Reducing Synchronization Overhead with Computation Replication in Parallel Agent-Based Road Traffic Simulation

Yadong Xu, *Member, IEEE*, Vaisagh Viswanathan, *Member, IEEE*, and Wentong Cai, *Member, IEEE*

Abstract—Road traffic simulation is a useful tool for studying road traffic and evaluating solutions to traffic problems. Large-scale agent-based road traffic simulation is computationally intensive, which triggers the need for conducting parallel simulation. This paper deals with the synchronization problem in parallel agent-based road traffic simulation to reduce the overall simulation execution time. We aim to reduce synchronization operations by introducing some redundant computation to the simulation. There is a trade-off between the benefit of reduced synchronization operations and the overhead of redundant computation. The challenge is to minimize the total overhead of redundant computation and synchronization. First, to determine the amount of redundant computation, we proposed a way to define extended layers of partitions in the road network. The sizes of extended layers are determined by the behavior of agents and the topology of road networks. Second, due to the dynamic nature of road traffic, a heuristic was proposed to adjust the amount of redundant computation according to traffic conditions during simulation run-time to minimize the overall simulation execution time. The efficiency of the proposed method was investigated in a parallel agent-based road traffic simulator using real-world network and trip data. Results have shown that the method can reduce synchronization overhead and improve the overall performance of the parallel simulation significantly.

Index Terms—Agent-based traffic simulation, parallel simulation, conservative synchronization, computation replication

1 Introduction

Agent-based road traffic simulation considers the behavior of driver-vehicle-units (DVUs) [1]. Large-scale agent-based simulation of road traffic (e.g., the whole city) is a useful tool to evaluate the impact of individual behaviors on road traffic as a whole [2]. It is useful in solving the severe problems that modern large cities face, such as congestion and high emissions. However, such traffic simulation usually involves thousands or millions of agents, which is computationally intensive. Parallel computing techniques can be used to speed-up the simulation.

To parallelize an agent-based traffic simulation, a common way is to decompose the road network into multiple spatial subregions (i.e., partitions). Each subregion is executed by a Logical Process (LP) which is assigned to a physical processing unit. To maintain the correctness of the simulation, *synchronization* of LPs is required for simulation time advancement due to data dependencies between LPs [3]. In a distributed memory environment, synchronization is typically achieved by message-passing. For agent-based traffic simulation, global barriers are commonly used [4], [5]. Agent models are updated with fixed intervals and global barriers are deployed at the end of update intervals. The limitation of this synchronization method is that all LPs have to wait at global barriers despite some LPs having no dependencies with other LPs. Another approach for synchronization is to allow LPs to exchange messages and progress *asynchronously* [6], [7]. LPs do not need to synchronize at the same time. The frequencies of synchronization can be different for different LP pairs and is determined by a measure termed *lookahead*. Lookahead of LP_i towards LP_j $(i \neq j)$ at simulation time t is a time interval in the simulated future within which LP_i will not have data dependencies with LP_j. The larger lookahead values are, the less frequent synchronization is performed. However, due to the frequent interaction of agents, agent-based simulations generally have small lookahead. High synchronization overhead is still an issue for the performance of parallel agent-based traffic simulations.

Computation replication is an effective approach reported in the literature to reduce inter-process communication in parallel applications [8], [9], [10], [11], [12]. The concept is to let LPs conduct some redundant computation to generate data locally instead of receiving them from synchronization. There is a trade-off between the benefit of reduced synchronization and the overhead of redundant computation: to further reduce synchronization operations, more redundant computation is usually required. This method has been used for solving partial differential equations and matrix multiplication [8], [9], as well as agent-based simulations [10], [11],

- *Y. Xu is with AIDA, TUMCREATE Ltd., Singapore 138602. E-mail: xuya0006@e.ntu.edu.sg.*
- *V. Viswanathan is with RP5, TUMCREATE Ltd., Singapore 138602. E-mail: vaisagh.viswanathan@tum-create.edu.sg.*
- *W. Cai is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798. E-mail: aswtcai@ntu.edu.sg.*

Manuscript received 25 Sept. 2016; revised 16 Apr. 2017; accepted 26 May 2017. Date of publication 9 June 2017; date of current version 11 Oct. 2017. (Corresponding author: Yadong Xu.)
Recommended for acceptance by C. Carothers.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2714165

[12]. However, there is a major difference between the simulation spaces in those simulations and those in agent-based road traffic simulation. The spaces in the existing works are either n-dimensional grids where computational tasks are distributed uniformly, or graphs where vertices represent computational tasks and edges represent the dependencies of the tasks. The simulation space in agent-based road traffic simulation is a spatial network composed of links and nodes. Agents are situated in the spatial network. The distribution of agents in the spatial network may not be uniform and it dynamically changes since agents move along the links. The interaction of agents depends on the positions of agents and is dynamic. The interaction of agents affects how computation is replicated. Thus, the existing solutions cannot be directly applied to agent-based road traffic simulation. It is non-trivial to determine how computation can be replicated in agent-based road traffic simulation. In addition, the trade-off between the redundant computation and the benefit of reduced synchronization operations should be carefully studied in order to gain overall performance improvement.

We solve two challenges in this paper. The first challenge is to limit the amount of redundant computation. We propose a way to define extended layers of partitions for agent-based road traffic simulation. The sizes of extended layers are determined by the behavior of agents and topology of road networks in our representation. The second challenge is to deal with the dynamic nature of road traffic. A method is proposed to adjust the amount of redundant computation according to traffic conditions on the road network dynamically during the simulation. Efficiency of the proposed methods is investigated in a parallel agent-based road traffic simulator using real-world network and trip data.

The remainder of the paper is organized as follows: the next section presents some background information about parallel agent-based traffic simulation used in this work: agent models, partitioning of the simulation, and the synchronization protocol. Section 3 introduces our proposed adaptive computation replication method. Extended layers of partitions on the road network are defined. A heuristic that adjusts the number of extended layers to replicate is developed. Subsequently, Section 4 presents experiments and results. Section 5 describes related works. Finally, Section 6 provides a summary of this work and recommendations for future work.

2 Parallel Agent-Based Simulation of Road Traffic

2.1 Simulation Space and Agents

2.1.1 Simulation Space

The simulation space of an agent-based traffic simulation is a road network. It is a *spatial network* that consists of links and nodes. Links represent real-world roads and can have one or more lanes. Links have speed limits. Nodes contain the connectivity information of links. The traffic flow on a link is unidirectional from the *start node* to the *end node*.

2.1.2 Agents

Agents are situated on roads (i.e., links) of the network. An agent in the simulation represents a DVU. The behavior of DVUs is often characterized by acceleration models and lane-changing models [1], [4], [13]. The models describe the movement of DVUs on roads, such as what acceleration a DVU should have and which lane to take. The models require agents to have *sensing ranges*, which are the areas in the road network within which other agents may affect the agent's behavior. An agent needs to examine the traffic condition within its sensing range to make acceleration and lane-changing decisions. This is challenging in parallel traffic simulation when the sensing range is reaching into other partitions as it then potentially requires synchronization between the responsible LPs.

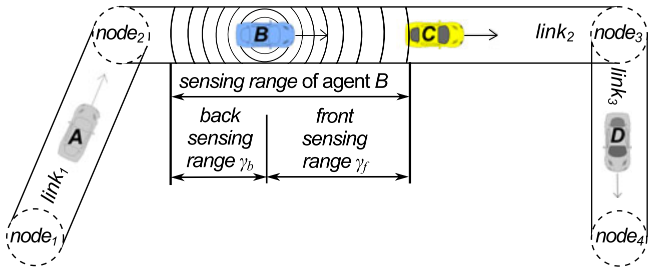


Fig. 1. Agent B with front sensing range γ_f and back sensing range γ_b in a road network. Agent C is in the sensing range of agent B; thus, agent B subscribes to agent-based state variables of agent C.

2.1.3 Agent State Variables

An agent has a *state* at a certain virtual simulation time. The state contains multiple state variables. Among them, there are *agent-based* and *component-based* state variables. Agent-based state variables belong to the entire agent and are visible to other agents, such as velocity and position. Component-based state variables belong to the models in an agent, such as a state-of-charge variable for a battery model in an electric vehicle. An agent subscribes to agent-based state variables of the agents in its sensing range. States of agents change as the simulation progresses by executing timestamped *events* which contain certain update functions. Agent-based state variables are updated periodically. The period is referred to as an *update interval*, denoted as δ. The events that change agent-based state variables may have an effect on other LPs, thus they affect synchronisation between LPs. Other events that change component-based state variables are internal to an LP. An illustration of sensing ranges and state subscription is shown in Fig. 1.

2.2 Parallelization and Data Dependencies

We denote the entire road network as G, and agent population at simulation time t as A^t. In parallel simulation, the road network is partitioned into I disjoint spatial subregions, $\mathbb{G} = \{G_0, G_1, \ldots, G_{I-1}\}$. The subset of A^t that resides in partition G_i $(0 \leq i < I)$ at simulation time t are denoted as A_i^t. By definition, $A^t = \cup_{i=0}^{I-1} A_i^t$. The LP that is responsible for executing the events from agents in partition i is LP_i. Agents in partition i are *local* to LP_i.

During partitioning, the network is cut on links. The links that are cut and therefore evenly divided between two partitions are named *boundary links*. A boundary link can be *incoming* to or *outgoing* from a partition, depending on the traffic flow on the boundary link. The two partitions that share boundary links are *neighboring partitions*.

There are data read and write *dependencies* between neighboring LPs. First, if there is an agent A in LP_i inside the

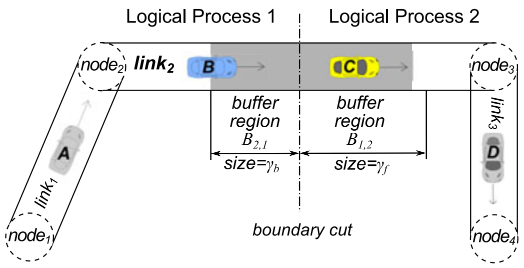


Fig. 2. Illustration of boundary cut and buffer regions. γ_f and γ_b are front and back sensing ranges, respectively.

sensing range of another agent B in LP_j, agent B should be aware of the agent-based state variables of agent A. To achieve this, a *proxy agent* is created in LP_j that mirrors agent A. It possesses exactly the same agent-based state variables as agent A. Hence, the agent-based state variables of agent A should be sent by LP_i to LP_j to keep the state of the proxy agent updated. In this case, there is a data *read dependency* between LP_i and LP_j. The agent-based state variables of agent A are *shared states*. Second, during the simulation, when an agent on a boundary link moves beyond the boundary of partition i and enters the area of partition j ($i \neq j$), the agent *migrates* from LP_i to LP_j. Migrated agents are destroyed in the original LP and recreated with all their state variables in the new LP. The migration of agents incurs a data *write dependency* between the two LPs. To identify those agents that need to share their states, *buffer regions* of partitions are defined. They are the regions at the boundary of partitions with sizes equal to the sensing ranges of agents. If an agent falls inside a buffer region, it is possible that the agent is in the sensing range of some agents in the neighboring LP.

To illustrate the concepts above, an example is shown in Fig. 2. $link_2$ is a boundary link. The left half is part of G_1, and the right half is part of G_2. The direction of traffic on $link_2$ is from G_1 to G_2, thus $link_2$ is an outgoing boundary link of G_1, and an incoming boundary link of G_2. LP_1 and LP_2 are neighboring LPs. $B_{1,2}$ is the buffer region for G_1 in G_2, and $B_{2,1}$ is the buffer region for G_2 in G_1. Given the traffic direction of the link, the lengths of $B_{1,2}$ and $B_{2,1}$ are equal to the front and back sensing ranges of agents, respectively. Agent C in LP_2 is in buffer region $B_{1,2}$. Therefore, there is a data read dependency between LP_1 and LP_2. LP_2 should send the agent-based state variables of agent C to LP_1. There is a proxy agent in LP_1 mirroring agent C. If agent B continues moves into G_2, LP_2 will be responsible for executing events of agent B. Thus agent B is removed from LP_1 and created anew in LP_2. To do so, the complete information about agent B, including all state variables and all model parameters, is sent from LP_1 to LP_2. There is a data write dependency between LP_1 and LP_2.

2.3 Mutual Appointment Synchronization Protocol

Data dependencies necessitate synchronization of LPs. The synchronization protocol used in this work is the mutual appointment (MA) protocol introduced in [14]. The idea of the protocol is that an LP communicates with other LPs by making *appointments* individually with them at certain *mutually agreed* simulation times.

The progression of the simulation in LP_i using the MA protocol is shown in Algorithm 1. For each update interval, there is a synchronization event. Associated with the synchronization event, there is a set of LPs that currently have appointments with LP_i, denoted as C_i^t. C_i^t may include all, none, or only a subset of the neighboring LPs of LP_i. When C_i^t is empty, no message-passing occurs for LP_i at time t. For each LP_j in the set C_i^t, LP_i sends and receives migrating agents, shared states and a *lookahead*. Lookahead is a predicted time period from the current time to the time when the next data dependency may happen. After messages are received, the next appointment is made according to the lookahead. An appointment is made by adding the LP to the future $C_i^{t+\Delta t}$ set, where Δt is the minimum of the two lookahead values of LP_i and LP_j. According to the definitions of data read and write dependencies, the minimum lookahead value between any two LPs is an update interval (i.e., δ). Lookahead decides the frequency of synchronization. In this work, the lookahead is determined by the number of replicated extended layers.

Algorithm 1. Simulation Progression in LP_i Using MA Protocol

```
1:  Definitions:
2:  $T_{end}$ simulation ending time
3:  $C_i^t$ LPs having appointments with $LP_i$ at simulation time $t$
4:  $l_{i,j}^t$ lookahead from $LP_i$ to $LP_j$ at simulation time $t$
5:  initialize $t \leftarrow 0$, $C_i^0$ as all neighboring LPs of $LP_i$;
6:  while $t < T_{end}$ do
      // synchronization event
7:    foreach $LP_j \in C_i^t$ do
8:      send migrating agents, shared states, and current look-
        ahead (i.e., $l_{i,j}^t$) to $LP_j$;
9:      prepare to receive a message from $LP_j$;
10:   end
11:   wait for all message sending and receiving to finish;
12:   update the local agent set and proxy agent set;
13:   foreach $LP_j \in C_i^t$ do
14:     add $LP_j$ to $C_i^{t+\Delta t}$, where $\Delta t = \min(l_{i,j}^t, ; l_{j,i}^t)$;
15:   end
      // event for updating agent-based states
16:   update the states of local agents for this update interval;
17:   $t \leftarrow t + \delta$;
18: end
```

Another consideration for efficient parallel simulation is workload balance of LPs. Agents should be distributed as evenly as possible among LPs. In this work, we focus on synchronization, thus load-balancing will not be discussed. More detail regarding load-balancing in traffic simulation, one can refer to [15].

3 Adaptive Computation Replication

The aim of this work is to reduce synchronization so as to reduce simulation execution time. This is achieved by increasing lookahead via copying more information from neighboring LPs at each synchronization operation. Interactions of agents are bounded by their locations on the road network; thus, we determine the extra information based on agent locations on the road network. Extended layers of partitions are defined in this section. Then an adaptive method that dynamically adjusts the number of extended layers to replicate is proposed.

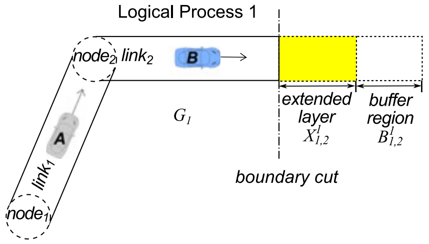


Fig. 3. Illustration of extended layers and new buffer regions: The view of the network from LP_1.

3.1 Extended Layers on the Network

We first explain the idea with the simplest case, i.e., one extended layer. Then the generalized multiple layer case is described.

3.1.1 One Extended Layer

An *extended layer* of a partition is defined as the space in the road network immediately next to the boundary of the partition that is required to calculate the agent states in an LP until the next synchronization. Buffer regions are shifted next to the extended layer.

When synchronization is conducted, each LP receives complete agents in its extended layer and shared states in buffer regions from neighboring LPs. Agents in the extended layer still exist in the original LPs, but they are replicated in the receiving LP. They are referred to as *external agents* of the receiving LP. LPs compute the states of external agents, so as to emulate the receive of a synchronization message by producing migrated agent and shard states. This way, the synchronization operation can be skipped.

The concepts above are illustrated in Fig. 3. The network is the same as that in Fig. 2. Partition G_1 has one extended layer in partition G_2, which is marked as $X^1_{1,2}$. $B^1_{1,2}$ is the new buffer region of G_1 inside G_2.

Suppose that at simulation time t, complete agents in $X^1_{1,2}$ are replicated to LP_1 from LP_2. Agent-based state variables of agents in $B^1_{1,2}$ are send to LP_1 by LP_2. The agents in $X^1_{1,2}$ are external agents of LP_1. States of those agents are updated by LP_1 together with local agents at time t. At time $t+\delta$, the agents hat fall inside $X^1_{1,2}$ will function as proxy agents. Due to movement of agents, agents currently in $X^1_{1,2}$ may be different from those at time t. Then, states of the agents currently in G_1 can be updated till time $t+2\delta$ using those agents. After that, the next synchronization operation is performed. The same procedure also applies to LP_2. Lookahead between the two LPs is 2δ.

3.1.2 Multiple Layers

More generally, a partition can have multiple layers of extended layers. The first layer is immediately next to the boundary of the partition, and other layers expand towards neighboring partitions. Buffer regions are shifted next to the outermost extended layer.

Supposing that LP_i and LP_j use $\hat{k}$ ($\hat{k} \geq 1$) extended layers between them, the lookahead between the two LPs will be $(\hat{k}+1)\cdot\delta$. To explain this, we denote the kth extended layer of partition G_i in neighbor G_j as $X^k_{i,j}$. After a synchronization operation is performed, in the mth ($1 \leq m \leq \hat{k}$) update interval, LP_i updates the states of local agents and external agents in the extended layers $\{X^1_{i,j}, X^2_{i,j}, \ldots, X^{\hat{k}-(m-1)}_{i,j}\}$. After the mth update interval, agents in the extended layer $X^{\hat{k}-(m-1)}_{i,j}$ will function as proxy agents. In the $(\hat{k}+1)$th update interval, LP_i only updates the states of its local agents. After that, another synchronization is required. A similar procedure is conducted by LP_j. The computation of external agent states is *redundant computation*, as they are computed by both LPs.

3.1.3 Requirements for Agent Models

Since the computation of external agents is replicated in LPs, agent models should satisfy the following requirements: i) given the same input values, models of agents always produce the same agent states, including agent-based and component-based state variables; and ii) the order of the agents being updated should not affect the result. Otherwise, replicas may generate different states. This requirement is met for deterministic models. For stochastic models, this can be achieved by manipulating the seeds of the random number generators in the two replicating LPs to produce the same random number sequence.

3.2 Sizes of Extended Layers

3.2.1 Representation of Extended Layers

As described before, a road network is represented as a collection of links and nodes, and agents are situated on links. Sensing ranges of agents usually do not cover entire links, thus an extended layer may only cover a portion of links. Therefore, we represent an extended layer as a collection of link segments. A *link segment* is a portion of a link between two points on the link. A link segment can be uniquely identified by specifying the link id that the segment is on and displacements of the two points. A *displacement* of a point on a link is the distance between that point and the start node of the link. Thus, a link segment can be denoted as $seg_{(id,s,e)}$, where id is the id of the link, s is the starting displacement of the segment on the link, e is the ending displacement. The length of this segment is $|e-s|$.

3.2.2 Sizes of Link Segments

To determine the sizes of extended layers, we need to analyze the behavior of agents and topology of the road network, because front and back sensing ranges of agents may be different and road links can have different lengths and connectivity too.

The agents in the kth extended layer should allow the agents in the $(k-1)$th layer (or local agents if k=1) to advance one update interval. Therefore, the kth extended layer must cover the sensing ranges of the agents in the $(k-1)$th layer (or those in the local area if k=1). Meanwhile, the movement of agents must also be considered. Different sizes of segments depending on the *direction of extension* of extended layers are illustrated in Fig. 4. The direction of extension is from the $(k-1)$th layer to the kth layer (or from the local area to the first layer if k=1). It can be the same as or opposite to the traffic direction of a link.

In Fig. 4, $X^k_{i,j}$ is in the upstream direction of $X^{k-1}_{i,j}$, i.e., the direction of extension for $X^{k-1}_{i,j}$ is opposite to the traffic direction. The segment of $X^k_{i,j}$ consists of two parts: the first part of size γ_b and the second part of size $V_l\cdot\delta$, where V_l is the speed limit of link l. The second part, which is marked

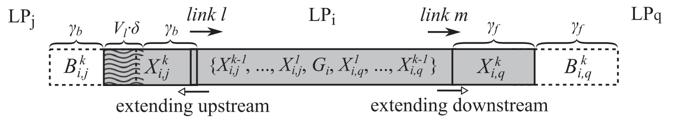


Fig. 4. Different lengths of link segments in extended layers of G_i in neighboring partitions G_j and G_q. $X^k_{i,j}$ and $X^k_{i,q}$ are on link l and m, respectively. The solid arrows indicate the traffic directions on links. The empty arrows indicate the directions of extension of extended layers. The direction of extension of $X^k_{i,j}$ is opposite to the traffic direction of link l, i.e, extending upstream; the direction of extension of $X^k_{i,q}$ is the same as the traffic direction of link m, i.e., extending downstream.

with a wavy pattern, ensures that after agents in $X^k_{i,j}$ are updated, the first part of $X^k_{i,j}$ contains all the agents required for the next round of state update for $X^{k-1}_{i,j}$. Its size is the maximum distance that agents can travel on link l in one update interval. In contrast, for $X^{k-1}_{i,q}$, the direction of extension is the same as the traffic direction, i.e., $X^k_{i,q}$ is in the downstream direction of $X^{k-1}_{i,q}$. The segment of $X^k_{i,q}$ only requires a size of γ_f, because after an update, $X^k_{i,q}$ will still contain all the agents needed to update $X^{k-1}_{i,q}$ for the next update interval.

To explain what happens to the second part of $X^k_{i,j}$ during the simulation, an example with two extended layers is depicted in Fig. 5. Supposing a synchronization operation is performed at time 0, LP_2 receives complete agents in two layers of extended layers and shared states in the buffer regions from LP_1 and LP_3. As shown in Fig. 5a, states of the agents in $X^2_{2,1}$, $X^1_{2,1}$, G_2, $X^1_{2,3}$, and $X^2_{2,3}$ (i.e., the dark gray region) are updated by LP_2. Fig. 5b shows that at time δ, $X^2_{2,3}$ and the first part of $X^2_{2,1}$ function as buffer regions. The second part of $X^2_{2,1}$ (marked with a cross) is dropped since agent states in that region may not be accurate (agents in $B^2_{2,1}$ may move into the region). Agents in $X^1_{2,1}$, G_2, and $X^1_{2,3}$ are updated. As the simulation progresses to time 2δ as shown in Fig. 5c, $X^1_{2,3}$ and the first part of $X^1_{2,1}$ function as buffer regions. Similar to the previous update interval, the region marked with a cross in $X^1_{2,1}$ is dropped. Agents in G_2 will be updated. It can be noticed that we have updated agents in G_2 for three update intervals with one synchronization.

3.2.3 Segments on Connecting Links

When a link is not long enough to cover an extended layer, the layer will expand to its connecting links. Link segments on the connecting links may have various sizes depending on how they connect to the link, and how agents sense the connecting links. In our network representation, there are four basic cases of how a link x can connect to another link y: i) they share the same start node; ii) they share the same end node; iii) the start node of link x is the end node of the link y; and iv) the end node of link x is the start node of link y. Sensing ranges of agents can cover connecting links.

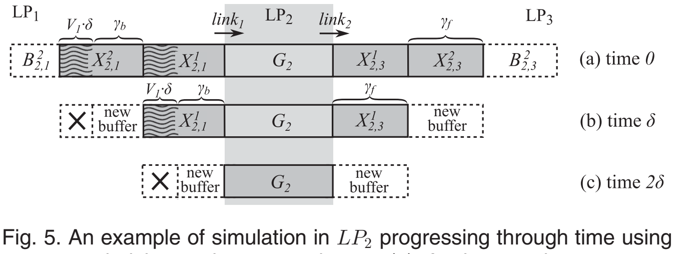


Fig. 5. An example of simulation in LP_2 progressing through time using two extended layers in LP_1 and LP_3: (a) At time 0, there are two extended layers for G_2 in G_1 and G_3. (b) At time δ, $X^2_{2,1}$ and $X^2_{2,3}$, excluding the crossed region, function as buffer regions. (c) At time 2δ, $X^1_{2,1}$ and $X^1_{2,3}$, excluding the crossed region, function as buffer regions.

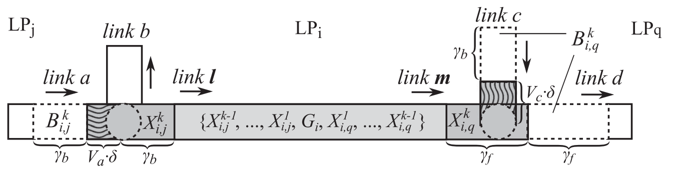


Fig. 6. Four basic cases of extended layer segments on connecting links, illustrated by links a, b, c, and d. The kth extended layers of G_i in G_j and G_q, i.e., $X^k_{i,j}$ and $X^k_{i,q}$, have extra segments on link a and link c respectively (marked by a wavy pattern).

The segment sizes for the four basic cases are illustrated in Fig. 6. Link a has the start node of link l as its end node. Link b and link l share the same start node. Link c and link m share the same end node. Link d has the end node of link m as its start node. $X^k_{i,j}$ contains an extra segment on link a, and $X^k_{i,q}$ contains an extra segment on link c. The extra segments are required so that the buffer regions for the next round of state update will contain all necessary proxy agents, which is for similar reasons explained in Fig. 4.

A real-world road network usually has a large number of links and very complex connectivity. The four cases described above can be arbitrarily combined. Depending on the topology of the road network and sizes of partitions, one extended layer can contain many segments. A searching algorithm is designed to determine the segments of extended layers in the next section.

3.2.4 Searching Algorithm for Determining Extended Layers

Extended layers of G_i inside G_j are calculated by LP_j. A search is performed by LP_j starting from all the boundary links between G_i and G_j towards the inner area of G_j. The searching algorithm is shown in Algorithm 2.

The algorithm starts with initializing the layer index and a flag that controls the termination of the algorithm. A set S_{add} is used to store the segments in the current layer. A set S_{cont} is used to store the segments in S_{add} from which the next layer continues. Note that some segments in S_{add} may not connect to the next layer (a detailed description is provided in the appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2017.2714165). A set S_{check} is used to keep track of all the segments that have been added to extended layers. Areas occupied by the segments in S_{check} will not be searched again.

The first extended layer is obtained by adding segments from boundary links (lines 12-17). It is assumed that boundary links are long enough to accommodate the segments of the first layer. Subsequently, the algorithm continues to search for segments for the next layer based on the current layer (*while* loop in lines 18-35). Each iteration of the *while* loop determines one layer of extended layers. In each iteration, segments in S_{add} are added to the current layer. Segments in S_{cont} are added to a temporary set S_{temp_cont}. Then, S_{add} and S_{cont} are emptied and reused to store new segments for the next layer. After that, the algorithm searches for segments in the next layer connecting to the segments in S_{temp_cont}. If $s < e$, the search continues towards the downstream direction of the link, using function $searchDownstream$. If $e < s$, the search continues towards the upstream direction of the link, using function $searchUpstream$. (For more details of functions $searchDownstream$ and $searchUpstream$ one can refer to Algorithms 3 and 4 in the appendix, available in the online supplemental material). In functions $searchDownstream$ and $searchUpstream$, when the remaining space on a link is not enough for a layer, the searching will expand to the connecting links.

Algorithm 2. Searching for Link Segments of Extended Layers of G_i Inside G_j by LP_j

```
1:  Definitions:
2:  $k$            index of the extended layer $X^k_{i,j}$
3:  $I_{j,i}$      set of incoming boundary links of $G_j$ from $G_i$
4:  $O_{j,i}$      set of outgoing boundary links of $G_j$ towards $G_i$
5:  $S_{add}$      candidate segments to be added to an extended layer
6:  $S_{cont}$     segments in $S_{add}$ from which the next layer continues
7:  $S_{check}$    all segments that are already inside extended layers
8:  $L_l$          length of link $l$
9:  $V_l$          speed-limit of the traffic on link $l$
10: $\zeta$        a flag that indicates if the searching should continue
11: initialize $k \leftarrow 1$, $\zeta \leftarrow true$;
12: foreach link $l \in I_{j,i}$ do
13:     put segment $seg_{(l, \frac{L_l}{2}, \frac{L_l}{2} + \gamma_f)}$ into $S_{add}$, $S_{cont}$ and $S_{check}$;
14: end
15: foreach link $l \in O_{j,i}$ do
16:     put segment $seg_{(l, \frac{L_l}{2}, \frac{L_l}{2} - \gamma_b - V_l \cdot \delta)}$ into $S_{add}$, $S_{cont}$ and $S_{check}$;
17: end
    // search layer by layer
18: while $\zeta$ do
19:     foreach $seg_{(id,s,e)} \in S_{add}$ do
20:         put $seg_{(id,s,e)}$ into layer $X^k_{i,j}$;
21:     end
22:     $S_{temp\_cont} \leftarrow S_{cont}$;
23:     $S_{add} \leftarrow \varnothing$, $S_{cont} \leftarrow \varnothing$;
        // search segments for the next layer
24:     foreach $seg_{(id,s,e)} \in S_{temp\_cont}$ do
25:         if $\zeta \wedge (s < e)$ then
                // direction of extension is downstream
26:             $searchDownstream(id, e, \gamma_f)$;
27:         else if $\zeta \wedge (e < s)$ then
                // direction of extension is upstream
28:             $searchUpstream(id, e, \gamma_b + V_{id} \cdot \delta)$;
29:         end
30:     end
31:     if $S_{add} = \varnothing$ then
32:     $\zeta \leftarrow false$; // whole $G_j$ has been searched
33:     end
34:     $k \leftarrow k + 1$;
35: end
```

The searching terminates if the whole area of G_j has been searched (Algorithm 2 line 32) or the extension reaches a third partition. In other words, extended layers of G_i inside G_j is restricted within G_j only. Otherwise, if they were extended to a third partition G_q, LP_q would also need to communicate with LP_i for synchronization between LP_i and LP_j. This may introduce extra communication which will downgrade the benefit of computation replication.

Algorithm 2 is executed after partitions are determined. If partitions change dynamically during the simulation, extended layers need to be recalculated. The time and space complexity of Algorithm 2 is $O(|Seg|)$, where $|Seg|$ is the number of segments in extended layers in the calculating LP. This is because the algorithm searches the links within the current partition only once. The time for searching and adding segments in the *while* loop and *for* loop, and the memory required to store the segments, are both proportional to the number of segments in the partition.

TABLE 1
Notation Used in the Analysis

Notation	Description
$\lvert A^k_{i,j}\rvert$	number of agents in extended layer $X^k_{i,j}$
$K_{i,j}$	available extended layers between G_i and G_j
$K^m_{i,j}$	adaptive range of extended layers between LP_i and LP_j in the mth evaluation period
$\lvert M_{i,j}\rvert$	number of migrating agents from LP_i to LP_j in one synchronization
$\lvert S^k_{i,j}\rvert$	number of shared states that LP_i sends to LP_j (i.e., in $B^k_{j,i}$) in one synchronization
Ta	computational workload of one agent in one update interval in terms of wall-clock time
bw	available bandwidth for message-passing
$\hat{k}_{i,j}$	obtainable optimum number of extended layers to replicate between LP_i and LP_j ($\hat{k}_{i,j} = \hat{k}_{j,i}$ since MA protocol is used)
$\hat{k}^m_{i,j}$	$\hat{k}_{i,j}$ for the mth evaluation period
$o_{i,j}(k)$	total overhead for LP_i, relative to LP_j in one synchronization cycle, when k layers are replicated, including redundant computation and message-passing overhead
$o_{i,j}(\tau, k)$	total overhead for LP_i, relative to LP_j during time period τ when k layers are replicated
$s_{i,j}(k)$	overhead of sending a synchronization message from LP_j to LP_i when k layers are replicated
srl	send and receive latency of sending a message
$\Gamma_{i,j}(k)$	redundant computation between two consecutive synchronization operations in LP_i due to external agents from LP_j, when k layers are replicated

3.3 Adaptive Extended Layers

The maximum number of extended layers that a partition can have is decided by the road network. However, to reduce the total execution time of the simulation, it may not be beneficial to replicate all available extended layers due to redundant computation and extra data in messages. This section introduces a method to determine the suitable number of extended layers to replicate using run-time traffic information. We start with analyzing the overhead of computation replication. The notation used in this section is listed in Table 1.

3.3.1 Analysis of Overhead

Here, we formulate the total overhead incurred for a pair of LPs when a certain number of extended layers is replicated. The MA protocol introduced in Section 2.3 is used for synchronization between LPs. Therefore, the number of extended layers replicated for G_i inside G_j and for G_j inside G_i will always be equal.

Considering a pair of neighboring partitions G_i and G_j, there are $K_{i,j}$ available extended layers between the two. Let a unit of computation be the wall-clock time required for processing the events from one agent during one update interval. If k $(1 \leq k \leq K_{i,j})$ layers are replicated, the total units of redundant computation in LP_i between two consecutive synchronization operations is

$$\sum_{x=1}^{k}\Big(|A_{i,j}^{x}|\cdot(k+1-x)\Big), \tag{1}$$

where $|A_{i,j}^{x}|$ is the number of agents in layer $X_{i,j}^{x}$. It can be estimated using the sizes of the segments in the extended layer and agent densities on the segments.

Therefore, the redundant computation in terms of wall-clock time in LP_i, relative to LP_j between two consecutive synchronization operations is

$$\Gamma_{i,j}(k) = Ta\cdot\sum_{x=1}^{k}\Big(|A_{i,j}^{x}|\cdot(k+1-x)\Big), \tag{2}$$

where Ta is a unit of computation, i.e., the wall-clock time for computing the events of an agent in one update interval. From Equation (2), we can derive the difference quotient of $\Gamma_{i,j}$ between $k+1$ and k as

$$\Delta\Gamma_{i,j}(k+1,k) = Ta\cdot\sum_{x=1}^{k+1}|A_{i,j}^{x}|. \tag{3}$$

It is known that $|A_{i,j}^{x}|\geq 0$; therefore, $\Delta\Gamma_{i,j}(k+1,k)\geq 0$. This equation shows that if extended layers are not empty (i.e., $|A_{i,j}^{x}|>0$), the redundant computation increases with an increasing k and the increase becomes faster as k increases.

The total overhead for LP_i, relative to LP_j, in one synchronization cycle using k extended layers is

$$o_{i,j}(k) = \Gamma_{i,j}(k) + s_{i,j}(k), \tag{4}$$

where $s_{i,j}(k)$ is the overhead for sending one messages from LP_j to LP_i during synchronization when k extended layers are replicated. It is the time that LP_i needs to wait for the message from LP_j to arrive before progressing.

Based on the fact that the time for message-passing generally involves the actual message transmission time and certain latencies on the sending and receiving ends, we use the following model to express the message-passing time

$$time = \frac{data\_volume}{bandwidth} + send\_recv\_latency, \tag{5}$$

where the first item is transmission time and the second item is latency. Note that bandwidth and latency may not be constants, i.e., they may vary with different message sizes.

We conducted an experiment to profile message-passing overhead for MPI in the cluster used for our experimentation to validate this simple model. Results show that a linear regression fits quite well to the MPI message passing overhead. However, as indicated in the existing literature (e.g., [16] and [17]), it is not straightforward (may be even impossible) to define an accurate cost model for message-passing that can be universally applied. Hence, the model should be adapted according to the specific experimentation set-up.

Based on Equation (5), $s_{i,j}(k)$ can be expressed as

$$s_{i,j}(k) = \frac{D_a}{bw}\cdot\left(\sum_{x=1}^{k}|A_{i,j}^{x}| + |M_{j,i}|\right) + \frac{D_s}{bw}\cdot|S_{j,i}^{k}| + srl, \tag{6}$$

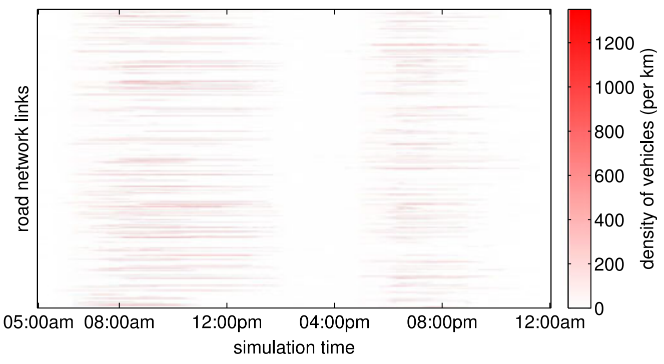


Fig. 7. Heat map of the dynamic traffic densities on main roads in a simulation of Singapore city traffic from 5 am to 12 am.

where D_a and D_s are the data sizes of a complete agent and the shared state of an agent respectively, $M_{j,i}$ is the set of migrating agents from LP_j to LP_i at the time of synchronization, $S_{j,i}^{k}$ is the set of shared states that LP_j sends to LP_i during the synchronization, bw is the available bandwidth for message-passing, and srl is a send and receive latency for the message. Values of bw and srl should be calibrated according to the actual simulation environment. They are not necessarily constants and can be different for different LP pairs. In general, message-passing time increases as the message size increases. So, this equation shows that as the number of replicated extended layers increases, the overhead of passing one synchronization message increases as well.

In order to compare the overhead of synchronization using different numbers of extended layers, we define the total overhead for LP_i, relative to LP_j, during a simulation period τ, $\tau \gg \delta$, with k extended layers, as follows:

$$o_{i,j}(\tau,\,k) = \frac{\tau}{(k+1)\cdot\delta}\cdot o_{i,j}(k). \tag{7}$$

Equation (7) shows that as the number of replicated extended layers (i.e., k) increases, the number of synchronization operations during period τ (i.e., the first term of the right-hand side of Equation (7)) decreases; however the total overhead per synchronization cycle (i.e., the second term of the right-hand side of Equation (7)) increases. The optimum k should give the best trade-off between the benefit of reducing the synchronization frequency and the total overhead caused by redundant computation. It can be noticed from Equations (2) and (6) that the total overhead is influenced by many factors, including the numbers of agents in the extended layers, time to compute agent states for one update interval, data sizes of complete agents and shared states, message-passing bandwidth, and send and receive latency. Obviously, $o_{i,j}(\tau,\,k)$ is not guaranteed to be a simple convex or concave function, thus the optimum k cannot always be determined mathematically using Equation (7). A naive way is to traverse all possible k values and pick the optimum.

3.3.2 Dynamic Determination of the Optimum Number of Extended Layers to Replicate

Traffic density and flow on the roads often change throughout the simulation. To give an example, the traffic densities on main roads in our simulation of Singapore city traffic are shown in Fig. 7.

The heat map in Fig. 7 shows that at a certain time of the day, different roads can have different traffic densities. Traffic density on the same link also changes throughout the simulation. Due to computation overhead, it is impractical to recompute the optimum number of extended layers to replicate whenever traffic condition changes. Hence, in this section, we propose a heuristic that periodically adjusts the number of extended layers for computation replication. We denote the obtainable optimum number of extended layers to replicate as $\hat{k}_{i,j}$.

Suppose that the simulation time from the start to the end can be segmented into M ($M \geq 1$) periods, and in each period the traffic flow and density of links have no or only marginal changes. Denote the mth ($1 \leq m \leq M$) period as τ_m. Reevaluation of $\hat{k}_{i,j}$ is only required at the beginning of each period. So, lookahead remains constant during the period. Obviously, the smaller the period is, the more accurately $\hat{k}_{i,j}$ can be determined, but the more frequent reevaluation will be required. The size of a period (i.e., τ) should be much larger than one update interval (i.e., δ).

To determine $\hat{k}_{i,j}$, $o_{i,j}(\tau, k)$ needs to be evaluated for each k and agent densities in all link segments needs to be obtained. In cases when there are a large number of available extended layers, the overhead of determining $\hat{k}_{i,j}$ may become significant. To reduce the overhead of reevaluating $\hat{k}_{i,j}$, the following method is proposed to limit the range of k. In consecutive periods τ_{m-1} and τ_m, the obtainable optimum lookahead *may* not vary much, thus it is very likely that the obtainable optimum lookahead in τ_m has a similar value to that of τ_{m-1}. Thus, the lookahead in period τ_{m-1} can be utilized to limit the range of k to be evaluated in period τ_m. An adaptive range in time period τ_m, denoted as $K^m_{i,j}$, determines the number of extended layers between LP_i and LP_j to be evaluated

$$K^m_{i,j} = \begin{cases} K_{i,j}, & \text{if } m = 1 \\ \min(K_{i,j},\, 2 \cdot \hat{k}^{m-1}_{i,j} + 1), & \text{otherwise} \end{cases},$$

where $\hat{k}^{m-1}_{i,j}$ is the obtainable optimum k value between LP_i and LP_j in period τ_{m-1}. The adaptive range limits the search range in period τ_m, such that the resultant lookahead is at most twice of that in τ_{m-1}. Meanwhile, it ensures that the lookahead is able to grow exponentially if necessary. In the first period (i.e., τ_1), the entire range of $[0, K_{i,j}]$ is searched. In period τ_m ($m > 1$), the search range of k is $[0,\, \min(K_{i,j},\, 2 \cdot \hat{k}^{m-1}_{i,j} + 1)]$. $k = 0$ here refers to the case where there are no extended layers. When $k = 0$, $o_{i,j}(0) = s_{i,j}(0)$. Based on Equation (6), $s_{i,j}(0)$ can be derived as follows:

$$s_{i,j}(0) = \frac{D_a}{b} \cdot |M_{j,i}| + \frac{D_s}{b} \cdot |S_{j,i}| + C, \tag{8}$$

where $|S_{j,i}|$ is the number of shared states LP_j sends to LP_i during synchronization when there are no extended layers.

To determine the obtainable optimum number of extended layers to replicate (i.e., $\hat{k}^m_{i,j}$), the adaptive range is first determined. Then, for each value k in the adaptive range, the overhead is calculated as the larger value between $o_{i,j}(\tau_m, k)$ and $o_{j,i}(\tau_m, k)$. $\hat{k}^m_{i,j}$ will be the value that gives the minimum overhead in the range.

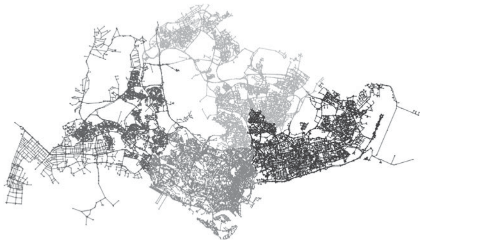


Fig. 8. Singapore city road network partitioned using METIS.

3.4 Overall Execution of the Simulation

When computation replication is incorporated to the MA synchronization protocol, the execution of the simulation shown in Algorithm 1 needs to be slightly altered. The first change is that lookahead is determined at the beginning of each evaluation period (by determining $\hat{k}_{i,j}$) and is kept fixed during the period, instead of being determined in synchronization operations. The second change is that external agents are sent between LPs during synchronization and their states are updated by the corresponding LPs.

The effectiveness of the proposed method of dynamically determining the number of extended layers to be replicated has been investigated in experiments and the results will be presented in the next section.

4 Experiments and Results

4.1 Set-Up

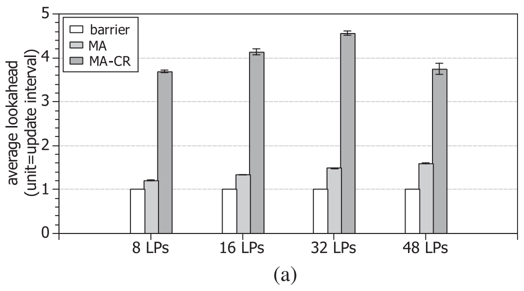
4.1.1 Implementation and Hardware

The proposed heuristics were experimented with SEMSim Traffic simulator [18]. It is implemented using C++. Communication between LPs is realized using OpenMPI 1.8.1. The experiments were run on a cluster composed of three compute nodes. The hardware of each compute node is: $Octacore\ Intel(R)\ Xeon(R)/2.6\,GHz \times 2$ CPUs, 192 GB memory. Compute nodes are connected via 56 Gbps InfiniBand. Different LPs are mapped to different CPU cores.

4.1.2 Workload

Experiments used a real-world road network. It is the road network of Singapore city consisting of approximately 80,000 links and 40,000 nodes in our representation. Agents move on the road network according to their trips derived from the data of the Household Interview and Travel Survey (HITS) in 2008. In every update interval, agent states are updated by the Intelligent Driver Model [19] and a rule-based lane-changing model. They form the major computational workload of the simulation. The computation time for models in an agent in one update interval is around 2 μs. The data size of migrated agents and shared states are around 200 and 100 bytes, respectively. The update interval (i.e., δ) is 0.5 seconds; and the front and back sensing ranges of agents are 40 and 20 meters respectively. The road network is initially partitioned using METIS [20]. An example of the road network in four partitions is shown in Fig. 8. Different intensities of gray represent different partitions.

The traffic of 19 hours from 5 am to the midnight of the same day was simulated. To reduce the influence of workload imbalance of LPs, dynamic load-balancing described in [15] was employed. Dynamic load-balancing repartitions



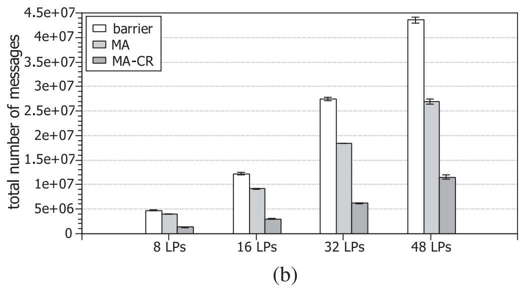


Fig. 9. (a) Average lookahead of all LPs throughout the simulation (larger lookahead means less frequent synchronization). (b) Total number of messages sent during the simulation (the smaller the better).

the road network during the simulation to balance the workload of LPs so as to improve the speed-up. The workload of LPs is periodically checked and a repartitioning is performed when workload imbalance exceeds a threshold. Due to the change of network partitions, extended layers need to be re-computed by running Algorithm 2. The simulation was run with 8, 16, 32, and 48 LPs. For each setting, four simulation runs using different seeds for random number generation were performed. Means and standard deviations of the measurements were collected.

4.2 Results

4.2.1 Lookahead and Synchronization Messages

We investigated whether the average lookahead of the MA synchronization protocol is increased. The optimum lookahead was re-evaluated every 10 minutes of simulation time. (The results using different evaluation periods are presented in Section 4.2.4.) The average lookahead is calculated by averaging the lookahead values throughout the simulation for all LPs. Let us denote the MA protocol with computation replication as *MA-CR*. The average lookahead values of MA and MA-CR methods, in terms of number of update interval, are shown in Fig. 9a. The total number of synchronization messages sent during the simulation is shown in Fig. 9b. For comparison, the results of using global barrier synchronization, denoted as *barrier*, are also shown in the figures. Note that the average obtainable optimum number of extended layers to replicate (i.e., $\hat{k}$) is equal to lookahead minus 1.

It can be observed from Fig. 9a that the lookahead values of the MA-CR method are larger than those of the MA

TABLE 2
Average Number of Available Extended Layers

	8 LPs	16 LPs	32 LPs	48 LPs
layer count	23.6	18.7	16.9	16.0

TABLE 3
Average Number of Agents and Shared States Sent per Message

	8 LPs	16 LPs	32 LPs	48 LPs
MA agent	0.34	0.22	0.16	0.15
MA-CR agent	25.3	20.5	14.2	13.7
MA shared state	2.53	1.86	1.53	1.48
MA-CR shared state	5.29	3.41	2.33	1.78

TABLE 4
Average Lookahead of All LPs Throughout the Simulation with and Without Adaptive Range (Unit=Update Intervals)

	8 LPs	16 LPs	32 LPs	48 LPs
with	3.69±0.03	4.13±0.06	4.56± 0.05	3.74±0.13
without	3.65±0.14	4.17±0.15	4.59± 0.13	3.59±0.03

method. For 32 LPs, there is approximately a threefold increase. Correspondingly, there are much fewer synchronization messages when the MA-CR method is used, compared to the MA method, as shown in Fig. 9b. This reduction of synchronisation messages comes with an increase of message sizes, which is shown in Table 3.

To observe whether all available extended layers are replicated, the average numbers of available extended layers are shown in Table 2. It can be observed that the number of available extended layers decreases as the number of LPs increases. The actual numbers of layers replicated are much smaller than the available numbers of layers.

The average numbers of complete agents and shared states sent in a message using MA and MA-CR methods are shown in Table 3.

It can be observed from Table 3 that the average size of messages has increased using the MA-CR method. This is mainly caused by the replication of external agents.

Moreover, we compared the lookahead values obtained with and without using adaptive range. The result is shown in Table 4. It shows that there is no significant difference in the lookahead values with and without adaptive range. The adaptive range does not affect lookahead much.

4.2.2 Overall Speed-Up

The overall speed-up of the parallel simulation is shown in Fig. 10. The speed-up is measured against the sequential simulation, of which execution time is around 9000 seconds.

The MA-CR method has the highest speed-up, and the barrier method has the lowest speed-up. This shows that using

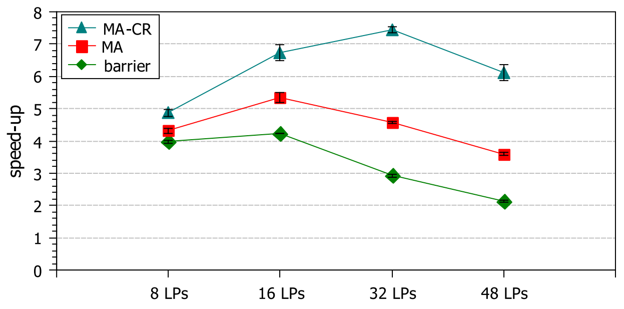


Fig. 10. Speed-up of the parallel simulation with respect to the sequential simulation.

TABLE 5
Redundant Computation as a Percentage of the Total Simulation Execution Time

	8 LPs	16 LPs	32 LPs	48 LPs
Percentage	0.77± 0.03	1.16 ± 0.04	1.50±0.03	0.87±0.03

adaptive computation replication has improved the overall performance of the simulation. The improvement is the result of increased lookahead and reduced number of synchronization messages, as shown in Fig. 9. This means that the benefit of reduced synchronization messages exceeded redundant computation and the overhead of increased message sizes.

When the number of LPs increases from 16 to 32, the speed-up using MA-CR increases further; whereas the speed-up using MA decreases. This is because the absolute number of messages for MA increases drastically as the number of LPs increases, although the percentages of increment on lookahead for MA and MA-CR are similar, as shown in Fig. 9.

4.2.3 Redundant Computation and Overhead of Algorithms

The amount of redundant computation and the overhead of the proposed method are investigated in this section. The amount of redundant computation, calculated as a percentage of the total simulation execution time, is shown in Table 5.

For all cases in Table 3, redundant computation is less than 1.5 percent of the total execution time.

The computational overhead of the method comes from calculating extended layers (Algorithms 2) and calculating optimum lookahead. Due to dynamic partitioning of the road network, Algorithm 2 was executed 47 ± 2, 40 ± 1, 31 ± 1, and 28±1 times for 8, 16, 32, and 48 LPs respectively. Optimum lookahead was calculated was executed 114 times for all different numbers of LPs (every 10 minutes in 19 hours simulation time). Both algorithms are executed by every LP, hence, the maximum overhead amongst the LPs is collected. Total overheads of the algorithms throughout the simulation are shown in Table 6.

Table 6 shows that the overheads introduced are at the magnitude of seconds which are insignificant with respect to the overall simulation execution time.

Besides computational overheads, extra memory is also required for the computation replication method. It consists of mainly two parts: storing segments of extended layers, and storing external agents. The memory for storing segments depends on the total number of segments in the road network. Storing external agents takes memory proportional to redundant computation. In our experiment, storing segments is not significant compared to the memory usage of the whole simulation. Memory for storing agents is increased by 1.5 percent (same percentage as redundant computation shown in Table 5).

TABLE 6
Total Overhead of Calculating Extended Layers and Estimating the Optimum Lookahead (Unit=Second)

	8 LPs	16 LPs	32 LPs	48 LPs
calc extended layer	6.1±1.9	1.6±0.5	0.6±0.1	0.2±0.01
calc lookahead	2.6±0.1	1.2±0.1	0.8± 0.1	1.0±0.02

TABLE 7
Comparison of Average Lookahead, Execution Time, and Overhead of Evaluating Optimum Lookahead Using Different Evaluation Periods with 32 LPs

reevaluation period	lookahead (update intervals)	execution time (second)	overhead (second)
1 min	4.50±0.15	1275.6±36.8	20.9± 1.34
5 min	4.47±0.14	1261.5±22.7	2.98± 0.29
10 min	4.56±0.05	1253.8±36.6	0.83± 0.06

4.2.4 Frequency of Evaluation

Experiments have also been conducted to analyze the effect of different evaluation periods on the average lookahead, the overall simulation execution time, and the overhead of adapting the optimum lookahead. The results of using evaluation periods of 1, 5, and 10 minutes with 32 LPs are shown in Table 7. There is no significant difference in the average lookahead using different evaluation periods. This is because the traffic condition does not change so often. Therefore, frequent reevaluations of the optimum number of extended layers to replicate is not necessary. The overhead of executing the algorithm increases when the frequency of reevaluation increases, which also results in slight increase of the total execution time.

The overhead of evaluating the optimal lookahead without adaptive range for 32 LPs is similar to that shown in Table 7. However, for 8 LPs, the overheads with and without adaptive range are 18.11±0.92, and 43.6±6.6 seconds, respectively, when the evaluation period is 1 minute. The benefit of adaptive range is more obvious for more frequent evaluations and fewer LPs (i.e., more available extended layers).

4.2.5 Weak Scaling Property

In addition to the strong scaling property, it is also interesting to analyze the weak scaling property on the MA-CR method. So, we conducted another set of experiments using four different problem sizes created using artificial road networks. They are 48×16, 96×16, 192×16, and 384×16 rectangular grid networks. Each road link in the road networks is 200 meters long. Agent populations are 5k, 10k, 20k, and 40k respectively. Simulations with the four different networks are run with 12, 24, 48, and 96 LPs, respectively. The road networks are partitioned into stripes vertically. Thus, an LP has the same workload for all cases in terms of both the size of the network partition and the number of agents. A larger cluster with four compute nodes was used in the experiments, where each compute node has 24 CPU cores. Each case was run four times and the average speed-up was taken. The result is shown in Fig. 11.

The results in Fig. 11 show that when the problem size is scaled up proportionally with increasing number of LPs, speedup remains increasing for both MA-CR and MA approaches. As expected, the amount of increment becomes less when number of LPs increases. With more LPs, the problem size also becomes larger. As a result, more communications are required between different compute nodes. In addition, synchronization overhead also increases with the number of LPs. As shown in the figure above, MA-CR

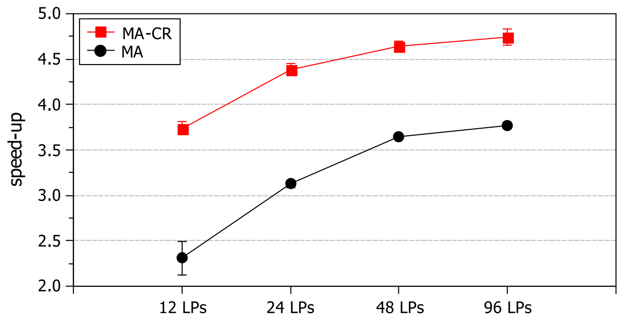


Fig. 11. Average speed-up of parallel simulation with MA and MA-CR using different problem sizes and LPs.

always outperforms MA and there is no significant drop in performance improvement (that is, the distance between the two lines) when number of LPs increases.

5 Related Work

The synchronization issue of parallel discrete-event simulations has been extensively studied in the literature [3], [6]. Nevertheless, improving the performance of synchronization protocols usually involves tuning the protocols for the models used in particular simulations. For parallel agent-based traffic simulations, there have been many approaches to conduct synchronization. Employing global barriers is the most frequently used approach due to its simplicity. LPs are blocked from executing agent models at the end of update intervals, then they exchange messages with relevant LPs and proceed to the next update interval simultaneously [4], [5]. For parallel simulation using multi-threading, all threads access the shared memory directly at barriers instead of communicating with each other by message passing [13]. This simplicity frequently comes at the cost of LPs waiting at global barriers. It does not fully exploit the parallelism of the simulation. The work in [7] used a conservative time window synchronization from [21]. The synchronization protocol allows LPs to progress asynchronously. LPs analyze their event lists and determine the lower bounds of simulation time for neighboring LPs, until which they do not affect the neighboring LPs. LPs collect the time bounds from *all* neighboring LPs and take the minimum values as their time windows. They execute local events within the time windows, and then synchronize again. This is inefficient since LPs still need to communicate with all neighbors during synchronization. Communicating to all neighbors is unnecessary. The methods above are conservative approaches, in which no violations of data dependencies are allowed at any time of the simulation.

Another attempt is to use an optimistic approach [22], [23]. LPs are allowed to progress over the synchronization point and violations of data dependencies are examined. If there is a violation, the simulation is rolled-back to the point before the violation happens. However, the disadvantage of this approach is the overhead of state saving and performing roll-back operations. Some optimistic simulation frameworks use reverse computation to avoid state saving overhead [24]. Our work aims at improving the efficiency of conservative synchronization methods in agent-based traffic simulation. In particular, we try to reduce synchronization operations by redundant computation.

The idea of reducing communication at the expense of performing some redundant computation has been presented in many previous works. Ghost cell expansion is used in [8] for solving partial differential equation (PDE) problems using finite difference method. A similar concept has also been applied in sparse matrix-vector multiplication problems [9]. The computational tasks in those works are organized as regular 2D and 3D grids of cells or meshes. They are static and their interactions are predictable. Thus, the overhead of computation replication can be analyzed prior to the execution of the program. Dynamic adaptation of redundant computation is not required. Consequently, the definition of extended layers based on grids or meshes cannot be used directly in agent-based traffic simulations.

Computation replication has also been applied in agent-based simulations [10], [11], [12]. However, the simulation space for many of these agent-based applications is a 2D grid. In some cases, agents are not even situated in a spatial environment. For instance, in a social network, agents are represented as vertices of a graph, and interactions of agents are represented as edges. Hence, the definitions of extended layers in those works are not applicable for traffic simulation either. The indexing of the positions of agents and their interactions are different in a 2D grid or interaction graph from those in a spatial road network. In addition, the dynamics of the workload in the replicated region during simulation run-time is not considered either in these works. Our work aims at accurately defining extended layers for agent-based traffic simulation, as well as developing an effective approach to dynamically balance the trade-off between the overhead of redundant computation and the benefit of reducing synchronization operations.

6 Conclusion and Future Work

The purpose of this work is to reduce the total execution time of parallel agent-based road traffic simulation by reducing synchronization overhead. Making agent-based road traffic simulation fast is crucial for real-time decision support systems and studies that require a lot of simulation runs. Computation replication has been applied mainly in applications such as matrix multiplication and solving partial differential equations to reduce inter-process communication. However, there is little work done for applying this approach to parallel agent-based road traffic simulation.

In this article, we mainly focused on two problems for effectively applying the concept of computation replication to agent-based traffic simulation: i) how to determine the redundant computation required to achieve a certain synchronization frequency, and ii) how to manage the trade-off between the overhead caused by redundant computation and the benefit of reduced synchronization. The first problem was solved by analyzing the characteristics of agents and road networks to determine extended layers of partitions in a road network. Sensing ranges and movement of agents were considered. The second problem was solved by developing an analytical model of the total overhead of redundant computation and synchronization, and using it to dynamically adjust the number of extended layers to replicate according to traffic conditions. To reduce the overhead of determining the optimum number of extended layers to replicate when traffic condition changes, an adaptive approach was used to limit the search range.

The efficiency of the adaptive computation replication method has been investigated in a parallel agent-based traffic simulator using real world data. Experiments have shown that the method is able to reduce synchronization messages significantly by increasing lookahead of LPs. The overall execution time of the parallel simulation has been significantly reduced, though there is redundant computation.

As of future work, there are two interesting directions. First, similar to the extended layers, inner areas of a partition can also be divided into layers. This can be used for overlapping communication and computation to further reduce synchronization overhead. Second, partitioning of the road network affects locations and shapes of extended layers, thus the performance may be further improved if the partitioning of the road network considers the optimization of computation replication.

ACKNOWLEDGMENTS

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

REFERENCES

[1] A. Kesting, M. Treiber, and D. Helbing, "Agents for traffic simulation," *Multi-Agent Syst. Simul. Appl.*, A. M. Uhrmacher and D. Weyns, Eds. Boca Raton, FL, USA: CRC Press, ch. 11, pp. 325–356, 2009.

[2] H. Mizuta, Y. Yamagata, and H. Seya, "Large-scale traffic simualtion for low-carbon city," in *Proc. Winter Simul. Conf.*, Dec. 09–12, 2012, pp. 1–12.

[3] R. M. Fujimoto, *Parallel and Distribution Simulation Systems*. New York, NY, USA: Wiley Interscience, 2000.

[4] K. Nagel and M. Rickert, "Parallel implementation of the TRANSIMS," *Parallel Comput.*, vol. 27, no. 12, pp. 1611–1639, 2001.

[5] T. Suzumura and H. Kanezashi, "Highly scalable X10-based agent simulation platform and its application to large-scale traffic simulation," in *Proc. IEEE/ACM Int. Symp. Distrib. Simul. Real Time Appl.*, Oct. 25–27, 2012, pp. 243–250.

[6] D. M. Nicol, "Principles of conservative parallel simulation," in *Proc. Winter Simul. Conf.*, Dec. 08–11, 1996, pp. 128–135.

[7] M. Namekawa, A. Satoh, H. Mori, K. Yikai, and T. Nakanishi, "Clock synchronization algorithm for parallel road-traffic simulation system in a wide area," *Math. Comput. Simul.*, vol. 48, no. 4–6, pp. 351–359, 1999.

[8] C. Ding and Y. He, "A ghost cell expansion method for reducing communications in solving PDE problems," in *Proc. ACM/IEEE Conf. Supercomputing*, Nov. 10–16, 2001, pp. 1–12.

[9] J. Demmel, "Avoiding communication in sparse matrix computations," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 14–18, 2008, pp. 1–12.

[10] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-GPU and multi-core clusters," in *Proc. 3rd Int. ICST Conf. Simul. Tools Techn.*, Mar. 15–19, 2010, pp. 1–10.

[11] T. Zou, G. Wang, and M. Salles, "Making time-stepped applications tick in the cloud," in *Proc. 2nd ACM Symp. Cloud Comput.*, Oct. 26–28, 2011, pp. 1–14.

[12] R. Zunino, "Trading computation time for synchronization time in spatial distributed simulation," in *Proc. IEEE Workshop Principles Adv. Distrib. Simul.*, Jun. 14–17, 2011, pp. 1–8.

[13] J. Barceló, J. L. Ferrer, and D. Garcia, "Microscopic traffic simulation for ATT systems analysis. A parallel computing version," in *Proc. Contribution 25th Aniversary CRT*, 1998, pp. 1–16.

[14] Y. Xu, W. Cai, H. Aydt, M. Lees, and D. Zehe, "An asynchronous synchronization strategy for parallel large-scale agent-based traffic simulations," in *Proc. 3rd ACM SIGSIM Conf. Principles Adv. Discrete Simul.*, Jun. 10–12, 2015, pp. 259–269.

[15] Y. Xu, W. Cai, H. Aydt, and M. Lees, "Efficient graph-based dynamic load-balancing for parallel large-scale agent-based traffic simulation," in *Proc. Winter Simul. Conf.*, Dec. 7–10, 2014, pp. 3483–3494.

[16] K. Al-Tawil and C. A. Moritz, "Performance Modeling and Evaluation of MPI," *J. Parallel Distrib. Comput.*, vol. 61, no. 2, pp. 202–223, 2001.

[17] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, "Toward performance models of MPI implementations for understanding application scaling issues," in *Proc. 17th Eur. MPI Users' Group Meeting Conf. Recent Advances Message Passing Interface*, Sep. 12–15, 2010, pp. 21–30.

[18] Y. Xu, H. Aydt, and M. Lees, "SEMSim: A distributed architecture for multi-scale traffic simulation," in *Proc. 26th ACM/IEEE/SCS Workshop Principles Adv. Distrib. Simul.*, Jul. 15–19, 2012, pp. 178–180.

[19] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, no. 2, 2000, Art. no. 1805.

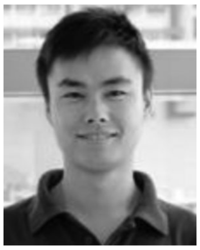
[20] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1999.

[21] R. Ayani and H. Rajaei, "Parallel simulation using conservative time windows," in *Proc. 24th Winter Simul. Conf.*, Dec. 13–16, 1992, pp. 709–717.

[22] S. B. Yoginath and K. S. Perumalla, "Parallel vehicular traffic simulation using reverse computation-based optimistic execution," in *Proc. 22nd Workshop Principles Adv. Distrib. Simul.*, Jun. 3–6, 2008, pp. 33–42.

[23] M. Hanai, T. Suzumura, G. Theodoropoulos, and K. S. Perumalla, "Exact-differential large-scale traffic simulation," in *Proc. 3rd ACM SIGSIM Conf. Principles Adv. Discrete Simul.*, 2015, pp. 271–280.

[24] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, 1999.



Yadong Xu is a currently research associate of TUMCREATE Ltd. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Nanyang Technological University (NTU), Singapore. His thesis topic is on the load-balancing and synchronization of parallel agent-based road traffic simulation. His research interests are primarily parallel and distributed systems, agent-based simulation, ontologies for modelling and simulation, and intelligent transportation system. He is a member of the IEEE.

Vaisagh Viswanathan received the PhD degree in modelling behavior in agent-based simulations of crowd egress from NTU, Singapore, in 2015. He was a post-doctoral research fellow at TUMCREATE Ltd. working on Modelling and Optimization of Architectures and Infrastructure. His research investigates the infrastructure requirements and the environmental impact of large scale electro-mobility from a complex systems perspective. His research interests include primarily agent based modeling and simulation, complex adaptive systems. He is a member of the IEEE.

Wentong Cai is a professor in the School of Computer Science and Engineering, NTU, Singapore. His expertise is mainly in the areas of modeling and simulation and parallel and distributed computing. He is an associate editor of the *ACM Transactions on Modeling and Computer Simulation* and an editor of the *Future Generation Computer Systems*. He has chaired a number of international conferences. Most recent ones include SIGSIM PADS 2017, SIMUTools 2016, DS-RT 2015, and CloudCom 2014. He is a member of the IEEE.