numerical experiments demonstrate that the implemented parallel computing algorithm can significantly improve the computational efficiency and it can reach up to a speedup of 10 on a workstation with 32 computing threads.

Keywords: synchronous parallel strategy; mesoscopic transportation simulation; space-time-event network; parallel discrete event simulation

1. Introduction

Compared to the sequential computing mode utilized in most existing traffic simulation and planning models, parallel computing not only efficiently utilizes widely available distributed computing powers and communication networks, but also redefines what is tractable for time-critical transportation simulation and management strategy optimization. Emerging multi-core computer processor techniques are offering unprecedented available parallel computing resources, through a wide range of high-performance laptops and desktops currently available in the market. This paper aims to develop a parallel algorithm design for transportation simulation to exploit this paradigm change in computing and to facilitate the most efficient use of emergent parallel hardware.

1.1. Literature review

At the core of transportation simulation, traffic flow models are interested in the quantitative relationship between flow, density and speed, and modeling the interactions between different agents. In a transportation network, there may be many routes between each origin and destination and agents choose better routes to reduce travel time. Motivated by network-wide traffic management application needs, such as regional traffic mobility analysis and real-time route guidance, dynamic traffic assignment (DTA) models has been increasingly recognized as an important approach for assessing performance of different traffic system management and information provision strategies. There are macroscopic, mesoscopic or microscopic simulation-based methods for generating time-dependent travel time measures in general traffic simulators and DTA models (Mahmassani et al., 1994; Mahmassani, 2001; Ben Akiva 2002; Peeta and Ziliaskopoulos, 2001; Adler and Blue, 2002; Celikoglu and Dell'Orco, 2007; Chen et al., 2009; Di Gangi and Cantarella, 2016; Dell'Orco et al., 2016). In an effort to reach the right balance between representation detail and computational efficiency, this study focuses on how to implement a mesoscopic-based dynamic network loading model, within a parallel computing framework, on medium and large-scale real-world networks.

One key method to achieve computational efficiency while simulating medium and large-scale networks is parallel computing. A parallel simulation implementation is valuable for researchers to quickly examine the interactions of vehicular flow and analyze the complex traffic phenomena on a larger scale. It also helps to improve the computational efficiency of traffic model validation and calibration, as well as on-line traffic state estimation and prediction, e.g., through a simulation-based optimization framework. In early studies, parallel computing techniques have been applied in several transportation simulation systems (Junchaya and Chang, 1993; Wong, 1997; Ziliaskopoulos et al, 1997). PARAMICS (Cameron and Duncan, 1996) was implemented on a Connection Machine CM-2, and its graph partition algorithm divided the set of links into different sequences of queues and each queue contained a certain number of moving vehicles. Based on a shared memory platform, Nagel and Schmidt (2000) and Cetin et al. (2003) studied the parallelization of microscopic transportation simulation based on TRANSIMS (Transportation Analysis and Simulation System) using another sophisticated graph partition algorithm. Potuzak (2012) reported a distributed microscopic discrete time-stepped simulator DUTS (Distributed Urban Traffic Simulator) and performed road traffic simulation on a cluster of computers with multi-core processors. Kallioras (2015) applied a GPU-based accelerated metaheuristics approach to solve the transit stop inspection and maintenance scheduling problem. In the field of traffic assignment, Florian and Gendreau (2001) offered a good review on parallel computing approaches for performing the shortest path algorithms, e.g., through network decomposition and network replication strategies. Liu and Meng (2013) demonstrated a solid effort for accelerating the Monte Carlo simulation method for solving probit based stochastic user equilibrium problems using a distributed computing system. Ng and Nguyen (2015) proposed a spatial partitioning method to implement parallel computing. Morosan and Florian (2015) applied a shared-memory strategy focused on parallel shortest path computation and reported that the speedup could reach up to 20 when solving the traffic assignment problem. Auld et al. (2016) developed an agent-based modeling software development kit POLARIS that contains a parallel discrete event simulation engine.

Other early implementation of parallel transportation simulations are also introduced by Barceló et al., (1996) and Lee & Chandrasekar (2002) and a parallel implementation of AIMSUN reported a speed-up of 3.5 on 8 CPUs using multiple threads. As a macro-particle model, a parallel version of DYNEMO has been implemented since 2001 (Nagel and Rickert, 2001). DYNASMART's research team reported their experiment in implementing functional decomposition (Mahmassani et al., 1994). DynaMIT introduced a parallelization concept of functional decomposition (i.e., task parallelization) (Sundaram et al., 2011). Based on GPU techniques, Zhen et al. (2011) recently proposed a parallel computing framework to speed up the traffic simulation and optimize the traffic signal timing.

A significant amount of attention has been devoted to advancing parallel implementation for traffic simulation models for specific hardware/software architecture. The major efforts are summarized as follows: (1) in a static fashion, partitioning different geographical areas of the studied region to different CPU cores, and (2) for distributed computing, designing sophisticated message passing and efficient synchronization methods to reduce communication overhead among different computing cores. In our research, from a broader perspective of parallel discrete event simulation, we aim to offer a more feasible task decomposition methodology to synchronize inter-correlated space-time simulation events. This space-time-event oriented approach could take advantages of automated coordination programming interfaces (e.g. through OpenMP) between threads, processors, distributed computers, and Graphical Processing Unit (GPU).

An important study by Nie et al. (2008) offered a comprehensive discussion on a unified dynamic network loading/simulation framework in capturing congestion propagation effects. They also clearly indicated that their double-buffer-based network loading framework could be used for further parallel simulation prototype development and system implementation. However, to ensure the actual speedup under specific parallel computing architecture, in-depth research is still critically needed to examine a number of important system implementation issues and address how to select an appropriate space-time resolution and simulation execution sequences for mesoscopic or microscopic simulation.

1.2. Space-time-event view for parallel computation

Many further developments (Fujimoto, 1990, 1993; Ferscha and Tripathi, 1998; Liu, 2009; Fujimoto, 2015) summarize a number of parallel processing algorithms in terms of time-parallel and space-parallel categories. In a space-time view presented by Chandy and Sherman (1989), a space-time discrete event simulation can be divided into regions of arbitrary shape and assigned to separate logical processors (LP) according to the spatial and temporal decomposable features (Liu, 2009). In a parallel computing method, a global simulation task is discretized into a set of communicating logical processes (LP), each LP has its own memory space and maintains its own simulation clock and event-list, which can be concurrently executed. One LP is only capable of processing events occurring in its sub-system and communications between different LPs takes place exclusively by exchanging events.

In the general field of parallel discrete event simulation, early research proposed some fundamentally important synchronization strategies, e.g., the CMB protocol (Chandy and Misra, 1979; Bryant, 1977) and the time-warping method (Jefferson, 1985). In the CMB Chandy–Misra–Bryant (CMB) algorithm, LPs are assumed to be connected statically via directional links. LPs communicate through timestamped messages, also called event messages, which are transmitted from one LP to another in a non-decreasing timestamp order (Jafer et al., 2013). The CMB mechanism avoids deadlocks by introducing null messages. A null message essentially indicates the future arrival of the next message.

Within this modeling framework, time-parallel simulation methods divide the space-time graph along the time axis into non-overlapping time intervals and assign them to different processors for parallel processing, while space-parallel simulation aims to partition the graph into a collection of space-independent subsystems. The space-parallel methods include two types of LP simulation frameworks: synchronous vs. asynchronous (Ferscha and Tripathi, 1998). In synchronous LP simulation, all LPs have the same global simulation clock and they are executed by a unique time-stepped procedure. In contrast, the asynchronous technique allows each LP to have its own local virtual time with generally different clock timestamps at a given point. The asynchronous LP simulation may cause causality errors, as some events (from the other LPs) could arrive later carrying a timestamp earlier than the target LP’s current simulation clock. Accordingly, a number of event-wise synchronization methods, including conservative protocol and optimistic strategy, are used for efficiently avoiding potential causality problems.

For a large-scale transportation simulator, in our view, the synchronous space-parallel LP simulation approach is a more desirable choice for several reasons. First, a transportation network is spatially consisting of sets of links and nodes and space-parallel simulation offers a more robust solution to decompose events. Second, in most mesoscopic transportation simulators, the length of simulation time intervals should be no shorter than the free-flow travel time of the shortest links in the network (e.g., 6 seconds used in DYNASMART). As opposed to the possibility of extremely short event execution time intervals such as 0.0001 s between two events in a generic discrete event simulator, the discretized time interval with reasonably fine resolution (e.g. 6 s) in a mesoscopic simulator enables an efficient use of barrier synchronization available in parallel processing environment.

1.3. Approach

A spatial graph partition approach has been implemented in several traffic simulation systems. In these systems, the events are not completely or logically separated which leads to reduced computational efficiency. To improve the computational efficiency, asynchronous LP simulation strategies, including a conservative strategy and optimistic strategy, have been applied in distributed computing traffic simulation. As an optimistic synchronization protocol, Hunter et al. (2009) proposed an innovative ad hoc distributed traffic simulation framework with the key elements of space-time memory, state aggregation, and rollback based synchronization.

In this paper, we introduce a space-time-event view to understand a parallel computation mechanism for large-scale mesoscopic traffic network simulation. Inspired by the classical conservative CMB protocol (Chandy and Misra, 1979; Bryant, 1977) and a few of early implementations by Mahmassani et al. (1994) and Nie et al. (2008), we model a direct traffic link as two event buffers. This double-buffer representation further classifies individual agent's movements as two types of events: arrival events (AE) and departure events (DE) at entrance buffers and exit buffers. This approach is consistent with the cumulative flow count-based traffic kinematic wave model proposed by Newell (1993) described in the Appendix. A unique space-time-event network-based parallel computing method is developed to schedule the AEs and DEs of all agents at different shared-memory or distributed processors. This study uses an Open Multi-Processing (OpenMP) Application Programming Interface (API) (Dagum and Enon, 1998) to facilitate our program to distribute computational tasks to different processors in a shared-memory multiprocessing environment. Without directly dealing with Message Passing Interfaces (MPI), the OpenMP API also provides a simple and flexible interface for developing parallel computing programs for Graphical Computing Units (e.g., Nvidia Tesla) developed by Intel Inc.

The rest of the paper is organized as follows. The next section presents the problem statement, network representation and a space-time-event view to decompose the event-based simulation to independent logical processes. Section 3 discusses the synchronous parallel computing implementation in a processor-space-time scheduling network. Finally, the proposed method is applied to four medium-scale and large-scale real-world networks with an examination of different speedup ratios under different CPU number configurations in Section 4.

2. Problem statement and framework of parallel computing

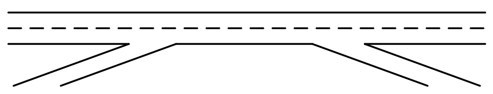
The parallel discrete event simulation system consists of events, actions (including movement actions and waiting actions), buffers, logical processes, and processors. Notations and definitions are listed below.

n	Index of a physical node.
l	Index of a physical link.
e	Event: agent's entering or leaving event with spatial and temporal attributes.
b	Buffer: upstream (entrance buffer) and downstream (exit buffer) segments of a link. A buffer stores a group of events occurring in this buffer.
a	Action: connecting two adjacent agent events. There are two types of actions: movement action and waiting action.
m	Movement action: A movement action connects two events in different buffers and it represents that an agent moves from a buffer to its spatially adjacent buffer. There are two categories of movement actions: link transfer and node transfer.
w	Waiting action: A waiting action connects two events in the same buffer and it represents that an agent is waiting in a buffer for a certain period of time. There are two categories of waiting actions: link waiting and node waiting.
LP	Logical Process: A LP contains events, buffers and actions. It independently executes a series of movements.
P	Processor: the logic circuitry that responds to and processes the basic instructions, e.g., CPU and GPU. One processor may contain one or more LPs. If there are many LPs, the processor could execute the LPs sequentially, e.g., LP1->LP2->LP3.

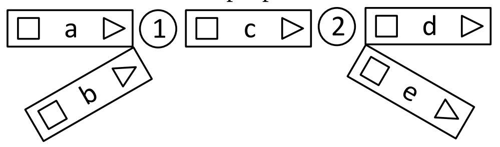
2.1. Problem statement

The proposed parallel computing method aims to perform a dynamic network loading (DNL) process. Given a set of time-dependent OD or path flow on a congested network, the DNL problem determines the time-dependent link/path travel times, and other traffic states such as link flow and density over a fixed time period. Consider a transportation network with a set of nodes $N = \{n\}$ and a set of links $L = \{l\}$, and the simulation time horizon is discretized to $t = 1, \ldots, T$. In traditional methods, the given data are the demand and supply data, which include (1) the initial time-dependent OD demand matrices between activity locations or traffic analysis zones, (2) the network supply in terms of time-dependent link capacity $q^{\max}(l,t)$, k_{jam} and lane miles $\Delta X(l)$ on each link l, as well as certain capacity distribution rules around intersections and freeway bottlenecks. Given a set of path inflow patterns, the dynamic traffic network simulation problem aims to find the cumulative arrival and departure curves on each link, which also leads to the simulated time-dependent link density and travel time along each used path. The output data include (1) individual time-space trajectory in the network, and (2) aggregated time-dependent link/path travel times. These data enable transportation researchers and software developers to expand its range of capabilities to various traffic management application, e.g., traffic prediction and analysis, emission estimation, traffic demand calibration.

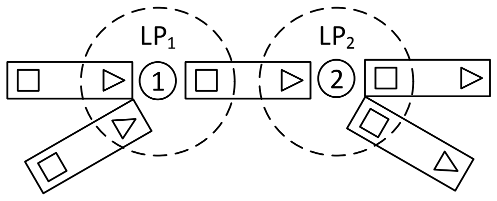
Specifically, the input of our core simulation model is a set of agents with given paths, which can be regarded as discretized path flows. For one OD pair, we first distribute the OD demand to path flows, then discretize the path flows to integer values, and finally generate a number of agents with these paths. Because we have assigned a certain path for each agent, the first -in first-out (FIFO) principle is adopted to describe the queue behavior and resolve capacity request conflicts at merge nodes. If there are several vehicles with the same entering time, the road capacity distributes to different paths or movements according to the path or movement flow proportion.



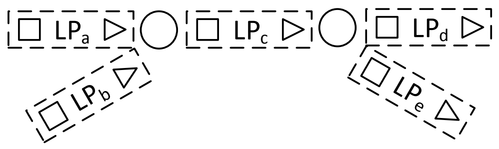
a) freeway corridor



b) double-buffer representation



c) node-based LP



d) link-based LP

Fig. 1 Network representation and logical process in parallel discrete event simulation

Fig. 1a illustrates a simple freeway corridor, which consists of the link set $\{a, b, c, d, e\}$ and the node set $\{1,2\}$. To clearly describe the event-based traffic simulation, a link is divided into two parts, namely entrance buffer (ENB) and exit buffer (EXB), to record agents' arriving and departing events. As shown in Fig. 1b, ENB (represented by a square) is located at the upstream of a link and EXB (represented by a triangle) is located at the downstream end. It should be noted that the exit buffer can be further decomposed to several exit buffers according to different link movements (directed to different downstream links) (Nie et al., 2008). Without a loss of generality, we only consider the single exit buffer situation in our paper.

In a parallel simulation system, each logical process (LP) possesses its own local simulation clock and local memory for private data. According to the spatial structure of a transportation network, the events of all agents can be spatially assigned into node-based LPs and link-based LPs. A node-based LP_n contains the EXBs of its upstream links and ENBs of its downstream links, while a link-based LP_l contains the ENB and EXB of the current link. For example, in Fig. 1c, there are two node-based LPs, that are LP_1 and LP_2. The node-based LP_1 consists of two exit buffers EXB_a and EXB_b and one entrance buffer ENB_c for vehicles to be loaded into the physical network. In Fig. 1d, there are five link-based LPs and each LP_l contains the ENB_l and EXB_l of link l.

2.2. Framework of transportation simulation

Fig. 2 illustrates the procedures of an overall parallel transportation assignment and simulation system. This process begins by reading the input supply and demand data from external files, and assigning a route for each agent. This routing task can be concurrently executed for each zone. After setting parameters, the simulation proceeds in a time-stepped strategy and at each time stamp there are five elementary sequential tasks. A node-based LP executes the node-based actions to perform the waiting action in an exit buffer or update spatial attributes of agents, while a link-based LP executes the link-based actions to perform the waiting action in an entrance buffer or update the timestamp attributes of agents.

Task 1: All vehicles with departure time t are loaded to the entrance buffers of their first links and some vehicles move forward to the entrance buffer if there is available space.

Task 2: The inflow capacity of each link can be calculated simultaneously using a traffic flow model, such as the simplified Kinematic Wave Model presented in the Appendix.

Task 3: Synchronize capacity distribution at each node (i.e., a bottleneck or queue server). Final outgoing capacity values for each incoming link are assigned, according to the inflow capacities of a bottleneck, using node models (e.g., Daganzo's (1995b) priority-based merge model).

Task 4: Node transfer from exit buffer to entrance buffer of connected links. Given assigned capacity, this procedure moves vehicles from the exit buffers of inbound links to the entrance buffers of outbound links or vehicles complete their trips at the destination nodes.

Task 5: Link transfer from the entrance buffer to exit buffer of the same link. Agents transverse from the link entrance buffers to link exit buffers by updating the travel times at the corresponding events. The arrival event at time t is deleted from the entrance buffer and replaced by a ready-to-depart event at time s at the exit buffer of the link. Here, the travel time $s - t$ depends on the traffic flow models and Newell's simplified kinematic wave model that uses free-flow-travel-time (FFTT) to move agents from the cumulative arrival curve $A(t)$ to the virtual cumulative departure curve $V(t)$. The cumulative departure curve $D(t)$ is finally updated when the agent moves out of this link with capacity quotas assigned from Task 3.

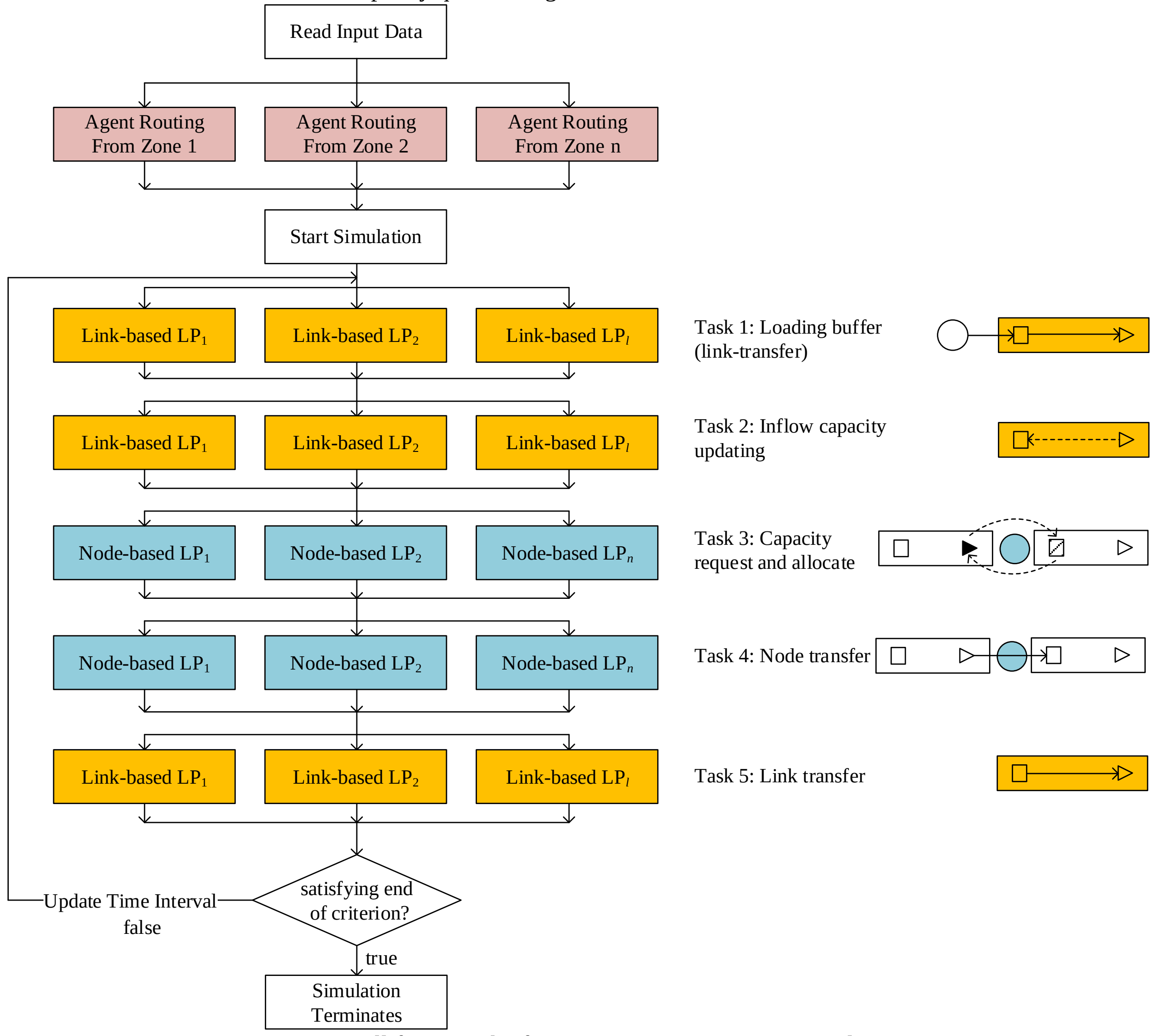

Read Input Data
Agent Routing From Zone 1
Agent Routing From Zone 2
Agent Routing From Zone n
Start Simulation
Link-based LP₁
Link-based LP₂
Link-based LP_l
Node-based LP₁
Node-based LP₂
Node-based LP_n
Task 1: Loading buffer (link-transfer)
Task 2: Inflow capacity updating
Task 3: Capacity request and allocate
Task 4: Node transfer
Task 5: Link transfer
satisfying end of criterion?
Update Time Interval
false
true
Simulation Terminates

Fig. 2 Overall framework of mesoscopic transportation simulation

The core of the traffic flow model used in this study is a discretized (vehicular) kinetic wave (KW) model. In the standard macroscopic KW traffic flow model and the node capacity distribution model, the values (i.e., traffic volume) are non-integers. In our simulation, an improved long-period pseudo-random number generator (Panneton et al., 2006) is used to generate uniform random numbers and then round floating-point capacity values to the nearest integers in terms of the number of vehicles.

After the simulation process, path-specific travel times are needed for the traffic assignment module. A standard way for approximating the link and path travel time is through tracking the link cumulative flow count curves, but it may involve interpolation and back-tracking and constitute a significant portion of the computational overhead. Snelder (2009) has proposed a method to reduce the error caused by the discretization rounding off errors. Different from the above flow-based method, we use vehicle trajectories from all agents to compute the time-dependent (experienced) path travel time and waiting time, based on the time stamps along individual paths. Define $t_p^a(\tau)$ as the travel time of agent a travels along path p with departure time τ, $t\left(a, l_p^{last}, EXB\right)$ as the time interval of agent a arrives the exit buffer (EXB) of the last link of path l_p^{last}. During the simulation, the individual entering time and leaving time of each buffer of each link are dynamically recorded, so the individual time-dependent path travel time $t_p^a(\tau)$ and the aggregated average path travel time $\bar{t}_p(\tau)$ can be easily calculated by Eq. (1).

$$t_p^a(\tau) = t\left(a, l_p^{last}, EXB\right) - \tau, \quad \bar{t}_p(\tau) = \frac{\sum_a t_p^a(\tau)}{N} \tag{1}$$

By following the time discretization approach presented by Lu et al. (2009), we generate time-dependent aggregated link travel cost using the shortest path algorithm and dynamic equilibrium gaps.

2.3. Space-time-event view for parallel computing

In this paper, we use a four-indexed notation (a, l, t, B) to record the space-time event (or state) of each agent, where a represents the index of an agent, l represents the index of a link, t represents the simulation time interval, and B represents the buffer type of the current link l. The space-time trajectory of one agent is recorded as an event list. For example, in Fig. 3, the space-time trajectory of one agent a in the corridor $l_1 \to l_2 \to l_3$ can be represented by the event list $(a, l_1, t_1, ENB) \to (a, l_1, t_3, EXB) \to \cdots \to (a, l_3, t_9, EXB)$. Accordingly, the input buffers store the events (a, l, t, ENB) that represent agents entering the upstream node of link l while the output buffers store the events (a, l, t, EXB) that represent agents entering and leaving the downstream node of link l. For a single agent/vehicle, the execution sequence of its events is dynamically driven by three categories of actions

1) Link transfer: $(a, l, t, ENB) \to (a, l, s, EXB)$, with a travel time of $s - t$ $(s > t)$.
2) Node transfer: $(a, l, t, EXB) \to (a, m, t, ENB)$.The downstream link m is predefined ahead of the agent's trip or determined by real-time routing behavior. In our study, we assume that the operation time of node transfer movement is zero to establish a clear activity-on-the-link network structure.
3) Node waiting: $(a, l, t, EXB) \to (a, l, s, EXB)$. Agent a is waiting at the exit buffer EXB of link l from time interval t and is ready to be considered for a move to the next link at time interval s. The waiting time $s - t$ depends on the inflow capacities of the adjacent downstream links and outflow capacity of the current link l.

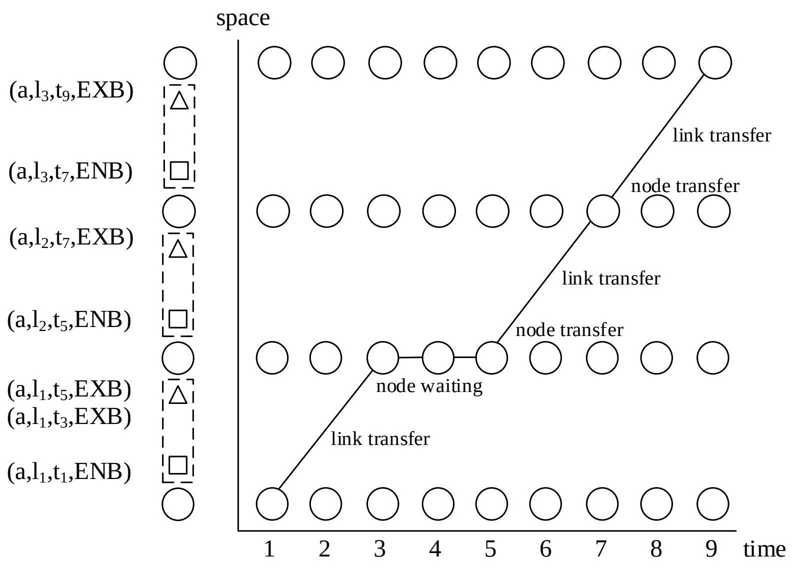


Fig. 3 Space-time trajectory and vehiclular event list

In a multi-agent simulation, tasks 2 and 3 proposed in Section 2.2 calculate capacity requests and allocation prior to moving agents, and during this process the agents' spatial and temporal attributes are not updated. After these two tasks, agents with permission perform node-transfer movements and link-transfer movements while other agents execute node-waiting actions/commands.

3. Scheduling logical processors for parallel computing implementation

Given a space-time-event representation presented above for an overall transportation simulation, we then need to execute all LPs and events synchronously or asynchronously in parallel. An important local causality constraint (Fujimoto, 1990) principle requires that every LP should process the events in a non-decreasing timestamp order. This section systematically describes an overall control/coordination mechanism to schedule the parallel processors to attain maximum speed-up.

3.1. Synchronous LP simulation

In most mesoscopic transportation simulators, the length of simulation time intervals are not shorter than the free-flow travel time of the shortest links in the network, e.g., 6 seconds used in DYNASMART. This discretized time horizon is very useful for conducting synchronous space-parallel LP simulation. Problems of deadlock and memory request are avoided with the help of the barrier synchronization mechanisms available in most every parallel processing environments (Ferscha and Tripathi, 1998). On the other hand, LPs with shorter execution times should wait for all other LPs to complete their executions and thus, this synchronous strategy could lead to idle time. If events are spatially and temporally distributed to all LPs in an even manner, the CPU time differences could be reduced, leading to less system-wide idle time.

While there are many advantages of adopting the synchronous LP strategy, we also need to recognize the following limitations. First, the time discretization might result in temporal precision errors as the occurrence time of every event should be integral multiples of the simulation time interval. Higher time precision requires finer simulation time resolution. Second, events are spatially decomposed into different LPs and each logical process can simultaneously manage non-overlapping sets of buffers. This is shown in Fig. 1-(c) for a node-based partition and Fig. 1-(d) for a link-based partition. It should be noted that under the space-parallel strategy, each LP can only communicate with their neighbors, which requires the distance d traveled by an agent in one time step Δt not to exceed Δl (condition (2)).

$$d \leq \Delta l = v_{\max} \cdot \Delta t \tag{2}$$

The LP parallel computing strategy can be further considered to a possible application in the microscopic traffic or pedestrian flow models that satisfy condition (1). Examples along line could include the floor field cellular automata model (e.g., Kirchner and Schadschneider, 2002) and force-based model (e.g., Qu et al., 2014; Qu et al., 2015) for pedestrian simulation, as well as the cellular transmission model for traffic simulation (Daganzo, 1994). In these models, an individual can only move to one of its neighbor cells/links in a single time stamp. However, in some other microscopic models, such as the cellular automata model for traffic simulation by Nagel and Schreckenberg (1992), a vehicle can move more than one cell/link at a time interval. In such a situation, condition (1) is not satisfied and the parallel strategy could be invalidated because it cannot exactly predict the potential conflict area and accordingly resolve the conflicts. To deal with such situations, some special microscopic models, such as simple linear car-following model (CF (L)), two cellular automata models (CA (L, M)), proposed by Daganzo (2006), can be converted to the kinematic wave model (KW) with a triangular fundamental diagram in order to enable the use of our proposed parallel computing strategy with simplified traffic flow models. For more complicated models considering lane changing and overtaking behaviors, it is still a very challenging task to design an effective parallel computing method.

The pseudo-code of the parallel simulation process is described as follows:

```
For time $t$ = 0 to $T$ (e.g., 24 hours)
  Step 1: Load newly generated agents to the loading buffer of their first links. If there is sufficient space available
          for each agent entering the entrance buffer, move it from the entrance buffer to the exit buffer.
  Step 2: Capacity request estimation for each link
     #pragma omp parallel for
     For link $l$ =1 to $L$
        Based on the external user input and traffic flow model, calculate inflow capacity for link $l$
        Use a link model and outflow capacities, adjust inflow capacity for link $l$
     End for // link
  Step 3: Synchronize capacity distribution at each bottleneck (merge and diverge)
     #pragma omp parallel for
     For each node $i$ = 1 to $N$
              Total Incoming Demand = 0;
        For each link $l$ in incoming link set $IL(i)$
             Total Incoming Demand += outflow demand from link $l$
        End for each incoming link
              If (Total Incoming Demand > Inflow Link Capacity $cap^{out}(l')$) //bottleneck
             For each incoming link $IL(i)$
                  Distribute inflow capacity to outflow capacity for each incoming link at bottleneck
             End for each incoming link
        EndIf
     End for each node
  Step 4: Node transfer from exit buffer to entrance buffer of connected links
```

```
        #pragma omp parallel for
        For each node $i$ = 1 to $N$
            For each link $l$ in the incoming link set $IL(i)$
                While (fetch the front vehicle from the exit buffer)
                    If current simulation time $t$ equals to or is greater than $t_v$, and both inflow and outflow capacity
                        are available (for kinematic wave based queuing models) // Check exit buffer
                        Move the vehicle from the exit buffer of link $l$ to the entrance buffer of link $l'$
                          Update cumulative arrival agent list on outgoing link $l'$
                          Reduce the inflow capacity on $l'$ at time t by 1
                          Reduce the outflow capacity on $l$ at time t by 1
                    Else
                          Break
                        End while
            End for each incoming link
        End for each node
    Step 5: Link traversal from the entrance buffer to exit buffer of the same link
        #pragma omp parallel for
        For each link $l$ =1 to $L$
            Arrival events at time $t_a$, departure events at time $t_v = t_a + FFTT(l)$, create the event into exit buffer,
            Update link cumulative arrival curve $A(l,t)$
        End for
End for //time
```

Fig. 4 briefly explains the fork-joint parallelism mechanism in OpenMP. A parallel region is a block of one or more statements with one point of entry at the top (*fork point*) and one point of exit at the bottom (*joint point*). Beginning from the code structure with hashtag "#pragma omp parallel for", a number of threads are created at the fork point to concurrently execute the decomposed tasks and then threads wait here for all threads to end at the joint point. Typically, a processor can only work on one thread per core. A CPU can have multiple cores and the hyper threading technology can allow one CPU to create and work on up to two threads per core.

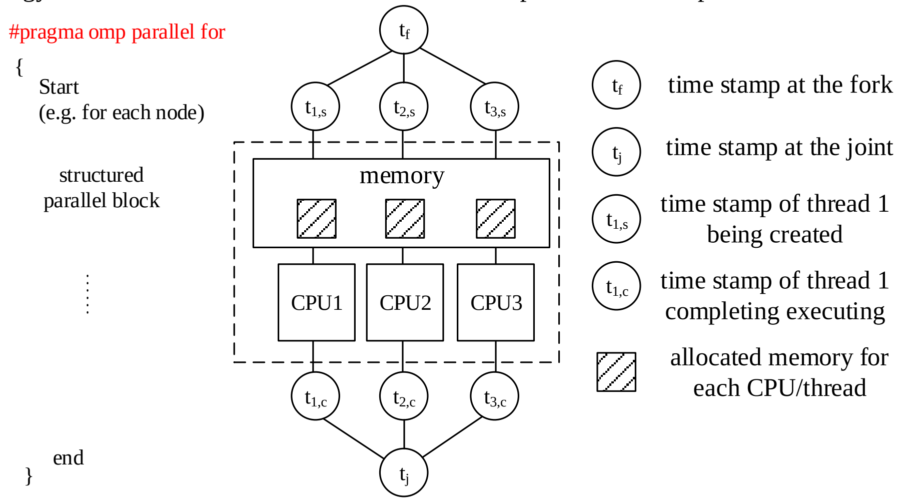
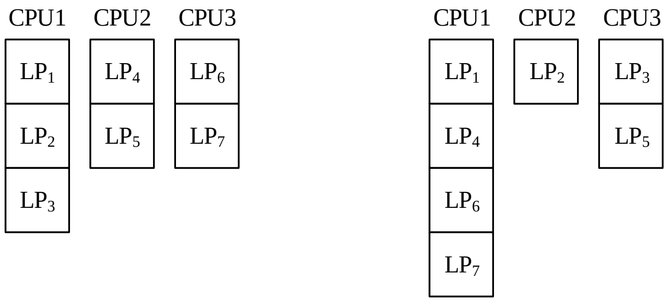


Fig. 4 Parallelism mechanism in OpenMP with multiple CPU threads

All CPU threads share the memory during the execution procedure in the structured parallel block, and OpenMP is able to dynamically create a private memory block for each task in a thread. Define t_f as the time stamp at the fork point, t_j as the time stamp at the joint point, $t_{i,s}$ as the time stamp of thread i being created, and $t_{i,c}$ as the time stamp of thread i completing its executing. Here, i is the total number of parallel executed threads. It should be noted that the time stamps are real computation time stamps, which are not simulation time stamps. The relationships between the time stamps can be expressed as Eq. (3).

$$t_f = \min\{t_{1,s}, \dots, t_{i,s}\}, t_j = \max\{t_{1,c}, \dots, t_{i,c}\} \tag{3}$$

It should be noted that OpenMP automates the distribution of the LPs (e.g. for different nodes, links or zones) to different processors during the simulation, and thus, the LP distributions may not be the same at different computing instances. Consider Fig. 5 as an example where there are three LPs (LP_1, LP_2, LP_3) and four LPs (LP_1, LP_4, LP_6, LP_7) allocated in CPU thread 1 at two time instances, respectively.



a) LP distribution at parallel computing instances 1

b) LP distribution at instance 2

Fig. 5 Different LP distributions at different parallel computing instances, a CPU label means a CPU thread

An important property for parallel computing is that different numbers of CPU threads and different LP distributions should not influence the simulation results. Thanks to the flexible coding structure in OpenMP, our program attaches its own random number seed for each node, link, or zone shown in Fig. 2. Thus, the computing process and results for each object (to be parallelized) remain the same, regardless of which CPU thread it is assigned to or how many CPU threads created by the system. We have proved this concept according to our simulation results.

3.2. Understanding the execution of simulation activities at multiple processors

The atomic traffic flow simulation logic at each time stamp consists of five sequential tasks, each of which is executed by a link-based LP or a node-based LP, as shown in Fig. 2. Focusing on the link-transfer and node-transfer tasks, Fig. 6 illustrates the space-time view of the dynamic LP decomposition procedure for a simple road segment example, which is a traffic corridor with two merge and diverge ramps. To simply describe the execution procedure, we only focus on nodes 1 and 2 with a little complicated topological structure. The simulation procedure has five link-based LPs and two node-based LPs, e.g., node-based LP_1 (node 1) contains the exit buffer of links a and b and the entrance buffer of link c.

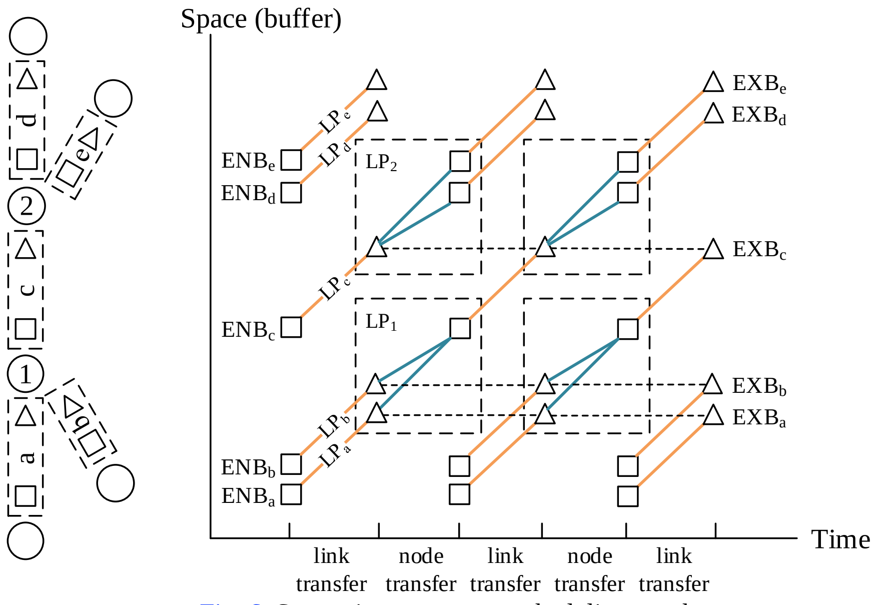


Fig. 6 Space-time-processor scheduling graph

Constructing a space-time-processor graph (Chandy and Sherman, 1989) is useful for us to understand how independent link-based and node-based LPs are assigned to different processors. This graph consists of a set of activity and barrier vertexes and a set of schedule arcs. Specifically, an activity vertex $v(i,t,p)$ represents a LP being executed at space i (a link l or a node n) at time t on processor p. A barrier vertex $v^*(i^*,t,p^*)$ represents a forking or joint activity at a virtual space i^* at time t on processor p^*, and it represents the intersection of results before re-distribution back to the CPUs. A schedule arc (v,v^*) connects an activity vertex and a barrier vertex. The schedule network is sequentially executed along the time dimension (in terms of simulation steps) and parallel computing activities are concurrently executed in the space dimension inside each link-based LP or node-based LP stage.

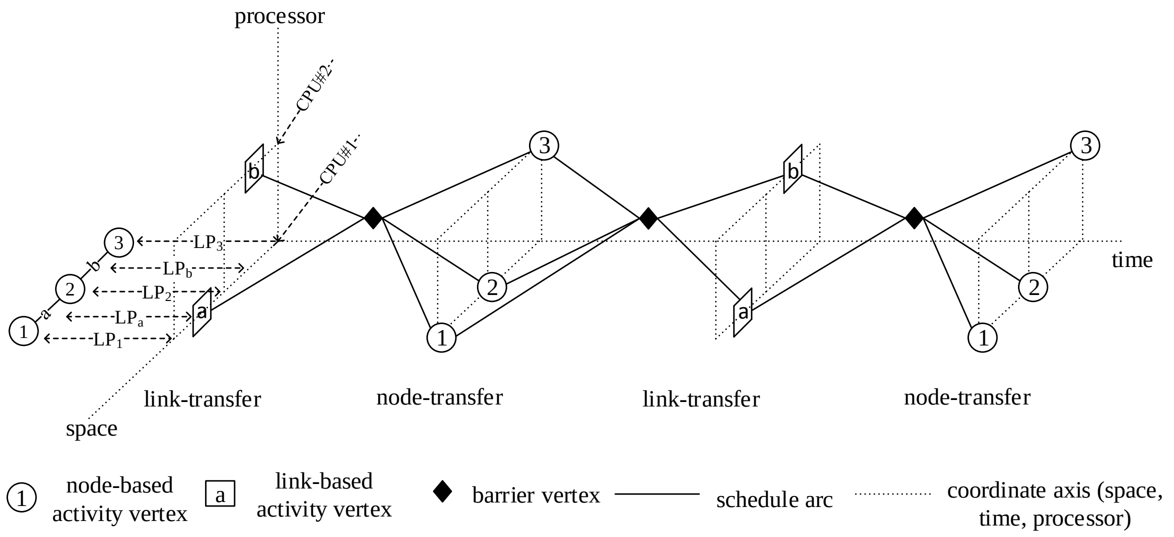


Fig. 7 Processor-space-time network representation

As illustrated in Fig. 7, we consider an example in a corridor (1->2->3) including 3 nodes (node #1, 2, 3) and 2 links (link #a, b), on a system with 2 CPUs. During the link-transfer step, link-based LP_a and LP_b are concurrently executed on CPU#1 and CPU#2, respectively; while inside the node-transfer step, the node-based LP_1 and LP_2 are performed on CPU#1, while LP_3 is performed on CPU#2. Between every two steps, there is a barrier vertex connecting the activity vertexes to synchronize all LPs. Fig. 8 illustrates the execution time flow chart of an instance of Fig. 7. Because fewer LPs are assigned to CPU2 compared to CPU1, the execution time of CPU2 is shorter than CPU1 and the overall speedup depends on the percentage of the idle time during each full execution step.

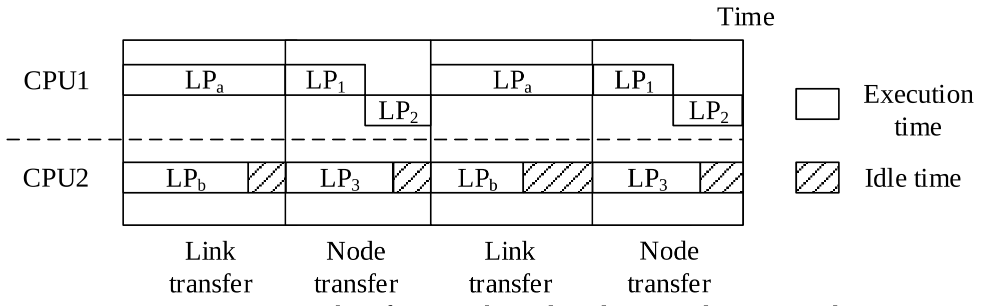


Fig. 8 CPU usage time series/Gantt chart for time-dependent discretized event simulation using OpenMP

4. Numerical experiments

The proposed parallel computing method is implemented on a workstation with an Intel Xeon E5-2680 2.80GHz multi-core CPU and 192 GB memory. This workstation has a total of 40 threads. To ensure the normal running of the workstation, we only use up to 32 threads for the simulation program in the experiments and reserve the remaining 8 threads for critical operating systems and other system-level threads. The source codes were implemented using Visual Studio C++ 2013 and its built-in OpenMP API library. Based on the proposed synchronous LP spatial-parallel computing method, our research group developed an open-source traffic simulation software DTALite (Zhou and Taylor, 2014; Zhou et al., 2015; Ruan et al., 2016) that was used to measure the total execution time and speedup under different numbers of threads.

4.1. Network and experiment design

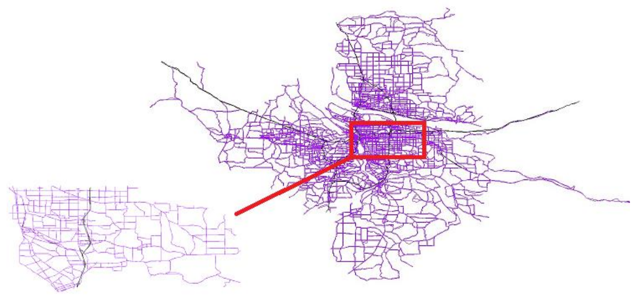
A set of experiments were performed on four different scales of networks. These include West Jordan network, Clackamas, Salt Lake City, and Maryland statewide network, with increasing scale respectively. Table 1 lists the supply and demand summaries of all the four networks. In scenario (a), the West Jordan area extracted from Salt Lake City (UDOT Report No. UT-15.09 by Zhou et al., 2015) is a small-scale network with 0.1K nodes and 0.4K links and the traffic demand loading period is 2 hours (15:30-17:30). In scenario (b), the Clackamas subarea network extracted from the Portland metropolitan network (Nevers et al., 2013) is a medium-scale network with 1.5K nodes and 4.1K links and the traffic demand loading period is 5 hours (14:00-19:00). In scenario (c), the Salt Lake City network is a large-scale regional network with 13.9K nodes and 26.8K links and the demand loading period is 5 hours. In scenario (d), the Maryland statewide network (Erdoğan et al., 2015) is a large-scale metropolitan/mega region network, which includes all of Maryland, Delaware and the District of Columbia, along with adjacent portions of Virginia, Pennsylvania and West Virginia. This network has a total of 26.6 M travelers during a 24-hour simulation horizon. Fig. 9 plots the corresponding networks and approximate peak hour demand values.

Tab. 1 Details of four real networks used for simulations

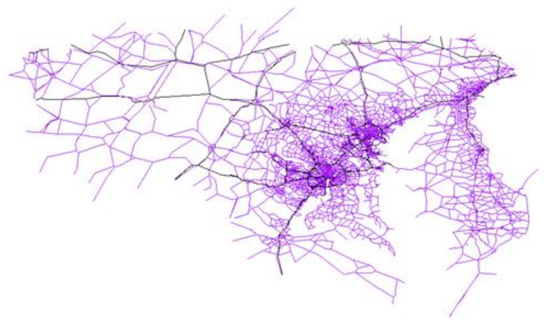
Scenario	Network	Number of nodes	Number of links	Number of agents	Demand loading period	Scale
a	West Jordan subarea	0.1K	0.4K	25.3K	2h	Small
b	Clackamas subarea	1.5K	4.1K	491.5K	5h	Medium
c	Salt Lake City region	13.9K	26.8K	1.3M	5h	Large
d	Maryland statewide	21.7K	54.5K	26.6M	24h	Large

a) West Jordan area in Salt Lake City
(0.1k nodes, 0.4k links)

c) Salt Lake City
(13.9k nodes, 26.8k links)



b) Clackamas subarea in Portland
(1.5k nodes, 4.1k links)



d) Maryland Statewide
(21.7k nodes, 54.5k links)

Fig. 9 Overall maps of four real-world transportation networks

4.2. Simulation results

The dynamic traffic assignment program includes both an assignment/routing stage and simulation stage. The dynamic network loading stage is able to produce time dependent link volume, speed, density, queue length, and has the ability to track trajectories of individual vehicles on each link. Focusing on the execution time of the simulation stage, each scenario is executed 10 iterations to reduce sampling errors for getting a tally of average execution time of dynamic network loading at each iteration. Taking scenario b) for example, the cumulative execution time of each iteration is listed in Table 2. The execution time of single-thread computing is 291.825 s. As the number of threads increases, the CPU usage is improved with shorter execution time, for example, with a speedup of about 9.30 in a 32-thread computing environment.

Tab. 2 Cumulative execution time of each iteration (Scenario b)

Iteration	Number of CPU threads								
	1	4	8	12	16	20	24	28	32
1	28.02	10.529	5.668	4.965	4.668	3.999	3.622	3.217	3.259
2	57.449	21.672	12.319	10.764	9.366	7.996	7.561	6.555	6.205

3	86.273	32.8	18.382	16.571	13.962	12.159	11.06	10.221	9.267
4	115.623	44.054	24.613	22.49	18.698	15.818	14.263	13.534	12.31
5	144.742	55.25	31.438	27.702	23.17	19.811	17.608	16.673	15.499
6	173.806	65.882	37.792	33.519	27.621	24.197	21.058	20.026	18.526
7	203.044	77.119	43.533	38.985	31.93	28.137	24.527	23.69	21.677
8	232.695	87.769	49.712	44.317	36.345	32.282	28.026	27.173	24.646
9	262.601	98.692	56.005	49.489	40.827	36.571	31.577	30.583	27.68
10	291.825	111.61	62.187	55.204	45.432	40.854	35.267	33.559	30.75

Recall that, a speedup is measured by comparing the execution time of a simulation using multiple threads and one obtained using one thread. The speedups of the aforementioned four test cases are plotted in Fig. 10, which indicate a possible nonlinear speedup as a function of the number of threads used in computing. Overall, both the scale of network (in terms of number of links and nodes) and travel demand (in terms of number of agents) could affect the overall parallel computing efficiency.

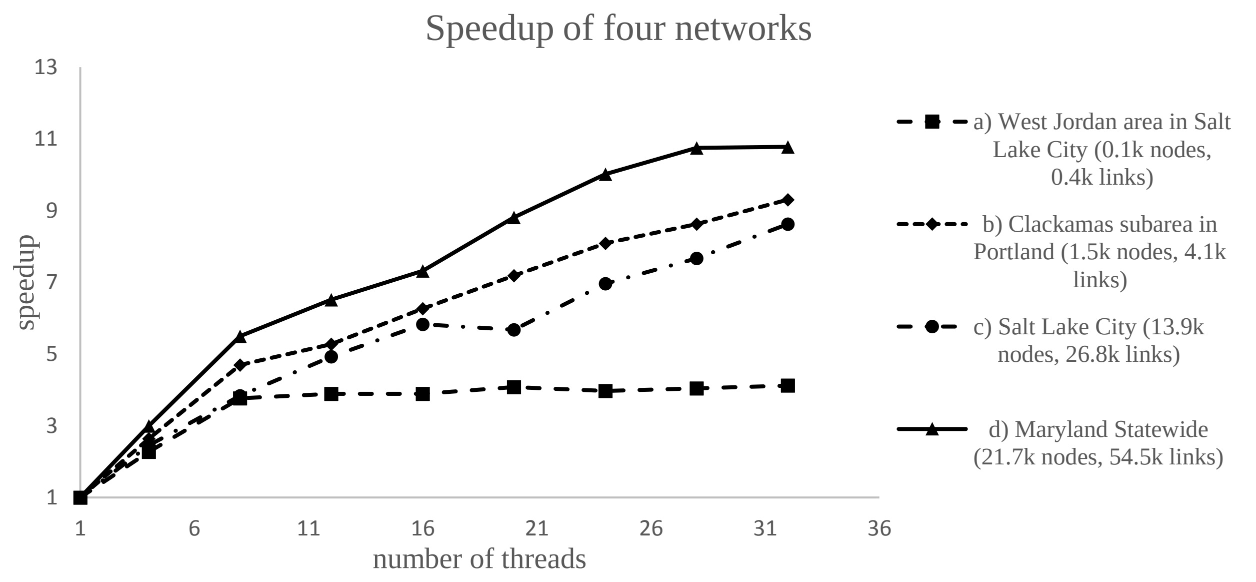


Fig. 10 Speedup of four networks

The parallel computing speedup values vary from 4.1 to 10.7 in simulating the four real-world networks. The speedup is higher for larger road traffic networks, because a large number of links and nodes allows a more balanced computing resource allocation to each processor, or more precisely, link-based or node-based LPs. In contrast, for smaller networks, the events have to be distributed to only a few of the processors, and therefore, the idle time of unused processors could be significant. The higher demand or a large number network could introduce a more balanced computing task assignment across nodes and links (at different CPU cores), which leads to a potentially higher speed up rate. The average execution time of scheduling events could dramatically increase with more frequent interactions and data exchanges between LPs. As a result, the overall communication overhead consumes more computation resources, leading to a slower increasing trend of the speedup curve. Overall, we have demonstrated that the speed-up in the parallel computing environment is significant regardless of the number of CPUs available in our example.

5. Conclusions

This paper has presented a parallel computing implementation approach for vehicular traffic simulation in real-world networks. According to the properties of individual events, a space-time-event view is proposed to allocate agent's entering and leaving events to different link buffers. Based on the double-buffer representation, we decompose the general mesoscopic transportation simulation into five sequential tasks that can be concurrently executed on link-based and node-based logical processes. The logical processes are then dynamically distributed to different processors by the proposed synchronous space-parallel LP strategy. As a unique contribution, this paper establishes a space-time-processor network to illustrate the spatial and temporal LP scheduling and synchronization mechanism for microscopic transportation system simulation. The proposed parallel computing method is then implemented and applied to simulate four real-world transportation networks. The simulation results show significant speedup using our method and that that speedup magnitude is dependent on the scale of the network and the travel demand. In the examples provided, our method attained a speedup of up to 10.7 on a workstation with 32 computing CPU threads.

In our future work, we will perform further testing of the proposed parallel computing algorithm using a more efficient communication protocol for inter-process communication. We will further evaluate a wide range of other parallelization techniques, such as distributed shared memory-based methods and GPU computing technology, to further improve the computational efficiency of large-scale mesoscopic traffic simulations. The proposed space-time parallel framework is computationally efficient in traffic simulation, and we will also plan to extend this parallel computing method to other large and complex multi-agent simulation systems, e.g., pedestrian simulation (Qu et al., 2014; Qu et al, 2015) and urban rail transit networks, by considering boundedly rational route choice behavior (Liu and Zhou, 2016).

Acknowledgement

This paper is mainly supported by National Science Foundation – United States under Grant No. CMMI 1538105 "Collaborative Research: Improving Spatial Observability of Dynamic Traffic Systems through Active Mobile Sensor Networks and Crowdsourced Data", and U.S. Department of Energy's (DOE) Advanced Research Projects Agency – Energy (ARPA-E), "Traveler Response Architecture using Advanced Novel Signaling for Network Efficiency in Transportation (TRANSNET)". The second and corresponding author especially thanks Mr. Brian J Gardner at Federal Highway Administration's Office of Planning, for his encouragement in our development process of the open-source traffic simulation package DTALite. We also thank Jiangtao Liu and Dr. Taylor Li at Arizona State University, and Dr. Jinjin Tang at Beijing Jiaotong University, Jeff Taylor at University of Utah and Dr. Jeffrey Stempihar at Arizona State University for their useful comments and suggestions. The work presented in this paper remains the sole responsibility of the authors.

References

Adler, J. L., & Blue, V. J. (2002). A cooperative multi-agent transportation management and route guidance system. Transportation Research Part C: Emerging Technologies, 10(5), 433-454.

Auld, J., Hope, M., Ley, H., Sokolov, V., Xu, B., & Zhang, K. (2016). POLARIS: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations. Transportation Research Part C: Emerging Technologies, 64, 101-116.

Barceló, J, Ferrer, J.L, García D., Florian, M. and E. Le Saux, The Distributedization of AIMSUN2 Microscopic Simulator for ITS Applications, Proc. 3rd. World Congress on Intelligent Transport Systems, Orlando, 1996.

Ben-Akiva, M. E., Bierlaire, M., Burton, D., Koutsopoulos, H. N., & Mishalani, R. (2002). Network state estimation and prediction for real-time transportation management applications. Paper presented at the Transportation Research Board 81st Annual Meeting.

Bliemer, M.C.J., H.H. Versteegt and R.J. Castenmiller (2004) INDY: A New Analytical Multiclass Dynamic Traffic Assignment Model, Proceedings of the TRISTAN V conference, Guadeloupe.

Bryant, R. E.. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, MIT, 1977.

Cameron, G. D., & Duncan, G. I. (1996). PARAMICS—Parallel microscopic simulation of road traffic. The Journal of Supercomputing, 10(1), 25-53. K. Nagel and M. Schmidt. Parallel DYNEMO: Mesoscopic traffic flow simulation on large networks. preprint, 2000.

Celikoglu, H. B., & Dell'Orco, M. (2007). Mesoscopic simulation of a dynamic link loading process. Transportation Research Part C: Emerging Technologies, 15(5), 329-344.

Cetin N., Burri A., and Nagel K. 2003. A large-scale agent-based traffic microsimulation based on queue model. Swiss Transport Research Conference, Monte Verita, CH.

Chandy, K. M., & Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on software engineering, (5), 440-452.

Chandy, K. M., Sherman, R. Space-time and simulation. University of Southern California, Information Sciences Institute, 1989.

Chen, B., Cheng, H. H., & Palen, J. (2009). Integrating mobile agent technology with multi-agent systems for distributed traffic detection and management systems. Transportation Research Part C: Emerging Technologies, 17(1), 1-10.

Comfort, J. C. The simulation of a master-slave event set processor. Simulation, 1984, 42(3): 117-124.

Daganzo, C. F. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. Transportation Research Part B: Methodological, 1994, 28(4): 269-287.

Daganzo, C. F. (1995a). A finite difference approximation of the kinematic wave model of traffic flow. Transportation Research Part B: Methodological, 29(4): 261-276.

Daganzo, C. F. (1995b). The cell transmission model, part II: network traffic. Transportation Research Part B: Methodological, 29(2), 79-93.

Daganzo, C. F. In traffic flow, cellular automata= kinematic waves. Transportation Research Part B: Methodological, 2006, 40(5): 396-403.

Dagum, L., Enon, R. (1998). OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE, 5(1), 46-55.

Dell'Orco, M., Marinelli, M., & Silgu, M. A. (2016). Bee Colony Optimization for innovative travel time estimation, based on a mesoscopic traffic assignment model. Transportation Research Part C: Emerging Technologies, 66, 48-60.

Di Gangi, M., Cantarella, G. E., Di Pace, R., & Memoli, S. (2016). Network traffic control based on a mesoscopic dynamic flow model. Transportation Research Part C: Emerging Technologies, 66, 3-26.

Erdoğan, S., Zhou, X., & Liu, J. (2015). A Simplified Dynamic Traffic Assignment Framework for Statewide Traffic Modeling. In Transportation Research Board 94th Annual Meeting (No. 15-6090).

Ferscha, A., Tripathi, S. K. Parallel and distributed simulation of discrete event systems. 1998.

Florian, M. and Gendreau, M., (2001). Applications of parallel computing in transportation. Parallel Computing, 27(12), 1521-1522.

Fujimoto, R. M. Parallel discrete event simulation. Communications of the ACM, 1990, 33(10): 30-53.

Fujimoto, R. M. Parallel discrete event simulation: Will the field survive?. ORSA Journal on Computing, 1993, 5(3): 213-230.

Fujimoto, R. (2015). Parallel and distributed simulation. In Proceedings of the 2015 Winter Simulation Conference (pp. 45-59). IEEE Press.

Hunter, M., Kim, H. K., Suh, W., Fujimoto, R., Sirichoke, J., & Palekar, M. (2009). Ad hoc distributed dynamic data-driven simulations of surface transportation systems. Simulation, 85(4), 243-255.

Jafer, S., Liu, Q., & Wainer, G. (2013). Synchronization methods in parallel and distributed discrete-event simulation. Simulation Modelling Practice and Theory, 30, 54-73.

Jefferson, D. R.. Virtual time. ACM Transactions on Programming Languages and Systems, 7(3):404–245, July 1985.

Junchaya, T., & Chang, G. L. (1993). Exploring real-time traffic simulation with massively parallel computing architecture. Transportation Research Part C: Emerging Technologies, 1(1), 57-76.

Kallioras, N. A., Kepaptsoglou, K., & Lagaros, N. D. (2015). Transit stop inspection and maintenance scheduling: A GPU accelerated metaheuristics approach. Transportation Research Part C: Emerging Technologies, 55, 246-260.

Kirchner, A., & Schadschneider, A. (2002). Simulation of evacuation processes using a bionics-inspired cellular automaton model for pedestrian dynamics. Physica A: Statistical Mechanics and its Applications, 312(1), 260-276.

Lee, D. H., & Chandrasekar, P. (2002). A framework for parallel traffic simulation using multiple instancing of a simulation program. ITS Journal, 7(3-4), 279-294.

Liu, J. Parallel Discrete - Event Simulation[M]. John Wiley & Sons, Inc., 2009.

Liu, J. & Zhou, Z. (2016) Capacitated transit service network design with boundedly rational agents. Transportation Research Part B: Methodological,93, 225-250

Liu, Z., Meng, Q. (2013). Distributed computing approaches for large - scale probit - based stochastic user equilibrium problems. Journal of Advanced Transportation, 47(6), 553-571.

Lu, C.C., Mahmassani, H.S., Zhou, X., (2009). Equivalent gap function-based reformulation and solution algorithm for the dynamic user equilibrium problem. Transportation Research Part B: Methodological, 43(3), 345-364.

Lu, C. C., Zhou, X., & Zhang, K. (2013). Dynamic origin–destination demand flow estimation under congested traffic conditions. Transportation Research Part C: Emerging Technologies, 34, 16-37.

Mahmassani, H. S., Hu, T-Y, Peeta, S., and Ziliaskopoulos, A. Development and testing of dynamic traffic assignment and simulation procedures for ATIS/ATMS applications. Report DTFH61-90-R-00074-FG, U.S. DOT, Federal Highway Administration, McLean, Virgina, 1994.

Mahmassani, H. S. Dynamic network traffic assignment and simulation methodology for advanced system management application. Networks and Spatial Economics, 2001, Vol. 1, 267-292.

Morosan, C. D., Florian, M. (2015). The benefits of parallel computing for large-scale network equilibrium models. Downloaded from https://www.inrosoftware.com/assets/pres-pap/TRB2015/P15-6591.pdf.

Nagel, K., & Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. Journal de physique I, 2(12), 2221-2229.

Nagel, K., & Rickert, M. (2001). Parallel implementation of the TRANSIMS micro-simulation. Parallel Computing, 27(12), 1611-1639.

Nevers, Brandon L., et al. The Effective Integration of Analysis, Modeling, and Simulation Tools. No. FHWA-HRT-13-036. 2013.

Newell, G. F. (1993). A simplified theory of kinematic waves in highway traffic, part I: general theory. Transportation Research Part B: Methodological, 27(4), 281-287

Ng, M. W., Duc. T. Nguyen. (2015). Domain Decomposition, Parallel Computing and Traffic Assignment. Downloaded from http://amonline.trb.org/trb57535-2015-1.1793793/t006-1.1818822/116-1.1809928/p15-6590-1.1820620/p15-6590-1.1955721?qr=1

Nie, Y., Ma, J., & Zhang, H. M. (2008). A polymorphic dynamic network loading model. Computer‐Aided Civil and Infrastructure Engineering, 23(2), 86-103.

Panneton, F., L'ecuyer, P., & Matsumoto, M. (2006). Improved long-period generators based on linear recurrences modulo 2. ACM Transactions on Mathematical Software (TOMS), 32(1), 1-16.

Peacock, J. K., J. W. Wong, and E. G. Manning. Distributed Simulation using a Network of Processors. Computer Networks, Vol. 3, No. 1, pp. 44, 1979.

Peeta, S., & Ziliaskopoulos, A. K. (2001). Foundations of dynamic traffic assignment: The past, the present and the future. Networks and Spatial Economics, 1(3-4), 233-265.

Potuzak, T. Distributed-Parallel Road Traffic Simulator for Clusters of Multi-core Computers. in Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th International Symposium on. 2012.

Qu, Y., Gao, Z., Xiao, Y., Li, X., (2014). Modeling the pedestrian’s movement and simulating evacuation dynamics on stairs. Safety science, 70, 189-201.

Qu, Y., Gao, Z., Orenstein, P., Long, J. and Li, X., (2015). An effective algorithm to simulate pedestrian flow using the heuristic force-based model. Transportmetrica B: transport dynamics, 3(1), 1-26.

Ruan, J. M., Liu, B., Wei, H., Qu, Y., Zhu, N., & Zhou, X. (2016). How Many and Where to Locate Parking Lots? A Space–time Accessibility-Maximization Modeling Framework for Special Event Traffic Management. Urban Rail Transit, 1-12.

Snelder, M. (2009). A comparison between dynameq and indy. CIRRELT.

Sundaram, S., Koutsopoulos, H.N., Ben-Akiva, M., Antoniou, C., Balakrishna, R., 2011. Simulation-based dynamic traffic assignment for short-term planning applications. Simulation Modelling Practice and Theory 19(1), 450-462.

Wong, S. C. (1997). Group-based optimisation of signal timings using parallel computing. Transportation Research Part C: Emerging Technologies,5(2), 123-139.

Yperman, I. The link transmission model for dynamic network loading. 2007.

Ziliaskopoulos, A., Kotzinos, D., & Mahmassani, H. S. (1997). Design and implementation of parallel time-dependent least time path algorithms for intelligent transportation systems applications. Transportation Research Part C: Emerging Technologies, 5(2), 95-107.

Zhen, S., W. Kai, and Z. Fenghua. Agent-based traffic simulation and traffic signal timing optimization with GPU. in Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on. 2011.

Zhou, X., Taylor, J. (2014). DTALite: A queue-based mesoscopic traffic simulator for fast model evaluation and calibration. Cogent Engineering, 1(1): 961345.

Zhou, X., Zlatkovic, M., Farhan, M. (2015). Simplified web-based decision support method for traffic management and work zone analysis. Utah Department of Transportation, Report No. UT-15.09.

Zhou, X., Tanvir, S., Lei, H., Taylor, J., Liu, B., Rouphail, N. M., & Frey, H. C. (2015). Integrating a simplified emission estimation model and mesoscopic dynamic traffic simulator to efficiently evaluate emission impacts of traffic management strategies. Transportation Research Part D: Transport and Environment, 37, 123-136.

Appendix: Generic Queue-based Traffic Flow Model

This appendix offers a brief problem statement for a generic queue-based mesoscopic traffic simulation. In our study, we select Newell's simplified Kinematic Wave model (Newell, 1993) as an example of possible queue-based traffic flow models. Newell's KW model can reproduce some realistic phenomena, such as queue spillback, congestion spreading, and it has been implemented by several research groups (Yperman, 2007; Zhou et al., 2014 (DTALite)). The following notation is used to describe Newell's kinematic wave model and general dynamic network loading algorithms.

t	index of simulation time intervals
ΔT	the length of a mesoscopic simulation time interval in KW model (e.g. 6 seconds)
v_f	free-flow speed (or forward wave speed)
w	constant backward wave speed
k_{jam}	jam density
l	index of links, $l = 1, 2, \ldots, L$
n	index of nodes, $n = 1, 2, \ldots, N$
$\Delta X(l)$	length of link l, its unit lane-miles to consider the number of lanes on link l
$A(l,t)$	cumulative number of arriving vehicles ready to move into link l at time t
$V(l,t)$	cumulative number of departing vehicles waiting at the vertical queue of link l at time t
$D(l,t)$	cumulative actual number of vehicles departing/moving out of link l at time t
$q^{\max}(l,t)$	maximum flow rate on link l between time t-ΔT and time t
$cap^{in}(l,t)$	inflow capacity of link l between time t-ΔT and time t
$cap^{out}(l,t)$	outflow capacity of link l between time t-ΔT and time t
$FFTT(l)$	free-flow travel time on link l; i.e., length(l)/v_f
$BWTT(l)$	backward wave travel time on link l; i.e., length(l)/w

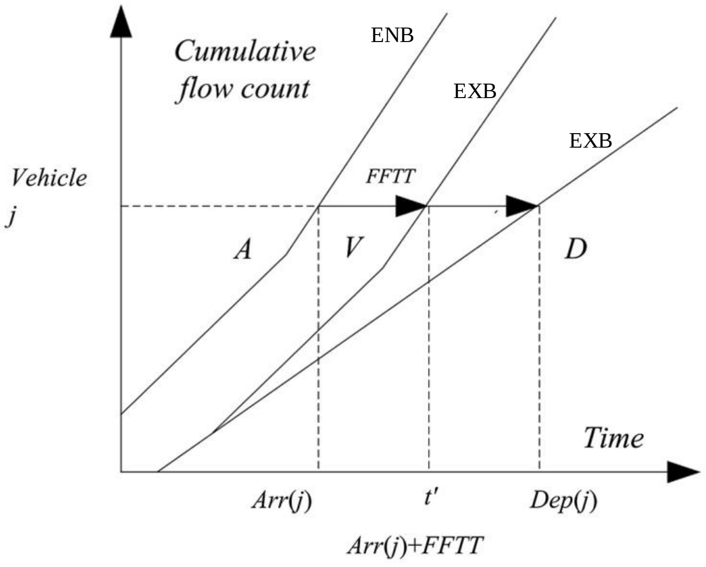


Fig. A1 Event-based simulation process

Fig. A1 illustrates the event-based simulation process by the mesoscopic KW model. For a single link, the curves *A*, *D* and *V* represent the cumulative arrival flow count at the ENB (entrance buffer), the departure flow counts at the EXB (exit buffer), and the cumulative flow count of the vertical queue at the EXB (exit buffer), respectively. *FFTT* is the free-flow travel time of the link.

In Newell's KW model, the cumulative counts are used to represent forward and backward waves, and the change of cumulative curves along a characteristic line at links are described as Eq. (A1):

$$dN(x,t) = \frac{\partial N}{\partial x}dx + \frac{\partial N}{\partial t}dt = qdt - kdx \qquad \text{(A1)}$$

Where x is the location along the corridor, and $N(x,t)$ is the actual cumulative count of vehicles that pass location x from time 0 to time t. Interested readers are suggested to read the paper (Lu et al., 2013) on the details of queue spillback modeling, which can be represented in Eq. (A2).

$$D(l, t - BWTT) - A(l,t) = -k_{jam}(l) \times \Delta x(l) \qquad \text{(A2)}$$

That is, the cumulative flow count difference between two ends along a backward wave has a congested time-space “mass” of $k_{jam}\ (l) \times \Delta x(l)$ vehicles. Once a queue spillback occurs (i.e. Eq. (A2) holds), the inflow capacity will be constrained by not only the maximum flow rates, but also the outflow rates at the downstream node.

$$cap^{in}(l,t) = \min\{q^{\max}(l,t), A^{\max}(l,t) - A(l,t - \mathit{\Delta T})\} \tag{A3}$$