A framework for parallel traffic simulation using multiple instancing of a simulation program

Der-Horng Lee
Department of Civil Engineering
National University of Singapore
Blk E1A, #07-03
1 Engineering Drive 2
Singapore 117576
Telephone: +65 874 2131
Fax: +65 779 1635
Email: dhl@nus.edu.sg

P. Chandrasekar
Department of Civil Engineering
National University of Singapore
Blk E1A, #07-03
1 Engineering Drive 2
Singapore 117576
Telephone: +65 874 5035
Fax: +65 779 1635
Email: cvepc@nus.edu.sg

Abstract

Parallel traffic simulation is an application of parallel computing techniques aiming to decrease the computational time by engaging different processors of a multiprocessor system or different computers of a network. Very few traffic simulation models have this capability of parallel simulation. However, it is possible to upgrade a simulation program that is not capable of running parallel simulation by applying a method proposed in this paper. The method involves dividing the network into regions and simulating each region under a separate instance of the program. Suitable inter-process communication techniques are employed to exchange data between different regions and synchronize time among the different regions. A significant increase in simulation speed is seen when the proposed method is applied to Paramics, a microscopic time-stepping simulation program.

Keywords:

Parallel Simulation, Domain Decomposition, Multiple Instances, Scalable Simulation, Large-Scale Simulation, Traffic Simulation.

1. Introduction

Computer-based traffic simulation had been a cost-effective approach adopted by researchers and practitioners to design and evaluates traffic management strategies for many years. Availability of at least forty traffic simulation programs (inclusive of all micro, meso and macroscopic models), most of them developed in the past decade only, portrays the importance that simulation has gained among the traffic community. Software design has been improved in several dimensions. These include graphics providing 2D/3D animation for better visualization, more user-friendliness in building a network and coding its traffic operations, and provision of interfacing capabilities if the program is to be coupled with other software or even hardware.

A factor has gained more importance, especially when simulating a large network, is the execution speed. For large-scale traffic simulation, in particular, when simulation needs to be performed faster than real-time, such as when simulation is employed as a tool for real-time traffic management strategies or as an engine for traffic forecasting purposes, execution speed is crucial. In order to increase the execution speed, developers have adopted parallel computing techniques. These techniques aim to decrease computational time by engaging different processors of a multiprocessor machine to share the workload of a simulation program when the program is executed. Therefore, these techniques enable scalable simulation, i.e., to run simulation at speeds relative to the number of processors in the computer. Thus, parallel simulation is an effective way to increase the execution speed.

A few large-scale traffic simulation programs have adopted parallel computing techniques, including TRANSIMS (Bernauer et al 1998), AIMSUN (Barcelo et al 1998) and

Paramics (Cameron and Duncon 1996). For programs that are parallel-based, like the above, parallelism is so embedded that no parallel simulation-specific input is necessary from the user. In the case of non-parallel (or sequential) simulation programs, even with the availability of source code, users need methods to implement parallelism without expending huge resources for re-designing, testing and validating their models.

There seems to be an opportunity to adopt parallel simulation for those simulation programs that were not originally designed to run parallel simulation, without completely redesigning the source code but by only adding a few routines. This paper introduces such a method. The idea is to disintegrate the traffic network and simulate the divided portions on individual processors of a multi-processor system, separately and simultaneously using what is known as the multiple instancing of the program. The method is very close to domain distributed computing, but there are differences (described later in this paper). The advantage of the proposed method is that a network containing distinct regions or neighborhoods linked by highways can be simulated separately. Under this environment, use of the proposed method can enable simulating each region simultaneously, maintaining the linkage between the regions, i.e., maintaining time synchronization and vehicle transfers at boundaries.

Paramics is used to test the proposed method. Although Paramics is designed to adopt parallel simulation (hence the acronym Paramics which stands for Parallel Microscopic Simulation) on a multi-processor machine, the proposed method does not engage any such feature available in the program. Instead, the program is used to test the proposed method, as if there is no such default feature available for use.

This paper contains six sections (including this introductory section). The second section describes the implementation of parallel simulation in a few large-scale simulation programs.

This review will be useful to identify in what ways the proposed method is different from the conventional methods. The third section describes the proposed method in detail. The fourth section describes the test environment where the proposed method is applied and tested. In the fifth section, results are presented and discussed. The last section concludes this paper and provides a note on the further enhancements planned for the proposed method.

2. Parallelism in traffic simulation programs

A variety of techniques have been employed in parallel simulation. The usual approach is to incorporate parallel algorithms in the source-code itself so that the computational workload is shared among different processors. In the following sections, the parallel implementations of simulation programs such as TRANSIMS, AIMSUN and Paramics are reviewed with respect to their abilities to handle very large sized networks.

2.1 Parallel implementation in TRANSIMS

TRANSIMS, (Transportation Analysis and Simulation System) is an integrated system of travel forecasting models designed to give transportation planners information on traffic impacts, congestion, and pollution (LANL 2001). TRANSIMS models create a virtual metropolitan region with a representation of the region's individuals, their activities, and the transportation infrastructure. Trips are planned to satisfy the individuals' activity patterns.

TRANSIMS then simulates the movement of individuals across the transportation network, including their use of vehicles such as cars or buses, on a second-by-second basis. The micro-simulation procedure uses a cellular automata (CA) technique for representing driving dynamics (Nagel and Rickert 1999). The road is divided into cells, each of a length that a car uses up in a jam. A cell is either empty, or occupied by exactly one car. Movement takes place by hopping from one cell to another; different vehicle speeds are represented by different hopping distances.

Parallel implementation in TRANSIMS is based on a domain decomposition principle, which means the network is partitioned into domains of approximately equal size, and then each CPU of the parallel computer is responsible for one of these domains (Nagel and Rickert 1999). There will be as many domains as CPUs engaged. A realistic measure for domain size is the accumulated length of all streets associated with the domain. It is reported that using CA helps with the design of a parallel and local simulation update; that is, the state at a time step depends only on information from the previous time step and only from neighboring cells. This means that domain decomposition for parallelism is straightforward, since one can communicate the boundaries for any given time step, then locally on each CPU perform the update from that time step to the next, and then exchange boundary information again.

2.2 Parallel implementation in AIMSUN

AIMSUN (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks) is a microscopic simulation program originally developed as a sequential but later ported to parallel computers (Barcelo et al 1998). Simulation can be either based on input traffic flows and turning proportions, or based on origin-destination demand matrices and route selection models. In the former, vehicles are distributed stochastically around the network, whereas in the latter vehicles are assigned to specific routes from the start of their journey to their destination.

For implementing parallelism, AIMSUN uses ‘threads’ to process ‘blocks’ of entities that need to be updated at every time step (Barcelo et al 1998). A thread is a sequence of instructions executed within the context of a process (here, the simulation program). A process can run its job by invoking several threads. Threads can be executed in parallel by the multiprocessor system. A block is a group of entities, while an entity can be any object (such as a vehicle) that needs to be updated. For example, a block can contain an intersection and its approaches, including vehicles on these approaches. AIMSUN handles grouping of such entities into blocks and allocation of each block to a single thread.

2.3 Parallel implementation in Paramics

The following details about the parallel implementation of Paramics are based on the paper by Cameron and Duncon (1996). The authors describe two approaches of implementing parallelism in Paramics. The first approach is based on data-parallelism (which is domain decomposition) and the second approach is based on a message-passing (MP) standard.

The road traffic network is supplied as a system of nodes (positioned at junction points) connected by links. These links are unidirectional, and represent opposing half-carriageways; hence, each pair of nodes is connected by two links. The first approach was reported as tested on a Connection Machine (CM-200). The authors describe that, to make good use of the CM-200, the data must be in a parallel array form so that operations can occur in parallel on the elements of the array. The approach used to build a parallel data framework for the simulation process was to associate a number of queues with each of these unidirectional links. A queue was used as the parallel item of data on the CM-200. The parallel array is a one-dimensional array consisting of a large number of these queues, and can be operated concurrently.

The second approach reported is based on domain decomposition at a traffic network level. The whole network is divided and a dedicated processor simulates each region. The network subdivision process uses a snapshot of the whole network as input and it based on a specified number of processors. Using the number of vehicles on a link as the first estimate of that link's activity level, a set of output files is produced, one for each region. Each file lists the links that are allocated in that region, the vehicles on those links (specifically their position, destination and type), and the name and remote end processor number of the boundary links that define the limits of the region. Clearly, the balance of traffic load (and by direct consequence, computational load on the simulator) may shift over time. In some networks, the magnitude of the shift between, for example, morning peak-hour and off-peak traffic may be so great as to necessitate moving the boundaries to improve the performance. This movement of processing responsibility is referred to as dynamic load balancing.

3. Proposed method for implementation of parallel simulation

The method proposed here takes advantage of running the simulation program under multiple instances (i.e., by opening and executing the program more than once, simultaneously). First, the traffic network is divided into different regions and coded separately as individual networks. The number of regions can be as many as the number of processors available on the machine. Each region is simulated under one instance of the program (see Fig. 1b). Each instance of the program is referred to as a "process". In a windowing environment, each process will appear on a separate window. Note from Fig. 1a that in the conventional parallel simulation, a whole network appears in one single instance of the program and network decomposition is handled automatically by the program. Therefore, in the proposed method, different portions of the divided network appear in different windows (see Fig. 1b). Using UNIX shell programming, processors of the multiprocessor machine are bound to handle each process exclusively. In this setup, time synchronization and vehicle transfer at boundaries are required to make the divided regions of the original network behave as a single network as a whole. To achieve this, processes need to share data among themselves. For this purpose, inter-process communication (IPC) method (Stevens and Richards 1999), specifically, communication using shared memory is employed. The IPC mechanisms allow processes to communicate with each other by sharing a portion of the random access memory (RAM) to write and read data as and when required. An analogous method is to use a text file and allow different processes to share the same file for reading and writing. However, shared memory is faster and more reliable. The procedure by which shared memory is used for time synchronization and transfer of vehicles across the boundaries is explained next.

3.1 Time synchronization

Time synchronization is maintained at a timestep level. Each process updates a variable (named as status variable) stored in memory with an entry of "1" when all the computation work associated with a timestep is over. Another variable (named as sum variable) is stored in memory to hold the sum of status variables associated with all the processes. This variable can be read by any process. Once a process completes a timestep, it must wait for the sum variable to reach a value equal to the total number of processes before executing the next timestep. Each process verifies the sum value, begins a new timestep and re-writes its status variable from "1" to "0". In this way, no process can proceed from one timestep to another until all the processes complete the corresponding timestep.

3.2 Vehicle transfers across boundaries

In order to maintain the spatial connectivity between regions simulated separately, vehicles reaching the boundary of a region must re-appear in another region connected at the boundary. This is done by capturing the data structure of the vehicle when the latter reaches its boundary, and storing it in the shared memory for future reference. The address where these are stored in the memory is known to all processes. At each time step, each process will check this address to determine whether any vehicle data are stored. In this case, the process will then copy the vehicle structure, release a similar vehicle into its region, and empty the location where the structure was read. These data should cover the type of vehicle, vehicle's lane, speed of vehicle while leaving the boundary, and any driver-specific factors such as driver aggression and awareness levels (these two items are Paramics-specific) (Quadstone 2000).

3.3 Network decomposition

Speed of simulation will be governed by the heaviest load bearing processor since all the processors are controlled under time synchronization. Conventional parallel simulation methods adopt techniques such as graph partitioning, in order to distribute the load balanced among different processors. For now, the proposed method assumes that the divided regions of the network will have approximately the same road surface area. Further research will be carried out to formulate suitable load balancing techniques, preferably dynamic, for the proposed method.

The partitioning does not require the division of the network on a link or on a node. The division is subjected to the coding requirements to be followed by the simulation program. The proposed method only pays attention to where to look along the boundary for vehicles to be transferred between regions. A text file containing these data is sufficient for the proposed method to make the processes watch and transfer vehicles as and when required.

3.4 Traffic assignment

As the network is divided into small regions, the default traffic assignment methods such as all-or-nothing and dynamic feedback methods cannot be readily applied in the proposed method. As such, the traffic assignment is based on turn counts at the intersections. This method is easier to implement and facilitates initial testing of the proposed method for its execution speed. A more sophisticated version that enables assignment based on all-or-nothing and dynamic feedback is being developed and tested.

3.5 A summary on the requirements for implementing the proposed method

The key requirements to implement the proposed method are summarized as follows:

- Multiple instancing (i.e., should allow itself to be opened and executed more than once simultaneously).
- Retrieving data on vehicles reaching the boundary of one region.
- Releasing vehicles based on the data retrieved as above.
- Control of simulation to enable time- synchronizing.

It is evident from above requirements that the proposed method implements parallelism with minimal modification to the source code of the program. If the program provides a programmer-level interface (i.e., an indirect access to the functions in the source code), the proposed method can be applied without modifying the original source code. The use of such application programming interface (API) is discussed in detail elsewhere (Lee and Chandrasekar 2001).

4. Test environment

In order to test the performance of the proposed method, a hypothetical grid-type network spreading over 150 sq. km with 500 nodes and 1000 links was built. There are 72 signalized intersections and 1084 loop detectors for vehicle actuated signal control. It was convenient to use a hypothetical network that is fairly larger in size, than to model a real network, due to lack of accurate data. Testing was carried out on a Sun E3500 Server system with 4 processors (450MHz each) and with 3GB RAM. Leaving one processor for routine system operation, three processors were engaged for parallel simulation (although reserving one processor is not mandatory). Three scenarios to be considered here are, by engaging one, two or three processors to simulate the network divided into one, two or three regions respectively. Before describing the performance results, a note on how Paramics was customized to implement the proposed method is described next.

4.1 Customizing Paramics to apply the proposed method

Paramics' Application Programming Interface (API) enables user-defined routines to be added in the main simulation loop. The use of "functions" (or routines that handle one or more specified tasks) is a convenient way of organizing the simulation program to allow interfacing with other programs or data. Paramics API uses C language to define the user functions. The main program comprises several functions that handle various tasks associated with traffic modeling and are called each timestep of the simulation. These functions can be replaced or modified using the API. In general, the API functions can be divided into three categories: overload, override and callback. Overload functions are used to attach additional routines to in-built functions. Override functions are used to replace the contents of the built-in functions. Callback functions provide the data required from simulation to define the overload and override functions. A program needs to be written using the API functions to create a Dynamic Link Library (referred as "DLL" in Windows operating system) or a Shared Object file (referred as "SO" in UNIX operating system). The "DLL" or "SO" file is loaded into the simulation when Paramics is executed. Hence, the files are often referred to as "plugins" as they add a plug-and-play feature to Paramics simulation. It is possible to create generic plugins (i.e. those that are independent of network configuration), so that they can be re-used for other networks.

Paramics allows multiple instancing. Using the API, other requirements of the proposed method are fulfilled. Firstly, data on vehicles reaching the boundary of the region (in one instance of the program) can be retrieved and stored in the shared memory for further use. The data stored are used to re-release the vehicle in the neighboring region connected at the boundary. The API is also used to control the simulation in order to maintain time synchronization.

5. Performance results

Here, the performance of the proposed method is measured based on its impact on the speed of simulation. The proposed method reproduces the same traffic conditions and volume levels before and after partitioning the network since this method does not interfere with any core simulation routines, except that it makes sure that each vehicle reaching a boundary is transferred to the neighboring region.

5.1 Real-time factor and speed-up factor

Performance can be measured using two measures of effectiveness, namely the real-time factor and the speed-up factor. Real-time factor is a measure of relative speed of simulation with respect to real time. For example, a real-time factor of "5" means that the simulation is running five times faster than real time. In other words, five hours of real time is simulated in one hour. Speed-up factor is a measure of parallel simulation performance, which reveals the impact of using more than one processor. For example, a speed-up factor of 3 would mean that simulation could be performed 3-times faster than by engaging a single processor.

In a time-stepping simulation process, computational load relies on the number of vehicles need to be updated for their positions (although an insignificant portion of the load is shared by updating the states of other stationary objects on the network, such as signals, every timestep). Therefore, speed of simulation at any instant of time will depend highly on the number of vehicles (hereafter, vehicle load) at that instant of time. Therefore, a snap-shot of vehicles taken at any instant of time and current speed of simulation will together help to measure the performance of parallel simulation at that instant of time. For the present study, snapshots of vehicles were taken at one-minute interval (of simulation time). Under multiple instances, the sum of vehicle load from each region gives the total vehicle load shared by all the processors engaged at a given instant of time.

The impact on real-time factor, when engaging different number of processors, is presented in Fig. 2. Each curve represents the real-time factor for a specific number of processors. From the figure, it appears that for a given vehicle load, the real-time factor increases as the number of processors engaged increases. Notice that for a vehicle load of 6000 vehicles, the real-time factor is approximately 3, 5 and 6 under 1, 2, and 3 processors, respectively. For this vehicle load, engaging two processors can speed-up the simulation by 5/3 times and engaging three processors can speed-up simulation by 2 times. Fig. 3 shows the relative speed with respect to vehicle load. The relative speed (measured in terms of the speed-up factor) under different numbers of processors varies depending on the vehicle load. Under a relatively high vehicle load, more processors can result in better performance of parallel simulation.

5.2 Effect of data exchange on the execution speed

The next step is to check whether the exchange of data between instances influences performance. For this purpose, the simulation was carried out under two scenarios: one with data exchange and the other without. Real-time factor was measured as before. The total number of vehicles transferred was 3000 veh/h and 6000 veh/h, respectively, when the network was divided into two and three portions. Synchronization of time was still maintained.

The effect of data exchange is shown in Fig. 4 and Fig. 5, respectively, for simulation under two processors and three processors. Notice from Fig. 4 that there is a drop in performance under higher vehicle load. This is expected because more vehicle transfers occur along system boundaries. The influence of data exchange is inevitable in parallel simulation. From the user side, care should be taken while dividing the network to minimize vehicle transfers between regions of the network. However, the effect of data exchange is insignificant, as the proposed method uses processors on the same machine and not processors on a network.

5.3 Effect of random seeds

One run of the simulation (i.e. running under one random seed, a value used by the program to govern random processes in simulation, such as driver behavior, vehicle release and so on) provides only one outcome of the random process. To examine the effect of random seeds on simulation results, five runs were performed on a test basis. Fig. 6 and Fig. 7 show speed-up factor obtained for different outcomes of simulation under two and three processors respectively. Clearly, the variation in the speed-up factor for higher vehicle loads is higher. This variation is a result of unsteadiness seen in traffic operations under highly congested situations, and should not be associated with running the simulation in a parallel environment. Nevertheless, looking at the performance of parallel simulation under the highest vehicle load (11000 vehicles) considered, it appears that parallel simulation could at the least double the speed under two processors and triple the speed under three processors. Superlinear speeds are probably due to cache effects (Dupuis and Chopard 1998). More literature on cache effects are available (Standish 2001, Gustafson 1990).

6. Summary and conclusions

A method to use a simulation program for running parallel simulation has been described and demonstrated. The method uses running multiple instances of the program for this purpose. The idea is to divide the network into several regions and simulate under different instances of the program simultaneously, allowing transfer of vehicles at the boundaries of regions. The method is demonstrated using Paramics, using its programming interface on a multi-processor UNIX system. The simulation speed is influenced by the number of vehicles simulated at that instant. The results from applying the proposed method are encouraging, showing an increase in speed ranging from 1.50 to 2.25 times when using two processors and from 1.75 to 3.75 times when using three processors, compared with the speed of simulation without parallel execution. Future enhancements to the proposed method include addition of suitable routines to perform dynamic load balancing and inclusion of all-or-nothing and dynamic traffic assignment procedures.

References

Advanced Traffic Analysis Center (ATAC), 2001. VISSIM Hardware-In-The-Loop Simulation - NEMA TS2 Interface. Accessed from http://www.atacenter.org/ on 5 Nov 2001.

Barcelo, J., Ferrer, J. L., Garcia, D., Florian, M., Saux, E. L., 1998. Parallelization of Microscopic Traffic Simulation for ATT Systems Analysis. Edited by Marcotte, P. and Nguyen, S. Kluwer Academic Publishers, Massachusetts.

Bernauer, E., Breheret, L., Algers, S., Boero, M., Taranto, C. D., Dougherty, M, Fox, K., Gabard, J. F., 1998. Review of Micro-Simulation Models - Appendix D., Ref: SMARTEST/D3, Institute of Transportation Studies, University of Leeds, U. K.

Dupuis, A., Chopard, B., 1998. Parallel Simulation of Traffic in Geneva Using Cellular Automata, Parallel and Distributed Computing Practices. 1 (3). Nova Publishers, New York.

Cameron, G., Duncan, G., 1996, PARAMICS – Parallel Microscopic Simulation of Road Traffic. Journal of Supercomputing, 10 (1), 25-53.

Gustafson, J., 1990. Fixed Time, Tiered Memory, and Superlinear Speedup. Proceedings of the IEEE 5th Distributed Memory Computing Conference, South Carolina.

Lee, D. H., Chandrasekar, P., 2001. Customized Simulation Modeling using Paramics Application Programmer Interface. In the Proceedings of the IEEE 4th International Conference on Intelligent Transportation Systems, California.

Los Alamos National Laboratory (LANL), 2001. TRANSIMS Overview. Accessed from http://transims.tsasa.lanl.gov/ on 5 Nov 2001.

Nagel, K., Rickert, M., 1999. Dynamic Traffic Assignment on Parallel Computers in TRANSIMS. Accessed from http://www.inf.ethz.ch/~nagel/papers/ on 5 Nov 2001.

Quadstone Limited, 1999. Paramics V2.0 Benchmarks. Quadstone Limited, Edinburgh, United Kingdom.

Quadstone Limited, 2000. Paramics User Guide - Version 3.0, Quadstone Limited, Edinburgh, United Kingdom.

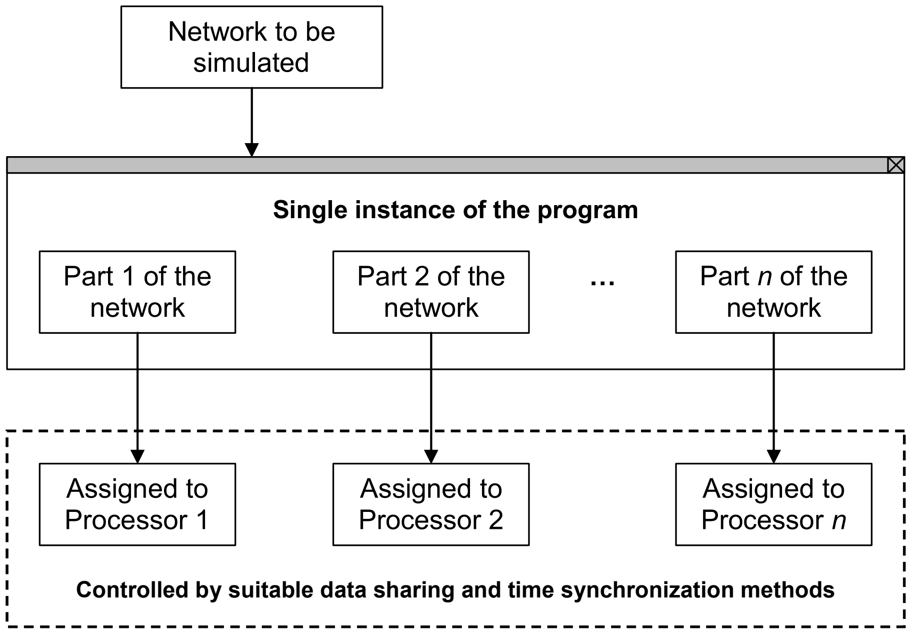
Standish, R., 2001. Introduction to High Performance Computing. Australian Centre for Advanced Computing and Communications. Accessed from http://www.ac3.com.au/edu/hpc-intro/hpc-intro.html.

Stevens, W. Richard., 1999. UNIX Network Programming – Interprocess Communications. 2nd Edition. Prentice-Hall Inc., New Jersey.

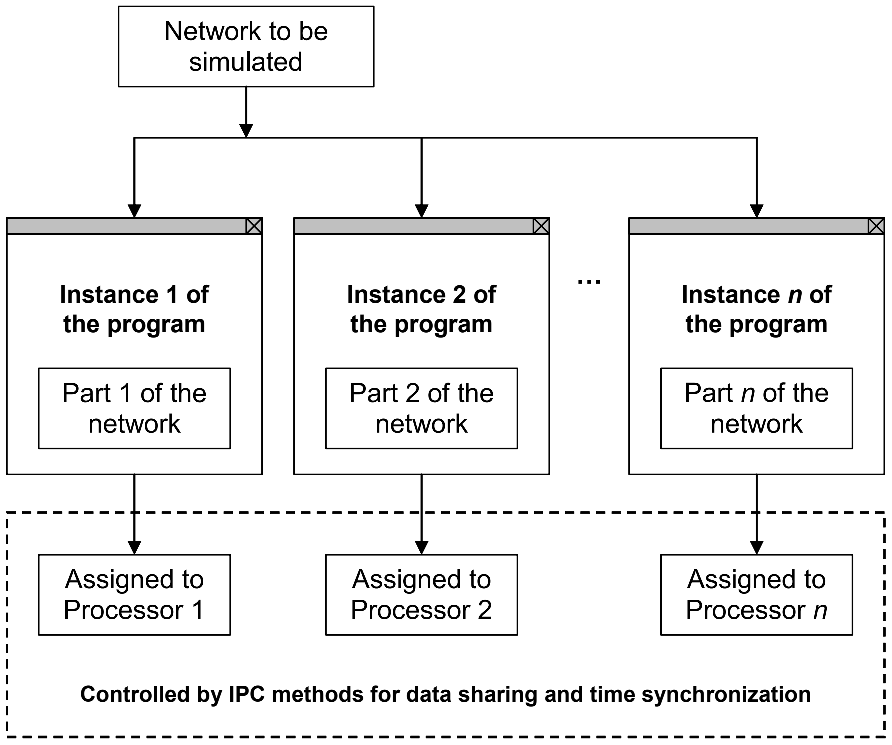
Table 1
Comparison of Performance of Parallel Simulation between Proposed and Default Methods*

Vehicle load (veh)	Speed-up factor	
	Proposed method	Default method
With 2 processors		
3000	1.53	1.30
6000	1.63	1.50
9000	1.84	1.70
11000	2.22	1.80
With 3 processors		
3000	1.86	1.70
6000	2.17	1.90
9000	2.80	2.30
11000	3.90	2.60

**The configurations of networks used in the two methods are different.*
*** Results from one run of simulation (see Fig. 6 and Fig. 7 for effects of random seeds).*
****Extracted from the graphs presented in the document (Quadstone 1999).*



(a) A typical example of conventional method
(n is the number of processors available on the machine)



Network to be simulated
Instance 1 of the program
Part 1 of the network
Instance 2 of the program
Part 2 of the network
...
Instance *n* of the program
Part *n* of the network
Assigned to Processor 1
Assigned to Processor 2
Assigned to Processor *n*
Controlled by IPC methods for data sharing and time synchronization

(b) Proposed method
(n is the number of processors available on the machine)

Fig. 1. Implementation of parallelism

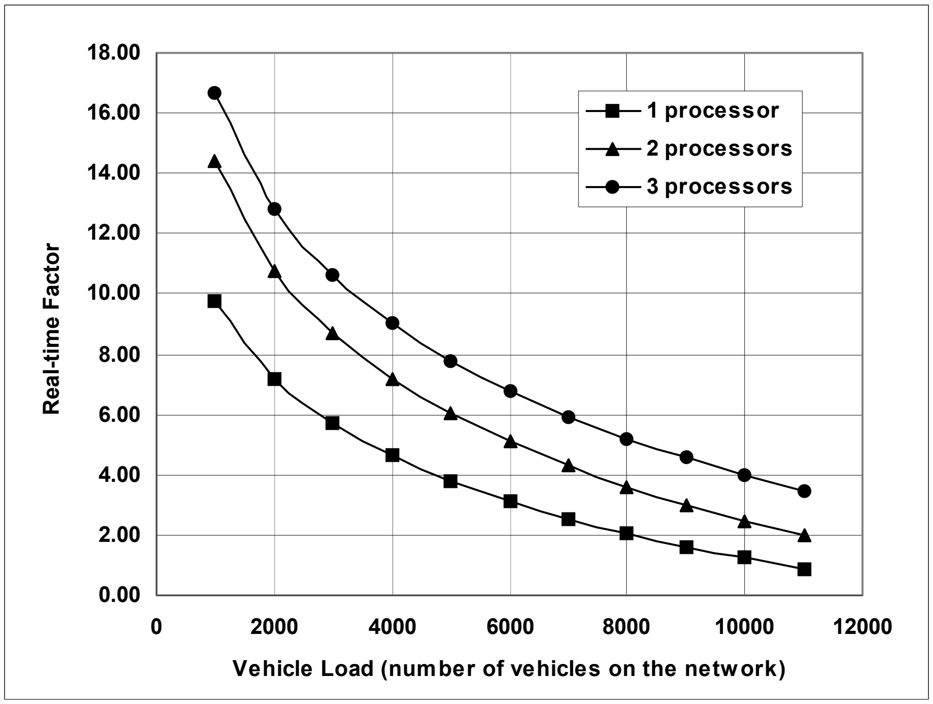


Fig. 2. Performance of parallel simulation using proposed method
(with data exchange and synchronization, measured by means of real-time factor).

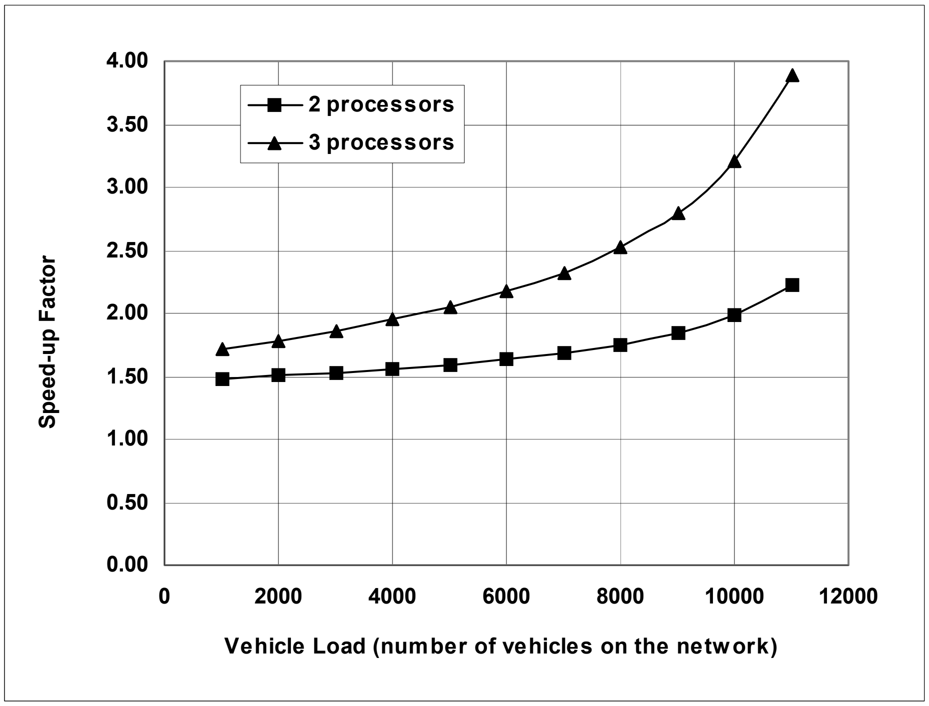


Fig. 3. Performance of parallel simulation using proposed method
with data exchange and synchronization, measured by means of speed-up factor.

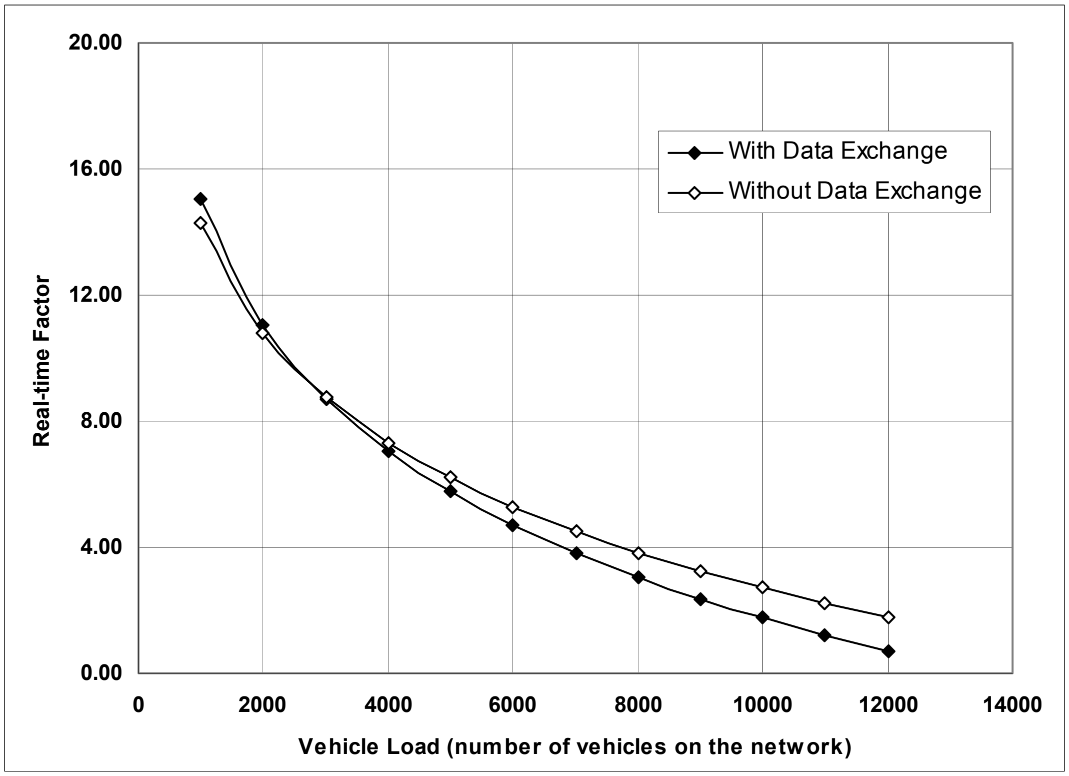


Fig. 4. Effect of data exchange on the performance of parallel simulation using proposed method under 2 processors.

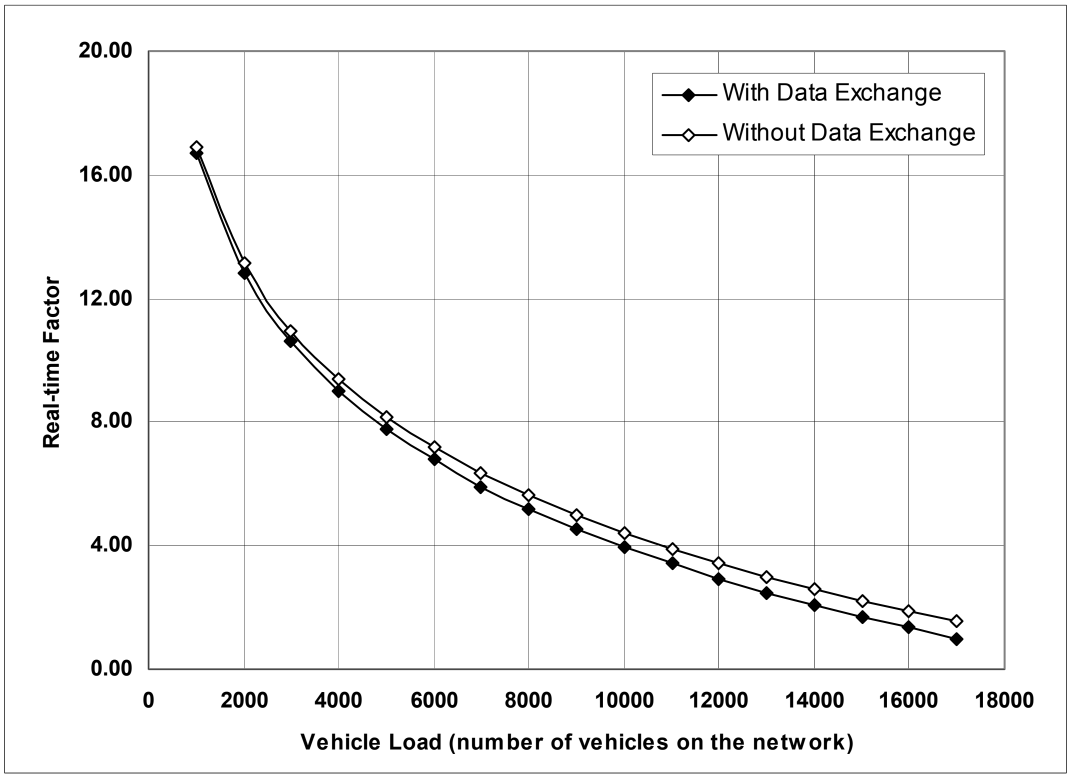


Fig. 5. Effect of data exchange on the performance of parallel simulation using proposed method under 3 processors.

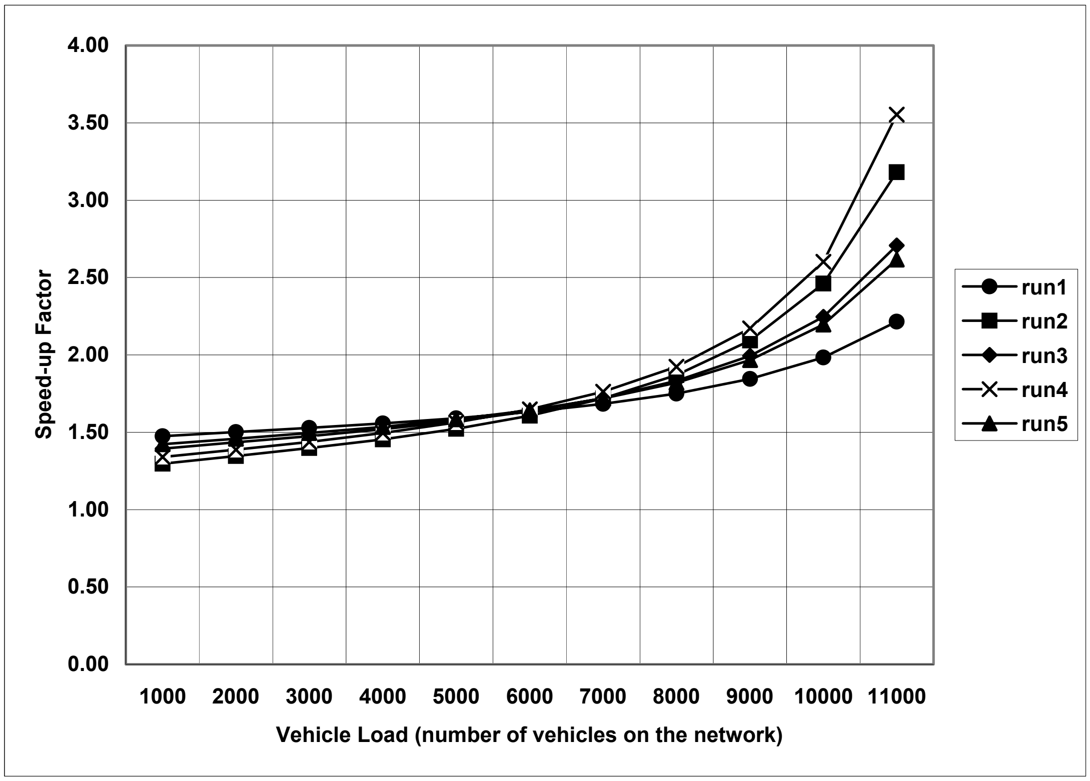


Fig. 6. Effect of random seeds (simulation under 2 processors).

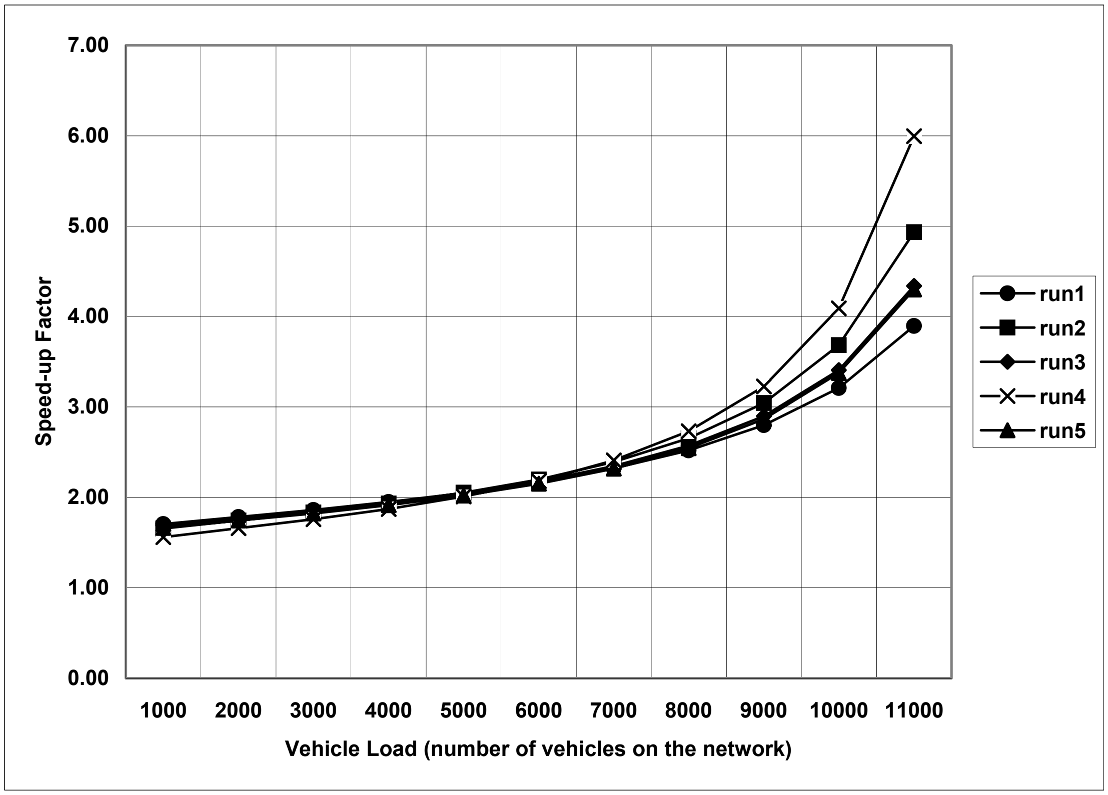


Fig. 7. Effect of random seeds (simulation under 3 processors).